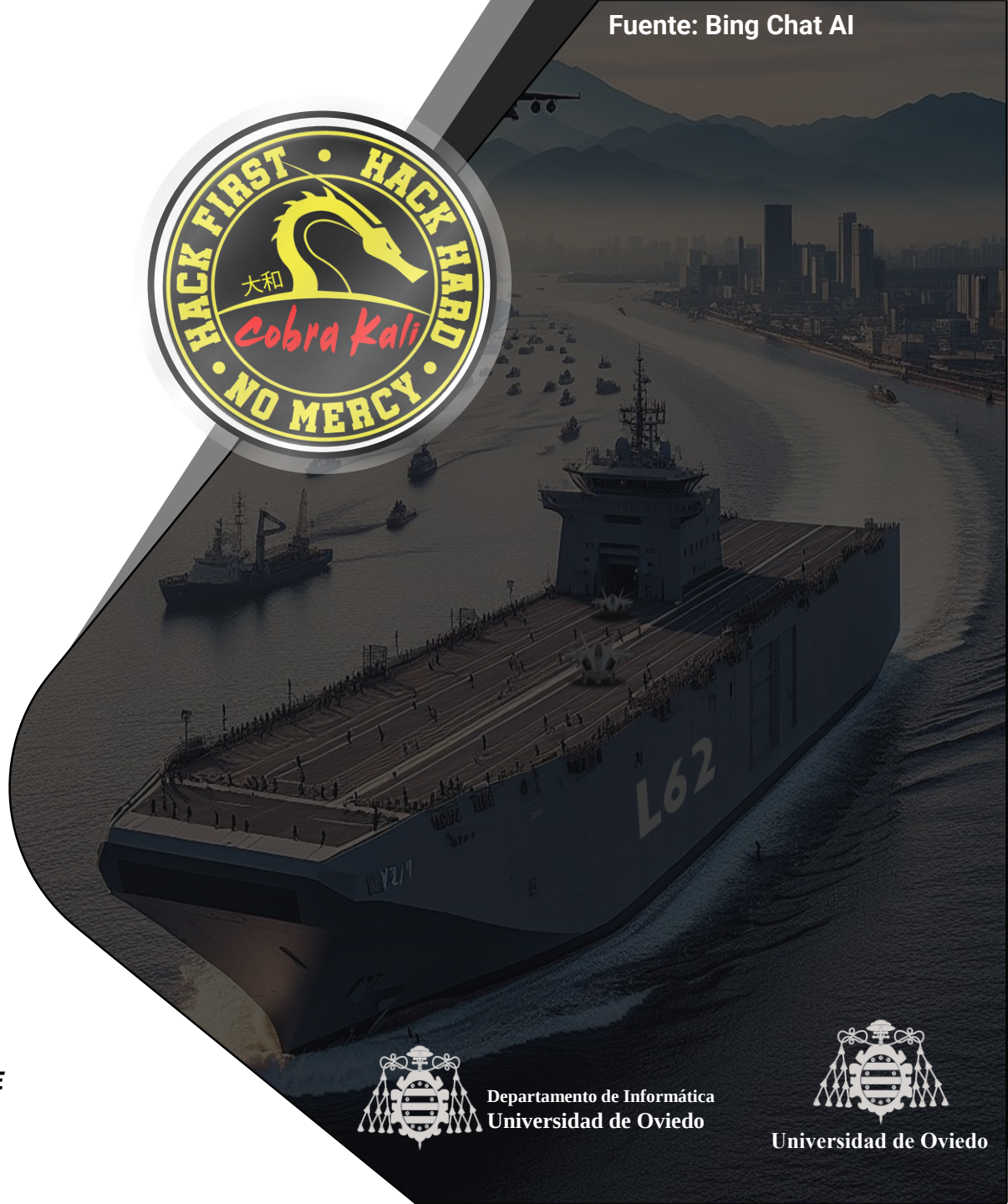
INTRODUCCIÓN A KUBERNETES

Desplegando nuestros contenedores de forma escalable

José Manuel Redondo López Proyecto "*L-62 Princesa de Asturias*" v1.2




Fuente: Bing Chat AI



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

ÍNDICE

- **Terminología y Arquitectura**
  - Conceptos vinculados con nodos Worker
  - Conceptos vinculados con nodos Master
- **Usando Kubernetes**
  - Despliegue local con Minikube
  - Nodos, Pods y deployments...en la práctica
  - Servicios y Etiquetas...en la práctica
- **Desplegando Kubernetes con código**
  - Deployments y Services
  - Volúmenes persistentes
  - ConfigMaps y Secrets
  - Network Policies
- **Escalando Aplicaciones**
  - Escalado manual de aplicaciones
  - Auto-escalado de aplicaciones
- **Para ampliar conocimientos...**
  - Herramientas complementarias
  - Más allá de la introducción

¿Qué es esta presentación?

José Manuel
Redondo López

- Una **introducción** rápida a **Kubernetes (también llamado K8s)**
  - **Sitio oficial**: https://kubernetes.io/
  - **No es un estudio en profundidad**: ¡Necesitaríamos la asignatura entera para Kubernetes! ☺
  - Contempla los **usos más típicos** de esta tecnología, incluyendo auto-escalado
  - Permite lograr **familiaridad y soltura** para luego explotarla en producción
  - Da las **bases** para un estudio más profundo de la misma si es necesario
  - Indica posibles vías de ampliación de lo impartido
- **Trabaja sobre una máquina local**
  - Pero lo impartido se puede aplicar muy fácilmente a un entorno de nube
  - La ventaja es que no hay coste monetario ☺
- **Te permite manejar lo más típico de Kubernetes, de forma sencilla y sin (demasiadas) complicaciones** ☺
- Está **enfocada al desarrollo de aplicaciones** sobre K8s
  - No se centra demasiado en conceptos de gestión de la infraestructura desplegada

< Ir al Índice

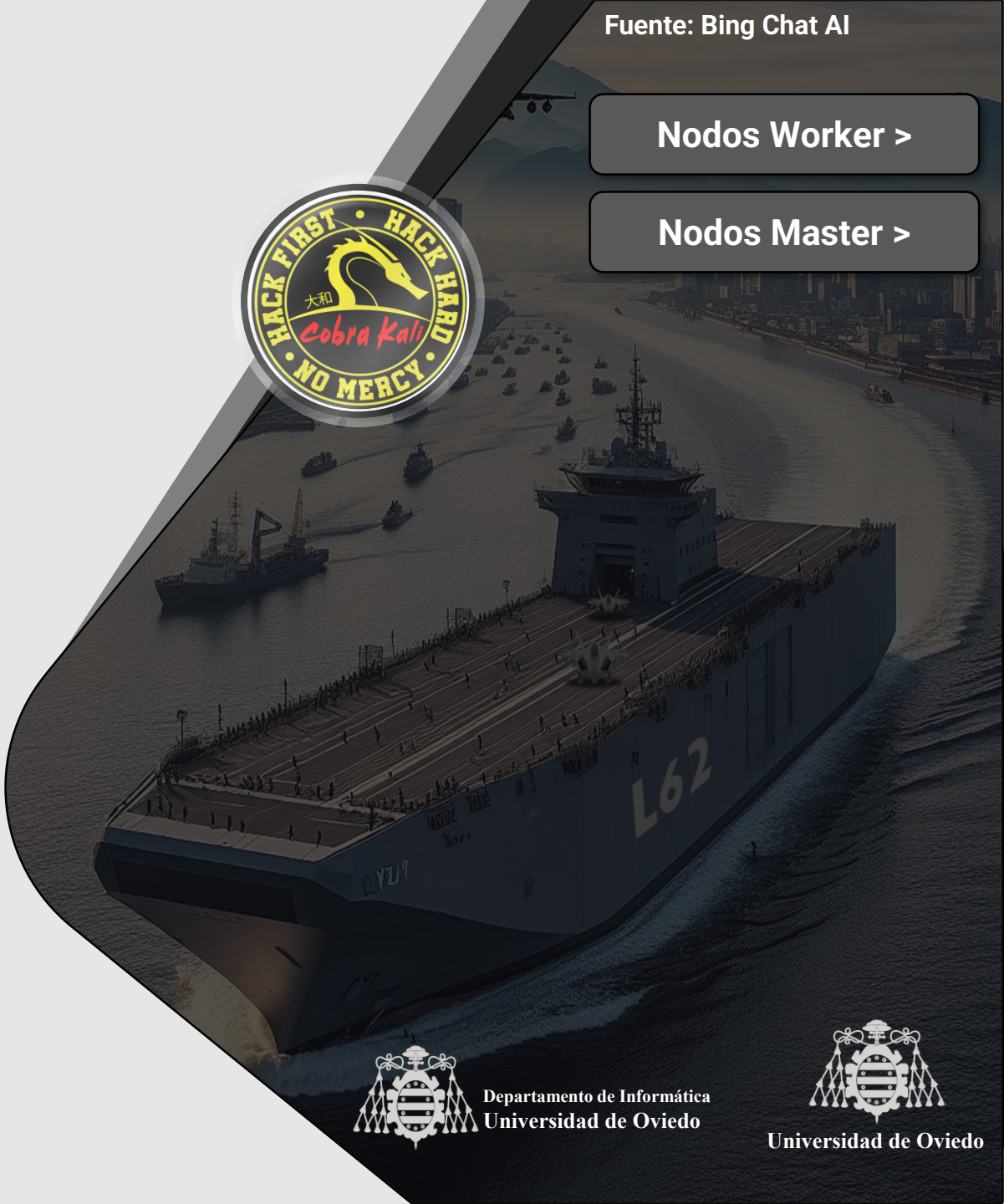
Fuente: Bing Chat AI

Nodos Worker >

Nodos Master >

Terminología y Arquitectura

Familiarizándonos con Kubernetes



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿Qué vamos a ver en este apartado?

José Manuel
Redondo López

- **Esta sección de la presentación es una toma de contacto con la tecnología Kubernetes para facilitar su estudio, que contempla**
  - Entender bien **cuál es el propósito general** de Kubernetes
  - Sus **conceptos principales**: Contenedor, pod, deployment, servicio, namespace, nodo...
  - La **arquitectura de una instalación típica** de un clúster de Kubernetes y sus distintas partes
  - **Cómo se coordinan y relacionan** todos estos elementos entre sí para cumplir su función

¿Qué es?

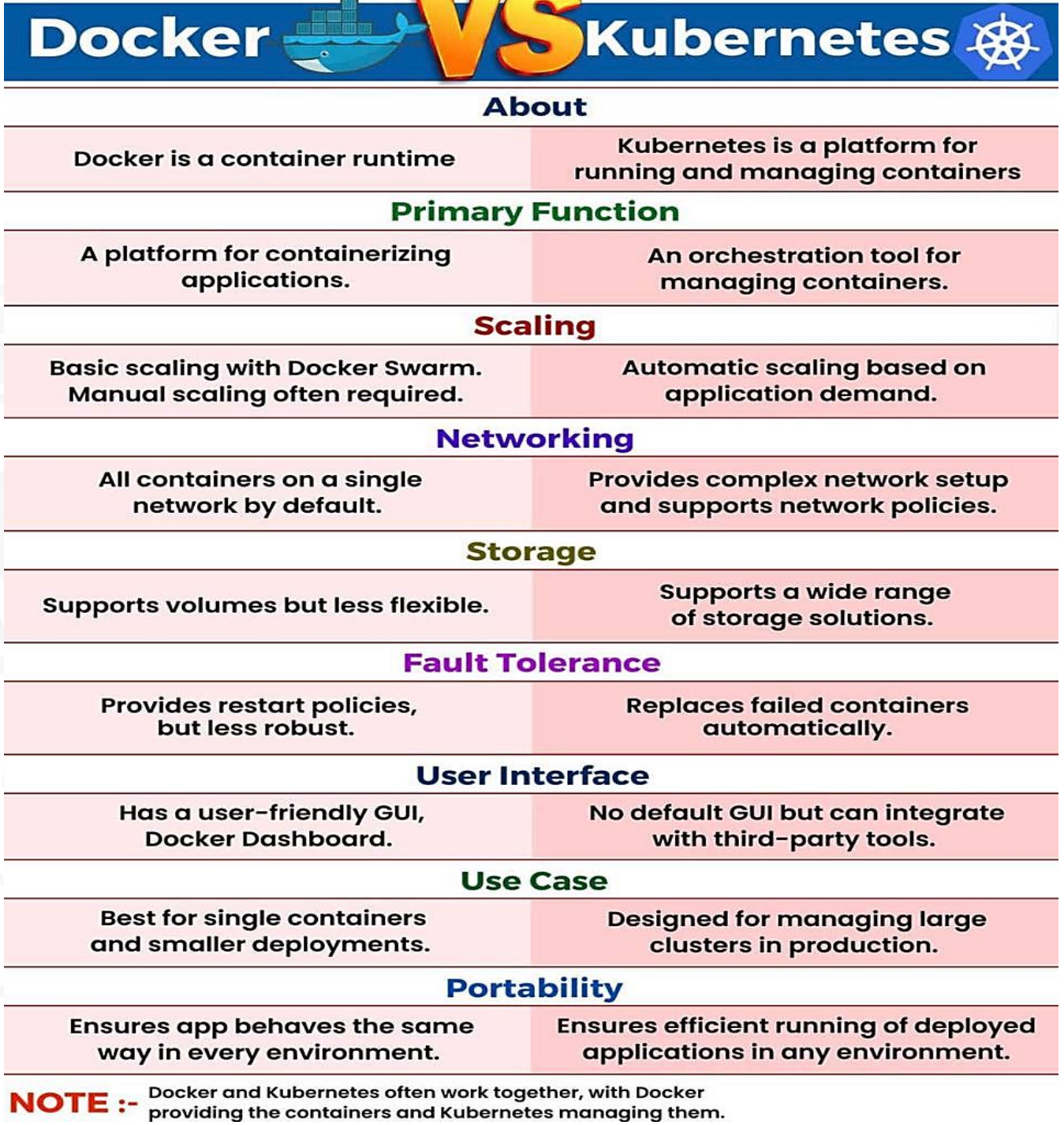
José Manuel
Redondo López

- Sistema para **automatizar el despliegue de aplicaciones en contenedores**
  - Permite el **escalado automático** de las aplicaciones desplegadas
  - Establece primitivas para construir **sistemas flexibles** y adaptados a distintas necesidades
  - También dispone de **API de programación**
  - Admite crear los **despliegues con código**
  - **Tiene una arquitectura Master-Worker**
- Diseñado por **Google**, ahora de código libre
- **Sus componentes se dividen entre los que pertenecen a Kubernetes node y a Kubernetes control plane**
- Haremos un repaso rápido de su estructura y terminología antes de ver cómo funciona

¿QUÉ ES?

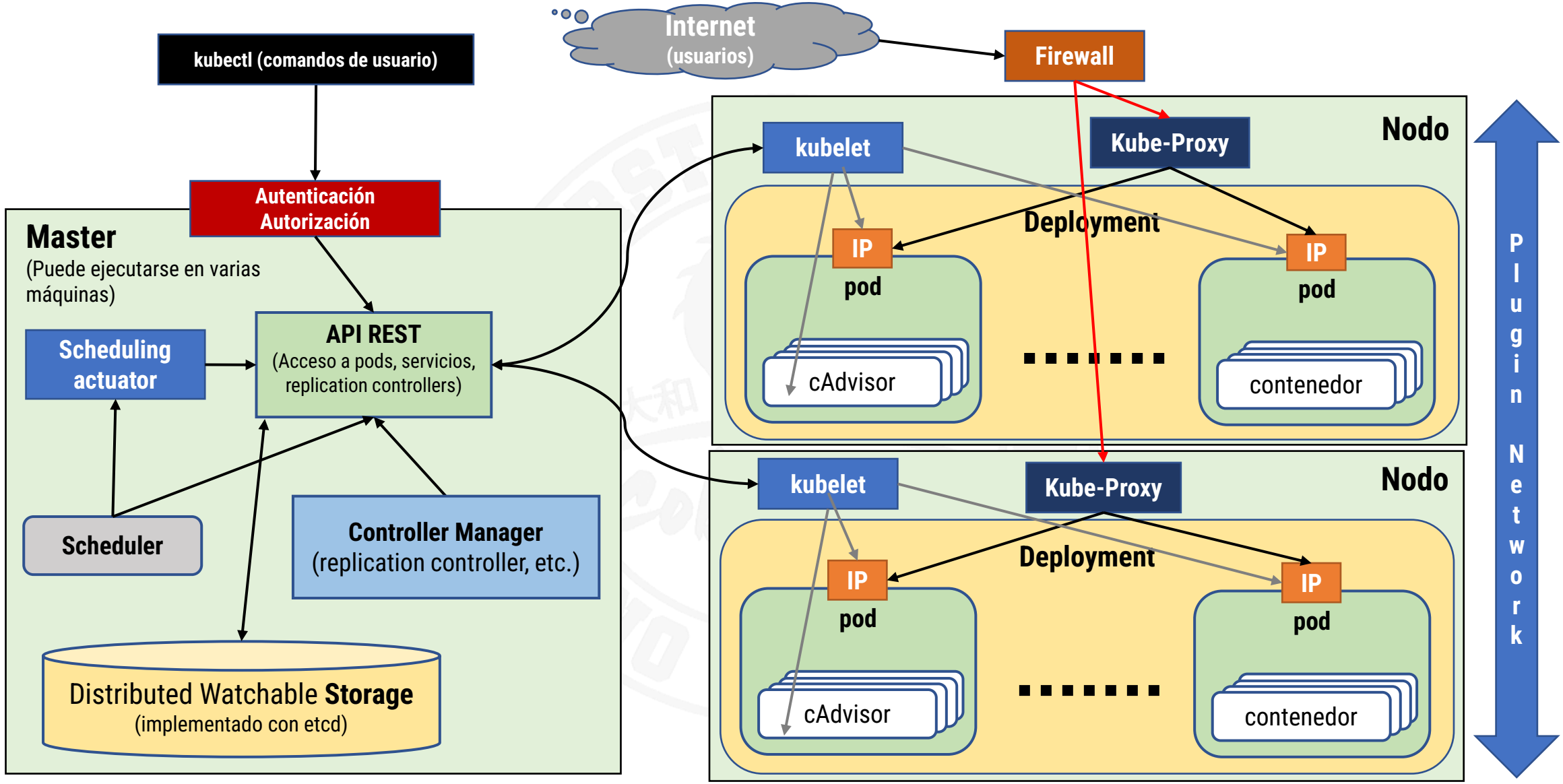
José Manuel
Redondo López

- **Kubernetes** orquesta contenedores de aplicaciones
  - Entre ellos, contenedores **Docker**
- Pero **Kubernetes** NO ES **Docker**
  - Ni ofrece las mismas funcionalidades
  - Y funciona con otros sistemas de contenedores, no solo **Docker** ☺
- Esta imagen muestra las diferencias principales
  - Fuente: https://mobile.twitter.com/paulramosm68



ARQUITECTURA COMPLETA

José Manuel
Redondo López



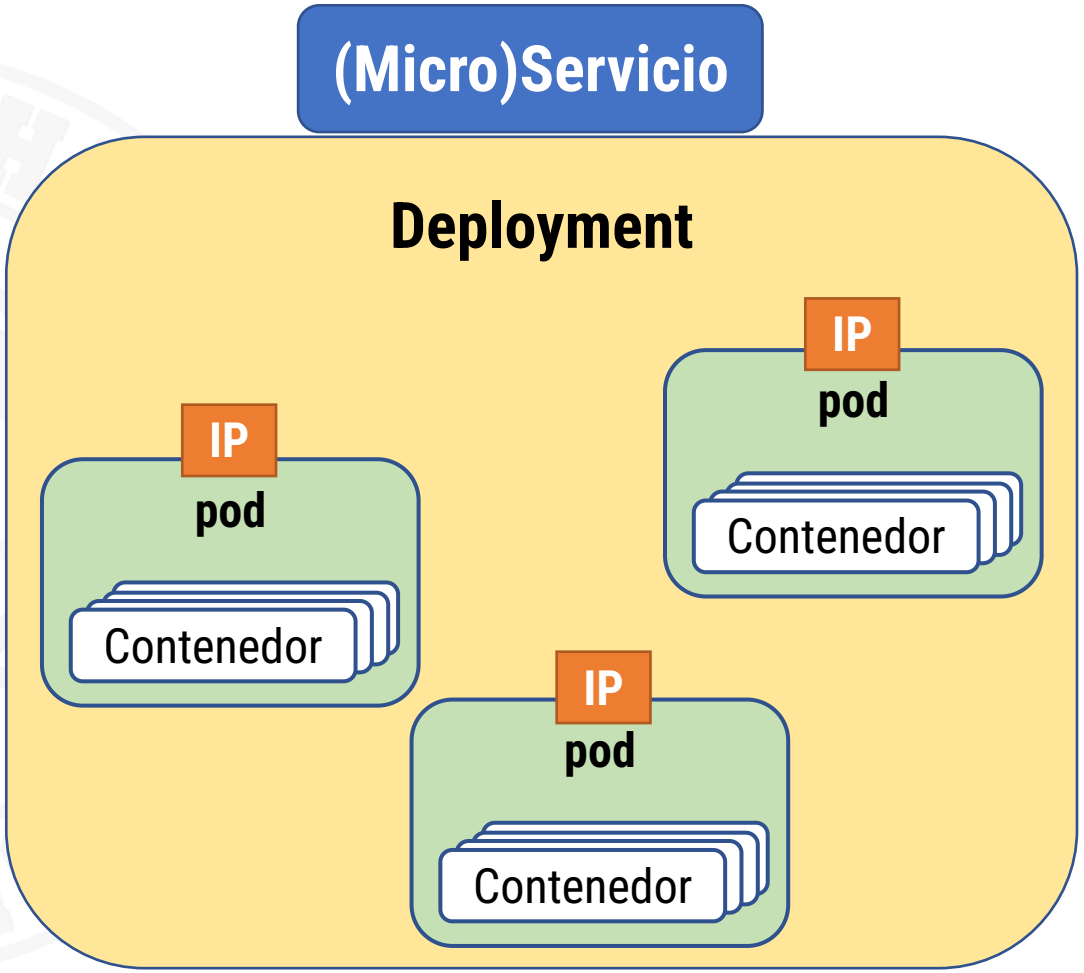
Conceptos vinculados con nodos Worker

Conceptos que hacen funcionar un clúster Kubernetes

Resumen de terminología básica que ampliaremos después

José Manuel
Redondo López

- **Contenedor**: Código de la aplicación en un entorno aislado
  - Implementado mediante tecnologías como Docker
- **Pod**: Conjunto de contenedores
  - Que **comparten red y volúmenes de almacenamiento**
  - Son volátiles
  - Se manejan desde una máquina (**nodo**)
  - Tienen una IP de pod y posibles **etiquetas**
- **Deployment**: Réplicas de un pod que se ejecutan en un clúster
  - Pueden tener **etiquetas**
- **(Micro)Servicio**: Encapsula una serie de Pods y sus políticas de acceso
  - Todo bajo una misma abstracción



CONTENEDORES

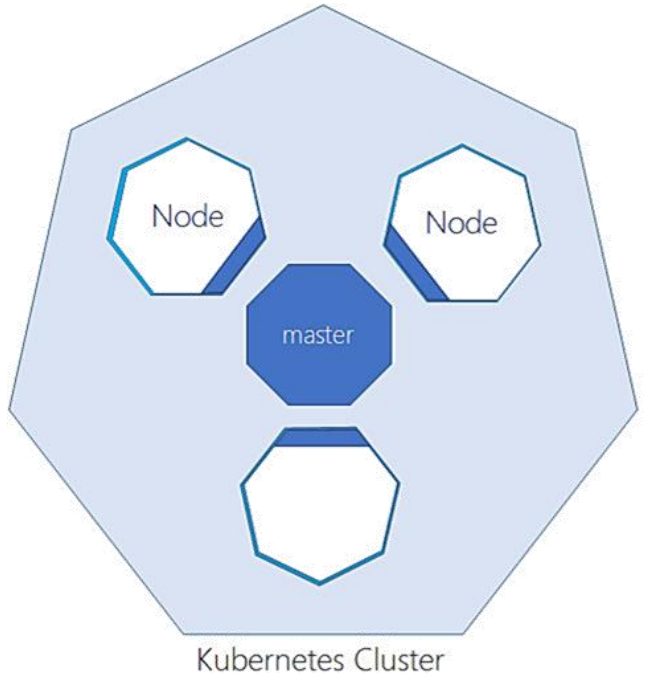
José Manuel
Redondo López

- Unidad más pequeña manejada por **Kubernetes**
- El propósito de **Kubernetes** es gestionarlos, desplegarlos y monitorizarlos
- Aunque se asociaba con **Docker**, ahora soporta otros **runtimes** de contenedores que usan la **Container Runtime Interface** (**CRI**)
  - Soporta así más tecnologías (containerd, CRI-O…)
    - Pero los despliegues anteriores a ese cambio podrían tener que adaptarse
      - https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/
  - Las imágenes construidas con Docker normalmente **siguen pudiendo usarse**
- Son los **miembros de los Pods** y, como todo contenedor de aplicaciones, tienen una aplicación, sus librerías y dependencias
- Pueden ser expuestos a clientes asignándoles una IP externa
- Más información: https://kubernetes.io/es/docs/concepts/containers/

Nodos

José Manuel
Redondo López

- **Máquinas** donde se despliegan y ejecutan los contenedores con el **runtime** de contenedores elegido
- Y ejecutan los siguientes componentes para comunicarse con el **Master** y configurar sus contenedores
  - **Kubelet**: Controla que los nodos estén en un estado correcto
  - **Kube-proxy**: Es un proxy de red y balanceador de carga
  - **cAdvisor**: Agente que monitoriza el uso de recursos
- Los describiremos en la parte práctica de la presentación

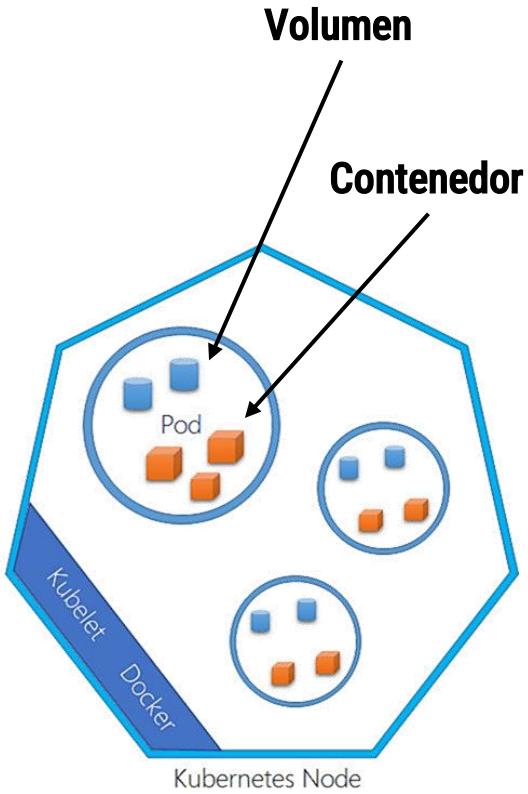


https://www.modb.pro/db/401773

Pod

José Manuel
Redondo López

- Grupo de uno o más contenedores a los que se les garantiza que se ejecutarán **en un mismo nodo o máquina**
- Cada pod tiene asignada una dirección IP única y su propio **storage namespace** (para guardar sus datos)
  - Las aplicaciones pueden usar distintos puertos asociados a esa IP sin entrar en conflicto
  - Los contenedores pueden compartir los recursos de red y almacenamiento de su pod
- Varios pods forman un **clúster de contenedores**
- Un pod se administra a través de la API de Kubernetes o bien mediante un **controller**
- **Son "mortales"**: cuando se borra un pod todo su contenido se destruye
- Más información: https://kubernetes.io/es/docs/concepts/workloads/pods/pod/

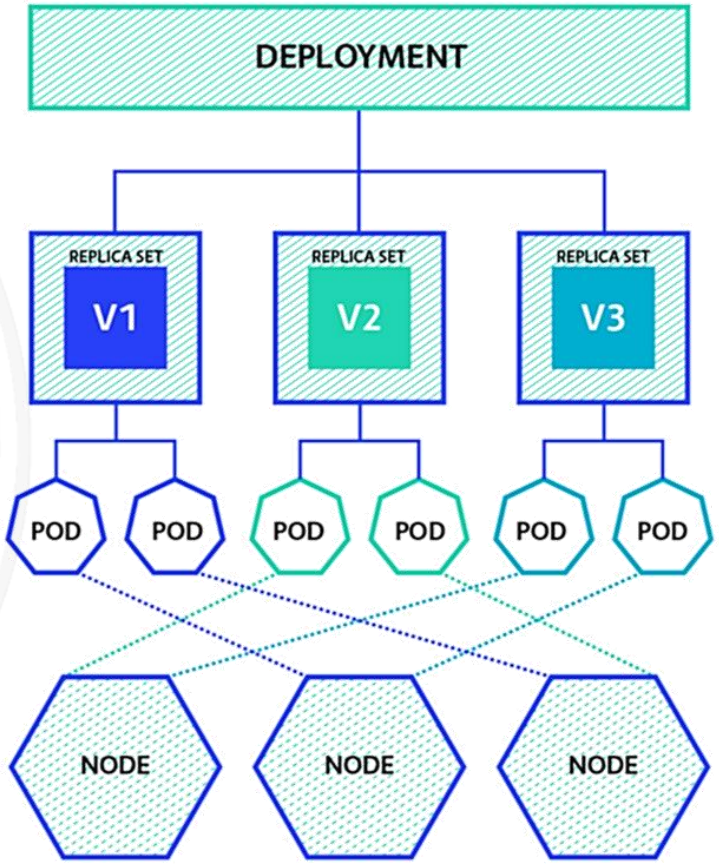


https://www.modb.pro/db/401773

DEPLOYMENTS

José Manuel
Redondo López

- **Hemos visto que un pod es "mortal" y puede ser un problema**
  - Pero con un Deployment se puede estar seguro de que **siempre habrá funcionando un número concreto de pods**
    - Número especificado por el usuario
  - Por tanto, especifica **cuántas instancias de un pod** van a ejecutarse para un determinado servicio
- **Los pods de un deployment pueden residir en distintos nodos (máquinas)**
  - Por tanto, **es una capa de abstracción más** que nos permite ejecutar pods **sin importar en qué máquina estén**
- **Más información**
  - https://kubernetes.io/docs/concepts/workloads/controllers/deployment/



https://thenewstack.io/kubernetes-deployments-work/

ETIQUETAS

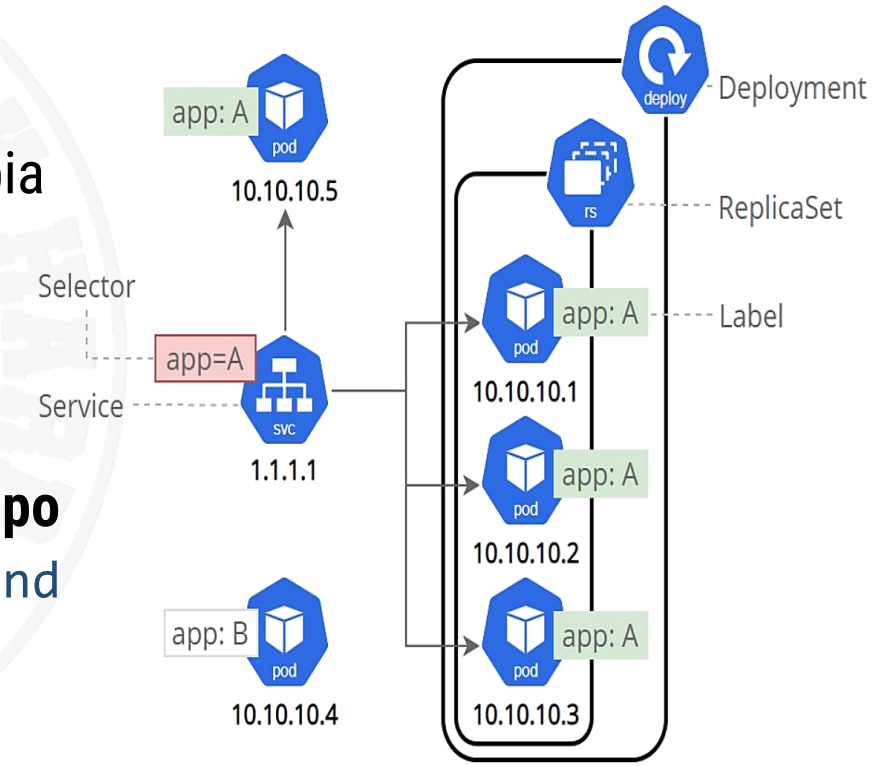
José Manuel
Redondo López

- Son pares clave-valor asociados a cualquier objeto del API de Kubernetes, como pods o nodos
- Se usan para agruparlos y hacer consultas o referenciar más fácilmente diferentes tipos de objetos. Ejemplos:
  - "**frontend**" o "**backend**" asociados a la clave "**capa**"
  - "**pruebas**" o "**produccion**" asociado a la clave "**tipo**"
- Es posible etiquetar varios pods de la infraestructura y hacer una operación sobre muchos nodos
  - Con consultas o referencias que usen las etiquetas de los pods que los contienen: `capa=frontend AND tipo=pruebas`
  - Se facilita aplicar determinadas operaciones sobre conjuntos de contenedores

SERVICIOS

José Manuel
Redondo López

- Son abstracciones que exponen conjuntos de **pods** o **deployment** como un servicio de red
- Nos **abstraen de toda la complejidad** que implica ejecutar un clúster **Kubernetes**
  - Los **pods** ejecutan contenedores, y habitualmente tienen su propia IP y nombre DNS, permitiendo el balanceo de carga entre ellos
  - Pero como los **pods** son mortales, es necesario algo que cree automáticamente nuevos si alguno finaliza: el **deployment**
  - El conjunto de **pods** que proporciona una funcionalidad (con su etiqueta asociada: "`backend`", "`frontend`"…) **cambia con el tiempo**
  - ¿Como hace **Kubernetes** para que los conjuntos de **pods** `frontend` y `backend` puedan comunicarse entre ellos, aunque haya estos cambios? **¡Gracias a los servicios!**
- Más información
  - https://kubernetes.io/docs/concepts/services-networking/service/



NAMESPACES

José Manuel
Redondo López

- **Kubernetes soporta crear varios clústeres virtuales en un mismo clúster físico**
  - Estos clústeres virtuales se llaman **espacios de nombres** (namespaces)
- **Están pensados para casos de uso con muchos usuarios distribuidos entre múltiples equipos o proyectos**
  - En clústeres pequeños (como los de esta asignatura) no tienen mucho sentido
- **Proporcionan aislamiento lógico entre clústeres**
  - Se soporta cualquier número de namespaces en un mismo clúster
  - Los nombres de los recursos **deben ser únicos** dentro de un namespace
  - Dos namespaces distintos pueden nombrar a un recurso de la misma forma
- **A pesar de este aislamiento, como veremos, recursos en distintos namespaces pueden comunicarse entre sí**

NAMESPACES

José Manuel
Redondo López

- **Kubernetes tiene tres namespaces inicialmente**
  - `default`: **Espacio de nombres por defecto** para los objetos que no especifican ningún espacio de nombres al crearse
  - `kube-system`: Espacio de nombres para objetos creados por el sistema de Kubernetes
  - `kube-public`: Este espacio de nombres se crea de forma automática y legible por todos los usuarios
    - Incluyendo aquellos no autenticados
    - Se reserva para uso interno del clúster, en caso de que algunos recursos necesiten ser visibles y legibles de forma pública
- **Cualquier operación que veremos posteriormente puede asignarse a un namespace con**
  - `kubectl --namespace=<nombre> ... (resto de la operación)`

CONTROLLERS

José Manuel
Redondo López

- **Gestionan una serie de pods y sirven para dejarlos en un estado determinado según su propósito**
  - Por ejemplo, los **replication controllers** permiten controlar cuantas copias de un pod existen para temas de escalado y redundancia
  - Otro ejemplo es el **job controller**, que ejecuta los pods asociados hasta que la tarea que ejecutan termina
    - Adecuado para trabajos batch
- **Las etiquetas de los pods se usan para asociar pods a controllers cuando estos se definen**
- **Más información:** https://kubernetes.io/es/docs/concepts/workloads/controllers/

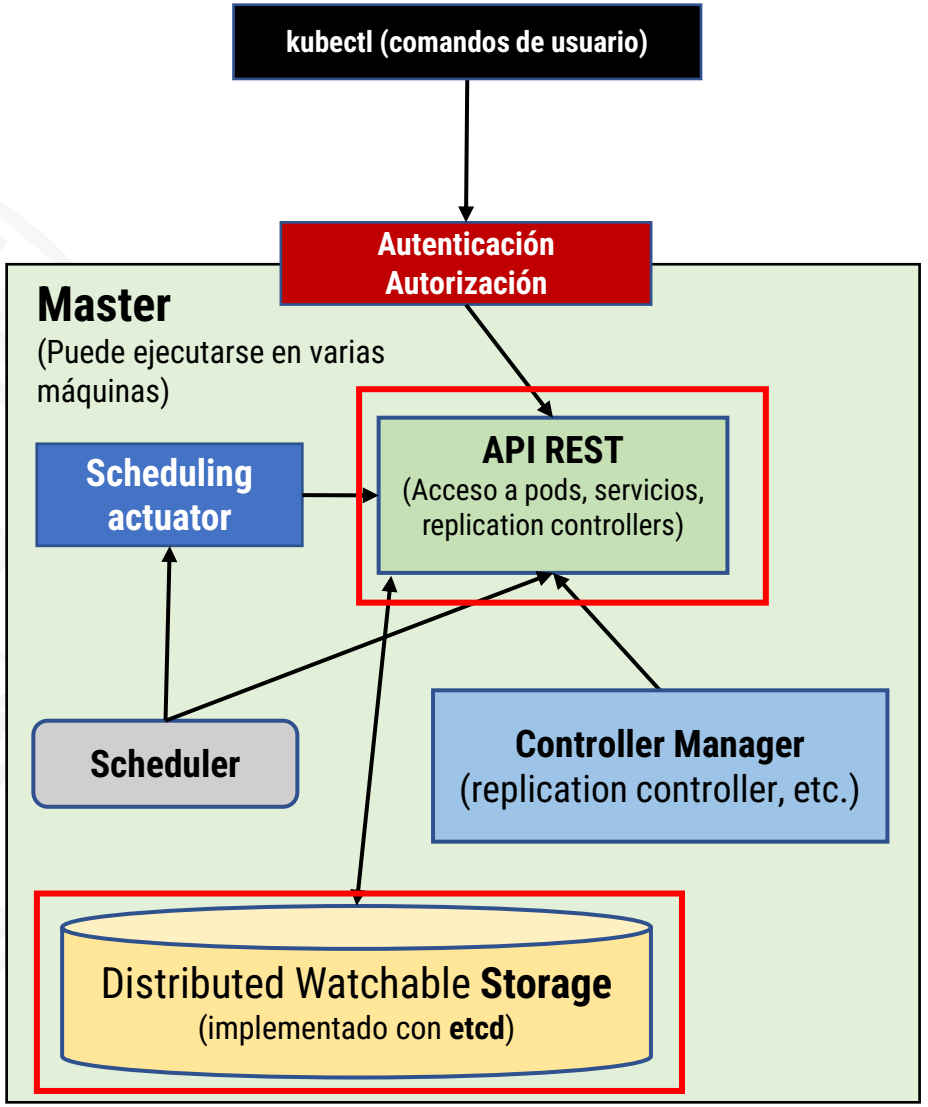
Conceptos vinculados con nodos Master

Conceptos que permiten controlar un clúster Kubernetes

KUBERNETES CONTROL PLANE

José Manuel
Redondo López

- Es el **Kubernetes Master**: Unidad de control principal de un clúster
  - Gestiona su carga y comunicaciones
- Tiene los siguientes componentes, cada uno en su propio proceso
- Se pueden ejecutar en un solo **Master** o en varios (alta disponibilidad):
  - `etcd`: Almacén distribuido de la configuración del clúster
    - Pares clave-valor
  - **API REST**: Ofrece el API de Kubernetes mediante JSON a través de HTTP
    - Tanto a clientes externos como a componentes internos
    - El API modifica la configuración que guarda **etcd**
    - Por lo que gracias a ella los clientes **pueden cambiar la configuración de todo el clúster**

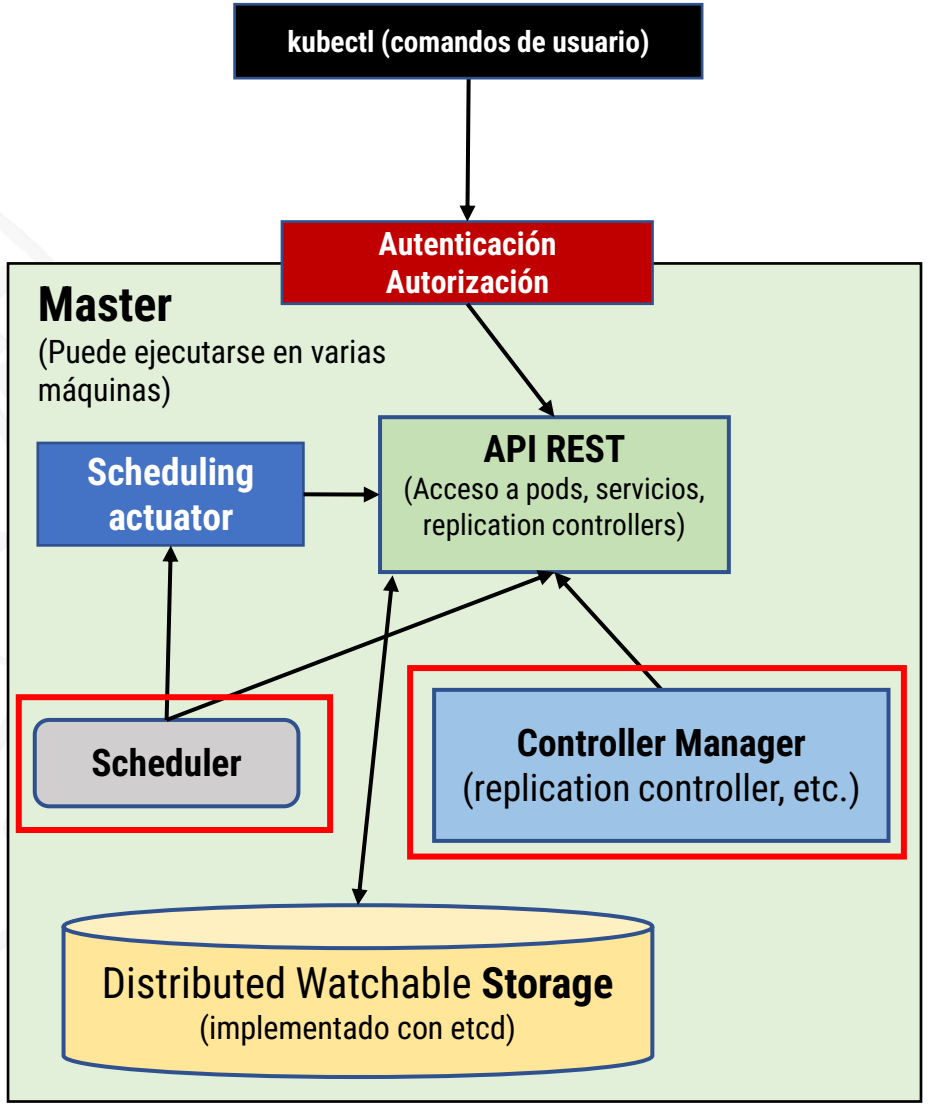


Kubernetes control plane

José Manuel
Redondo López

- **Scheduler**: Selecciona en que nodo se ejecuta un pod según los recursos disponibles
  - Controla el uso de los recursos de los nodos
  - Para no asignar nuevos pods a aquellos que tengan ya una alta carga de trabajo
  - El usuario puede indicar directivas que controlan esta asignación y los **límites de uso**
- **Controller manager**: Ejecuta los controller, que a su vez usan la API para hacer su trabajo

- **Muchos de estos componentes hacen posible las órdenes que veremos a continuación vía `kubectl`**
  - Los usaremos, pero no directamente
  - Esta sección es solo una introducción "para que os suenen los términos"
  - Nos abstraeremos todo lo posible de estos conceptos para que el resto del curso sea eminentemente práctico



¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

- *¿Entiendes para que sirve básicamente **Kubernetes** y su estrecha relación con los contenedores de aplicaciones?*
- *¿Recuerdas cómo es la arquitectura básica de un clúster **Kubernetes** instalado?*
- **Conceptos de los nodos Worker**
  - *¿Entiendes qué es un contenedor, su relación con temas pasados, y qué es un **pod** de contenedores?*
  - *¿Puedes entender la relación entre **pods** y **deployments**?*
  - *¿Comprendes la relación entre **deployments** y (micro)servicios?*
  - *¿Has entendido el tipo de separación de clústeres que proporcionan los **namespaces**?*
  - *¿Comprendes para qué sirven las etiquetas?*
  - *¿Recuerdas qué es un nodo?*
  - *¿Entiendes bien qué significa que un **pod** sea "mortal" y que elementos tiene **Kubernetes** para solventar los problemas que eso podría causar?*
- **Conceptos de los nodos Master**
  - *¿Entiendes que cuando usas un clúster **Kubernetes**, están interactuando con uno de sus nodos **Master**?*
  - *¿Eres consciente de que toda la complejidad subyacente de la gestión de contenedores y las tecnologías necesarias está abstraída gracias a esta arquitectura?*

< Ir al Índice

Fuente: Bing Chat AI

Despliegue >

Pods, deployments...>

Servicios, labels... >

Usando Kubernetes

Una primera aproximación práctica a la tecnología

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿Qué vamos a hacer?

José Manuel
Redondo López

- **Con Kubernetes podemos llevar el manejo de contenedores Docker (y de otras tecnologías) a otro nivel**
  - Pasamos de manejar contenedores sueltos a poder ejecutarlos en varias máquinas y escalarlos para distribuir la carga dinámicamente
- **Dividiremos la presentación en 4 partes**
  1. **Despliegue local** con Minikube y **ver sus conceptos en la práctica** (esta sección)
  2. Uso de servicios y etiquetas, empleando pods, nodos y servicios, tanto vía comando como vía fichero de código
  3. Escalado de aplicaciones
  4. Auto-escalado para responder a un aumento/disminución de peticiones
- **Con ello, conseguiremos entender el funcionamiento base de la tecnología y ser capaz de usarla en escenarios típicos**

Despliegue local con Minikube

Conceptos básicos y nuestro primer despliegue

¿Orquestación?

José Manuel
Redondo López

- **Los contenedores nos proporcionan un medio de hacer despliegues de aplicaciones idénticos en desarrollo y producción**
  - De forma sencilla, automática y eficiente
  - Lo vimos con Docker Compose
- **Pero esto en la práctica tiene un límite**
  - Solo funciona adecuadamente con un nº pequeño de contenedores
- **¡Los sistemas reales pueden estar desplegados en cientos o miles de contenedores!**
  - Google, por ejemplo, usa un sistema similar a Kubernetes
  - **No se puede manejar este volumen de entidades manualmente**: necesitamos un orquestador
  - Nos cubre necesidades como la planificación, el balanceo de carga, distribución entre las máquinas virtuales o físicas disponibles, etc.
  - Desplegar un sistema para atender un enorme nº de peticiones con mucho menos esfuerzo

MINIKUBE

José Manuel
Redondo López

- **Kubernetes** se usa habitualmente en entornos de **cloud computing**
- Pero **minikube** es una herramienta que nos permite ejecutar **Kubernetes** en una máquina local, ideal para pruebas y lograr "soltura"
- Usaremos una distribución **Ubuntu Server 18.04**
  - **Necesitamos al menos 2 Gb de RAM y 2 cores** asignados a una MV
- Al terminar el proceso de instalación nuestro sistema debería tener
  - **Un hipervisor** (VirtualBox, VMware Fusion o HyperKit en MacOS, VirtualBox o KVM en Linux, VirtualBox o Hyper-V en Windows)
    - **O ninguno**, si se arranca con `--vm-driver=none` o con `--vm-driver=docker` (nuestra mejor opción)
      - Ejecuta los contenedores directamente sobre el SO, no sobre una VM en el host
      - Necesita Linux y Docker ya instalado
      - Esta será la opción que usemos puesto que, aunque puede proporcionar menos aislamiento y estabilidad, consume muchos menos recursos
  - `kubectl` (Kube control tool) y `minikube`

Instalación de Minikube

José Manuel
Redondo López

- Primero instalaremos **Docker**
- Y luego seguimos esta guía para **Ubuntu Server 18.04**
  - https://computingforgeeks.com/how-to-install-minikube-on-ubuntu-debian-linux/
  - Pero ¡**OJO**! No instalaremos ningún hipervisor (usaremos directamente **Docker**)
  - Primero `minikube` y luego `kubectl`
  - **Debe ser la versión 1.23 o inferior**
  - Instalamos `conntrack`: `sudo apt install conntrack`
  - **El primer arranque de `minikube` debe hacerse con `--vm-driver=docker`** para usar **Docker** como base
  - Comprobamos que funciona correctamente
    - `sudo kubectl cluster-info`
    - `sudo kubectl config view`

```
ssiuser@vagrant:~$ minikube version
minikube version: v1.23.1
commit: 84d52cd81015effbdd40c632d9de13db91d48d43
ssiuser@vagrant:~$ minikube start --vm-driver=docker start
😄  minikube v1.23.1 on Ubuntu 18.04 (vbox/amd64)
✨  minikube 1.26.0 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.26.0
💡  To disable this notice, run: 'minikube config set WantUpdateNotification false'

✨  Using the docker driver based on user configuration
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
💾  Downloading Kubernetes v1.22.1 preload ...
    > preloaded-images-k8s-v13-v1...: 511.84 MiB / 511.84 MiB  100.00% 14.38 Mi
    > gcr.io/k8s-minikube/kicbase: 355.39 MiB / 355.40 MiB  100.00% 6.67 MiB p/
🔥  Creating docker container (CPUs=2, Memory=2200MB) ...
🐳  Preparing Kubernetes v1.22.1 on Docker 20.10.8 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass

❗  /usr/local/bin/kubectl is version 1.24.3, which may have incompatibilites with Kubernetes 1.22.1.
    ▪ Want kubectl v1.22.1? Try 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Uso básico de kubectl

José Manuel
Redondo López

- `kubectl` **es un programa en línea de comandos que nos permite interactuar con nuestro clúster Kubernetes, desplegando aplicaciones en el mismo**
  - Hemos visto que la arquitectura de Kubernetes está compuesta por máquinas Master y Worker llamadas nodos
  - El Master decide qué se ejecuta en cada nodo Worker
    - Lo que incluye tareas programadas o aplicaciones web desplegadas en contenedores
  - Con `minikube` **tenemos ambos elementos funcionando en la misma máquina** física o nodo
    - Se hace para ahorrar recursos
  - ¿Cómo sé qué nodos hay disponibles para trabajar? `kubectl get nodes`
    - Inicialmente con `minikube` vamos a tener un solo nodo (suficiente para nuestras pruebas)
- **Cheatsheet de todas las opciones de** `kubectl`
  - https://kubernetes.io/docs/reference/kubectl/cheatsheet/

```
redondo@miw:~$ sudo kubectl get nodes
NAME    STATUS    ROLES     AGE    VERSION
miw     Ready     master    17m    v1.18.3
```

DESPLEGANDO NUESTRA PRIMERA APLICACIÓN

José Manuel
Redondo López

- Hecho esto, ya podemos desplegar nuestra primera aplicación en **Kubernetes**, usando una imagen de **Docker** que la contenga
- Usaremos la imagen oficial `httpd:2.4` del **Docker Hub** y la llamamos `kubesample1`

```
sudo kubectl create deployment kubesample1 --image=httpd:2.4
```

```
redondo@miw:~$ sudo kubectl create deployment kubesample1 --image=httpd:2.4
deployment.apps/kubesample1 created
```

- Es un simple servidor web **Apache** con una página por defecto
- Podemos comprobar que ha funcionado con: `kubectl get deployments`
- Si todo es correcto, deberíamos ver algo como esto:

```
redondo@miw:~$ sudo kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
kubesample1   1/1     1            1           59s
```

Nodos, Pods y deployments...en la práctica

Mostrando los conceptos teóricos en un despliegue real

¿Qué ha pasado? Nodos

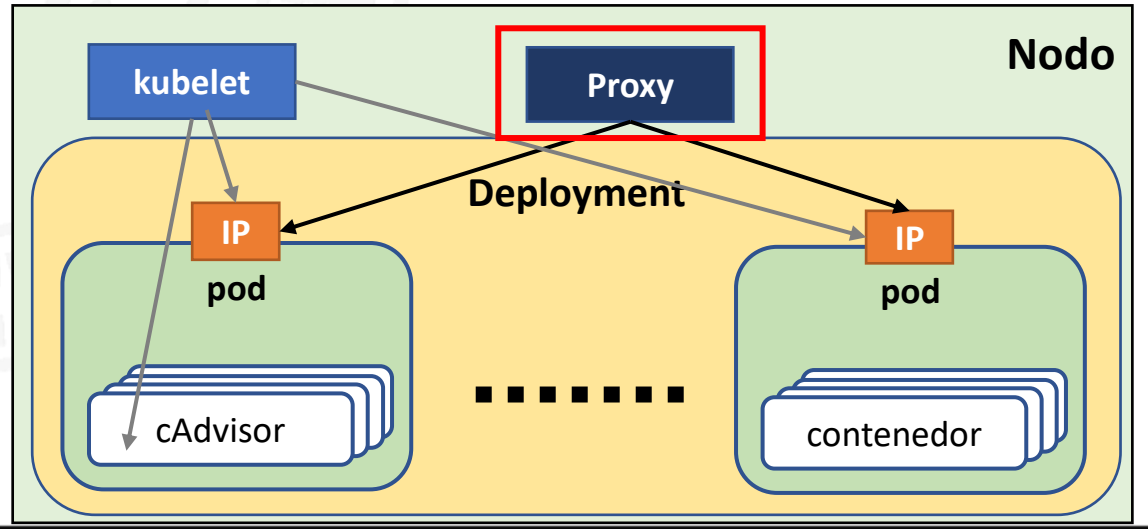
José Manuel
Redondo López

- **Para desplegar nuestra aplicación en el clúster de antes, Kubernetes ha hecho lo siguiente**
  - **Ha buscado un nodo adecuado** para ejecutar la instancia de la aplicación
    - En nuestro caso solo había uno, en despliegues en nube haría ese trabajo por nosotros
  - **Se ha programado la ejecución** de la aplicación dentro de ese nodo (que en nuestro caso fue inmediata)
  - Se ha **creado automáticamente un pod** para el contenedor que hemos arrancado
  - Se ha **creado automáticamente un deployment** para esos pods
  - **Aún no se ha creado un servicio** asociado al deployment (ya lo haremos)
  - **Se ha configurado el clúster** para rearrancar la instancia de la aplicación en un nuevo nodo (si lo hubiera) si fuese necesario
  - A partir de nuestra imagen Docker…¡han ocurrido muchas cosas automáticamente!
  - Como vemos, **nos abstrae de MUCHA complejidad**

¿QUÉ HA PASADO? POD

José Manuel
Redondo López

- El pod es la **unidad de despliegue más pequeña**
  - **No veremos un contenedor de la imagen desplegada nunca, veremos su pod**
  - Puede estar formado por **varios contenedores** (en este caso solo habrá uno de momento)
  - Por tanto, la aplicación arrancada **no** va a un nodo (máquina) **sino a un pod dentro de ese nodo**
  - Cada pod ejecuta una red privada aislada
  - A esa red no se puede acceder desde otros elementos de Kubernetes, como otros pods
- **¿Podemos usar la aplicación desplegada desde fuera?**
  - ¡Sí! pero tenemos que configurar cómo
  - Una de esas formas es **usando un proxy**



Accediendo a la aplicación desplegada

José Manuel
Redondo López

- **El servidor del API de Kubernetes crea un endpoint para cada pod con un nombre**
  - Para saber los nombres de los pods disponibles: `sudo kubectl get pods` **(1)**
  - De momento solo tenemos uno
- **Podemos acceder a los contenidos mediante un proxy**
  - Desde **otra terminal** (ALT+F2..9) ahora podemos ejecutar: `sudo kubectl proxy` **(2)**
  - Esto expondrá kubectl como una API a la hacer peticiones mediante HTTP
  - Por ejemplo: `http://localhost:8001/version` (equivalente a `kubectl version`)
- **¡Pero hay más métodos! (3)**

1

```
redondo@miw:~$ sudo kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
kubesample1-8649c7d75c-4qpjv   1/1     Running   0          89s
```

```
operario@operario-VirtualBox:~$ sudo kubectl proxy
Starting to serve on 127.0.0.1:8001
```

2

```
redondo@miw:~$ curl http://localhost:8001
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/apis/admissionregistration.k8s.io",
    "/apis/admissionregistration.k8s.io/v1",
    "/apis/admissionregistration.k8s.io/v1beta1",
    "/apis/apiextensions.k8s.io",
    "/apis/apiextensions.k8s.io/v1",
    "/apis/apiextensions.k8s.io/v1beta1",
    "/apis/apiregistration.k8s.io",
    "/apis/apiregistration.k8s.io/v1",
    "/apis/apiregistration.k8s.io/v1beta1",
    "/apis/apps",
    "/apis/apps/v1",
    "/apis/authentication.k8s.io",
    "/apis/authentication.k8s.io/v1",
    "/apis/authentication.k8s.io/v1beta1",
    "/apis/authorization.k8s.io",
    "/apis/authorization.k8s.io/v1",
```

3

ACCEDIENDO A LA APLICACIÓN DESPLEGADA

José Manuel
Redondo López

- **No obstante, el nombre de un pod por defecto es poco manejable**
  - Una buena opción es guardarlo en una variable **(1)**
  - Y usar así el API vista para hacer una petición de información acerca de un pod: `curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME` **(2)**
- **Nos da un JSON con toda la información del pod creado**
  - Y también nos abre la opción de programar contra un clúster Kubernetes para su manejo
  - No entraremos en este apartado
  - Más información: https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/

```
redondo@miw:~$ POD_NAME=kubesample1-8649c7d75c-4qpjv
redondo@miw:~$ echo $POD_NAME
kubesample1-8649c7d75c-4qpjv
```

1

```
redondo@miw:~$ curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME
{
  "kind": "Pod",
  "apiVersion": "v1",
  "metadata": {
    "name": "kubesample1-8649c7d75c-4qpjv",
    "generateName": "kubesample1-8649c7d75c-",
    "namespace": "default",
    "selfLink": "/api/v1/namespaces/default/pods/kubesample1-8649c7d75c-4qpjv",
    "uid": "99e92d27-326e-4cc0-aff2-ffaea82c069a",
    "resourceVersion": "651",
    "creationTimestamp": "2020-07-15T05:37:42Z",
    "labels": {
      "app": "kubesample1",
      "pod-template-hash": "8649c7d75c"
    },
    "ownerReferences": [
      {
        "apiVersion": "apps/v1",
        "kind": "ReplicaSet",
        "name": "kubesample1-8649c7d75c",
```

2

ACCEDIENDO A LA APLICACIÓN DESPLEGADA

José Manuel
Redondo López

- Con la aplicación en marcha, una de las cosas que podemos necesitar es saber **cómo** está realizándose el servicio
- podemos consultar sus **logs** con: `kubectl logs $POD_NAME` **(1)**
- Pero olvidemos que estamos trabajando con contenedores
  - Podemos interactuar con ellos como ya sabemos
  - Por ejemplo, consultando el valor de sus variables de entorno: `kubectl exec $POD_NAME -- env` **(2)**

```
}redondo@miw:~$ sudo kubectl logs $POD_NAME
AH00558: httpd: Could not reliably determine the server's fully qualified domain
 name, using 172.17.0.3. Set the 'ServerName' directive globally to suppress thi
s message
AH00558: httpd: Could not reliably determine the server's fully qualified domain
 name, using 172.17.0.3. Set the 'ServerName' directive globally to suppress thi
s message
[Wed Jul 15 05:37:43.641812 2020] [mpm_event:notice] [pid 1:tid 140062043010176]
 AH00489: Apache/2.4.43 (Unix) configured -- resuming normal operations
[Wed Jul 15 05:37:43.643059 2020] [core:notice] [pid 1:tid 140062043010176] AH00
094: Command line: 'httpd -D FOREGROUND'
```

1

```
redondo@miw:~$ sudo kubectl exec $POD_NAME -- env
PATH=/usr/local/apache2/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/s
bin:/bin
HOSTNAME=kubesample1-8649c7d75c-4qpjv
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
HTTPD_PREFIX=/usr/local/apache2
HTTPD_VERSION=2.4.43
HTTPD_SHA256=a497652ab3fc81318cdc2a203090a999150d86461acff97c1065dc910fe10f43
HTTPD_PATCHES=
HOME=/root
```

2

Accediendo a la aplicación desplegada

José Manuel
Redondo López

- O entrar en ellos con un shell interactivo
  - `kubectl exec -ti $POD_NAME bash` **(1)**
- Ahora podemos
  - Consultar el contenido de sus ficheros y hacer los cambios que consideremos oportunos
  - En el ejemplo, instalamos la aplicación `curl` tras hacer `apt-get update` **(2)**
  - Dentro del contenedor `httpd:2.4` que estamos usando, la web servida se encuentra en el puerto 80: `curl http://localhost` **(3)**
- **Pero recuerda**: Nada de esto es realmente útil en producción, sólo es para pruebas
  - Si el pod al que pertenece el contenedor de la aplicación se destruye, **todo se perderá**
  - Los contenedores de un pod se regeneran a partir de su imagen, que es inmutable (recuerda Docker)

```
redondo@miw:~$ sudo kubectl exec -ti $POD_NAME -- bash
root@kubesample1-8649c7d75c-4qpjv:/usr/local/apache2# ls
bin  build  cgi-bin  conf  error  htdocs  icons  include  logs  modules
```

1

```
root@kubesample1-8649c7d75c-4qpjv:/usr/local/apache2# apt install curl
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  lsb-base
Use 'apt autoremove' to remove it.
The following NEW packages will be installed:
  curl
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
Need to get 264 kB of archives.
After this operation, 410 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian buster/main amd64 curl amd64 7.64.0-4+deb10u1
 [264 kB]
Fetched 264 kB in 0s (1832 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package curl.
(Reading database ... 9613 files and directories currently installed.)
Preparing to unpack .../curl_7.64.0-4+deb10u1_amd64.deb ...
Unpacking curl (7.64.0-4+deb10u1) ...
Setting up curl (7.64.0-4+deb10u1) ...
root@kubesample1-8649c7d75c-4qpjv:/usr/local/apache2#
```

2

```
root@kubesample1-8649c7d75c-4qpjv:/usr/local/apache2# curl http://localhost
<html><body><h1>It works!</h1></body></html>
```

3

Más sobre pods

José Manuel
Redondo López

- **Recapitulando**
  - Los pods son la unidad de despliegue atómica más pequeña de Kubernetes
  - Un pod está asociado a un nodo hasta que sea finalizado o borrado
  - Son una abstracción que **representa a un grupo de N contenedores que pueden compartir recursos**
    - **Almacenamiento** compartido (volúmenes)
    - **Red**, con una IP única de clúster
    - Información acerca de cómo ejecutar cada contenedor
      - Versión de la imagen del contenedor, puertos usados…
- **Si un pod tiene más de un contenedor, normalmente los contenedores extra hacen tareas de soporte de la aplicación principal**
  - Asignando distintos **roles**: data pullers, data pushers, proxies… (no los veremos en este curso)
- **En cualquier caso, los contenedores de un pod…**
  - Comparten una **misma dirección IP** y mapeo de puertos
  - Residen en el **mismo nodo**
  - Y **se planifican conjuntamente**

Más sobre nodos

José Manuel
Redondo López

- **Un nodo es como el "padre" de los pods:**
  - Los pods siempre se ejecutan dentro de uno
  - **Un mismo nodo puede tener varios pods**
- **Los nodos son "Worker" (MV o física) manejados por un Máster**
- **Cada nodo Worker ejecuta por lo menos**
  - **Kubelet**: Contiene el estado en ejecución de cada nodo y controla que todos estén en estado correcto
    - **Arranca, para y mantiene** contenedores organizándolos en los pods que le indique el control plane
    - Si un pod no está en el estado deseado, Kubelet lo vuelve a desplegar dentro del mismo nodo
    - Se usa un sistema de heartbeat para controlar que un nodo está operativo
    - En caso de que un nodo deje de estarlo, **se replican sus pods en otros nodos que si lo estén**
    - Dicho de otra forma, si el sistema falla, **se autocorrige** sin que tengamos que hacer nada

Nodos

José Manuel
Redondo López

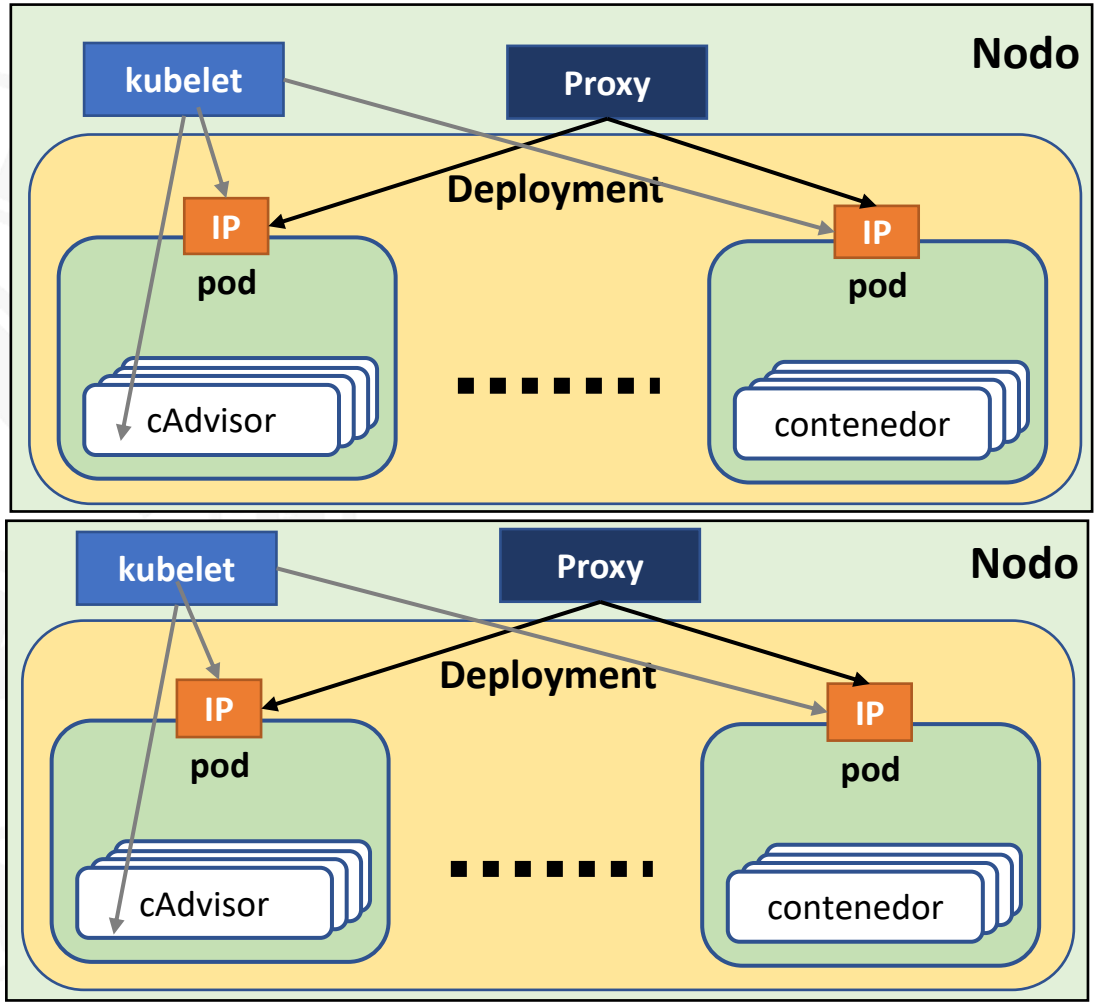
- **Un motor de contenedores** (ej.: Docker)
  - Capaz de sacar las imágenes de un registro, desempaquetarlas y **ejecutar las aplicaciones**
- **Kube-proxy**: Es un proxy de red y balanceador de carga, visto en un ejemplo de la sección anterior
  - **Reparte las peticiones** a los servicios entre los diversos contenedores que puedan atenderla
  - Abstrae este aspecto de los clientes
  - Es por tanto el **principal interfaz de comunicación entre nodos**
- **cAdvisor**: Agente que monitoriza el uso de recursos
  - Ejecuta **métricas de rendimiento** de CPU, memoria, disco y red
  - Toma decisiones en base a ellos

- En versiones anteriores, **Minikube** estaba limitado a un solo nodo
  - **Ya no es el caso**: https://minikube.sigs.k8s.io/docs/tutorials/multi_node/

MÁS SOBRE NODOS

José Manuel
Redondo López

- **Vemos una sección de la imagen de la primera parte que representa los elementos ya vistos**
  - Dos nodos Worker (maquinas)
  - Un despliegue (deployment) de una aplicación en cada nodo
  - Varios pods/nodo
  - Varios contenedores/pod
  - El kubelet de cada nodo
  - El cAdvisor de cada nodo



¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

- *¿Comprendes la importancia que tiene **Kubernetes** en escenarios de grandes despliegues de contenedores?*
- *¿Entiendes que nuestra instalación de **Kubernetes** (**minikube**) es idéntica a una que se haga en nube, pero permitiendo hacer pruebas en local y con pocos recursos?*
- *¿Has comprendido lo fácil que es desplegar una aplicación que tengamos en un contenedor sobre un clúster **Kubernetes** creando un **deployment** asociado?*
- *¿Has conseguido relacionar cada concepto teórico de **Kubernetes** con algo que has podido ver en tu primer despliegue?*
- *¿Recuerdas cómo activar un **proxy** para leer información de un **pod**?*
- *¿Entiendes que, aunque pueda entrarse físicamente a un contenedor de un **pod**, esto no es útil y por qué no lo es?*
- *¿Crees que ahora has entendido mejor lo que es un deployment y un nodo?*

Servicios y Etiquetas...en la práctica

Y más sobre pods y nodos

Servicios: Motivación

José Manuel
Redondo López

- Recuerda: los pods son “mortales”
  - **Tienen un ciclo de vida**
  - Si un nodo deja de estar disponible, los pods que ejecuta **se pierden**
- El objetivo principal de Kubernetes es que esto no suponga la pérdida de datos de las aplicaciones
- Para ello, sus usuarios deben definir el **desired state** del clúster
  - Es decir, cuántos contenedores de cada tipo queremos estar ejecutando en cada momento
  - Ej.: 2 contenedores MySQL, 3 contenedores Apache…
- Todo eso configura un concepto llamado **ReplicaSet**
  - Que modela **el estado en el que un clúster** (y sus pods) **deben estar**
  - De forma que si alguno de los pods que lo forma se pierde, Kubernetes crea nuevos pods para sustituir los que hemos perdido
  - Cómo y dónde lo hace depende de Kubernetes: ¡**nos ahorra todo el trabajo**!

Servicios: Motivación

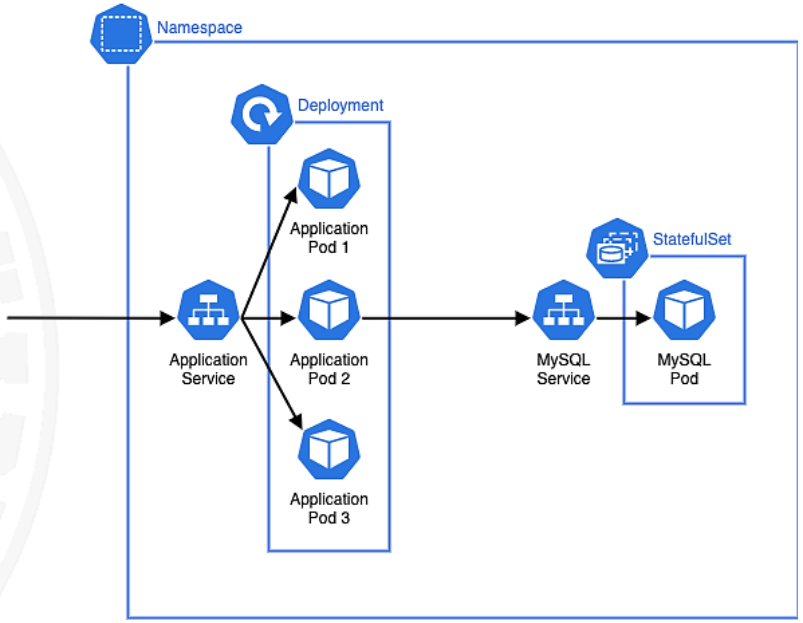
José Manuel
Redondo López

- **Por tanto, se le dice a Kubernetes el estado del clúster que deseamos tener**
  - Y él **se encarga automáticamente** de intentar que el clúster esté en el mismo todo el tiempo posible
- **Pero cada pod tiene su propia IP para comunicarse con él…**
  - ¿Qué pasa si ese pod se cae?
  - ¿Como puedo saber la IP del pod creado automáticamente para sustituirle?
- **Nunca debemos referirnos a un pod por su IP, sino por una forma de nombrarlos de más alto nivel que nos aísle de estos cambios**
  - ¡Ese es el motivo por el que existen los **servicios**!
  - Dicho de otra forma, un servicio es **la abstracción de alto nivel que usamos para referirnos a los pods** que implementan una determinada funcionalidad
  - **Nos aísla** de todo tipo de cosas que puedan ocurrir con los pods que implementan la funcionalidad
  - Nosotros **solo vemos el servicio**, no cómo está implementado en estos momentos

SERVICIOS

José Manuel
Redondo López

- Por tanto, un servicio de **Kubernetes** (también llamado microservicio) es una abstracción que define
  - Un **conjunto lógico de pods**
  - **Una política de acceso** a los mismos
- Permiten conseguir un **acoplamiento débil** entre pods que dependen entre ellos
  - Un servicio puede, por ejemplo, contener una capa completa de una aplicación
    - Ej.: Front-end, back-end...
  - Y varios servicios comunicándose entre ellos definirían la aplicación completa
    - Solo tendremos los servicios como punto de acceso a cada capa
    - Lo que pase dentro de cada una no le interesa al resto
  - Se pueden definir también con ficheros YAML y JSON, como el resto de los elementos de Kubernetes
    - Veremos ejemplos más adelante

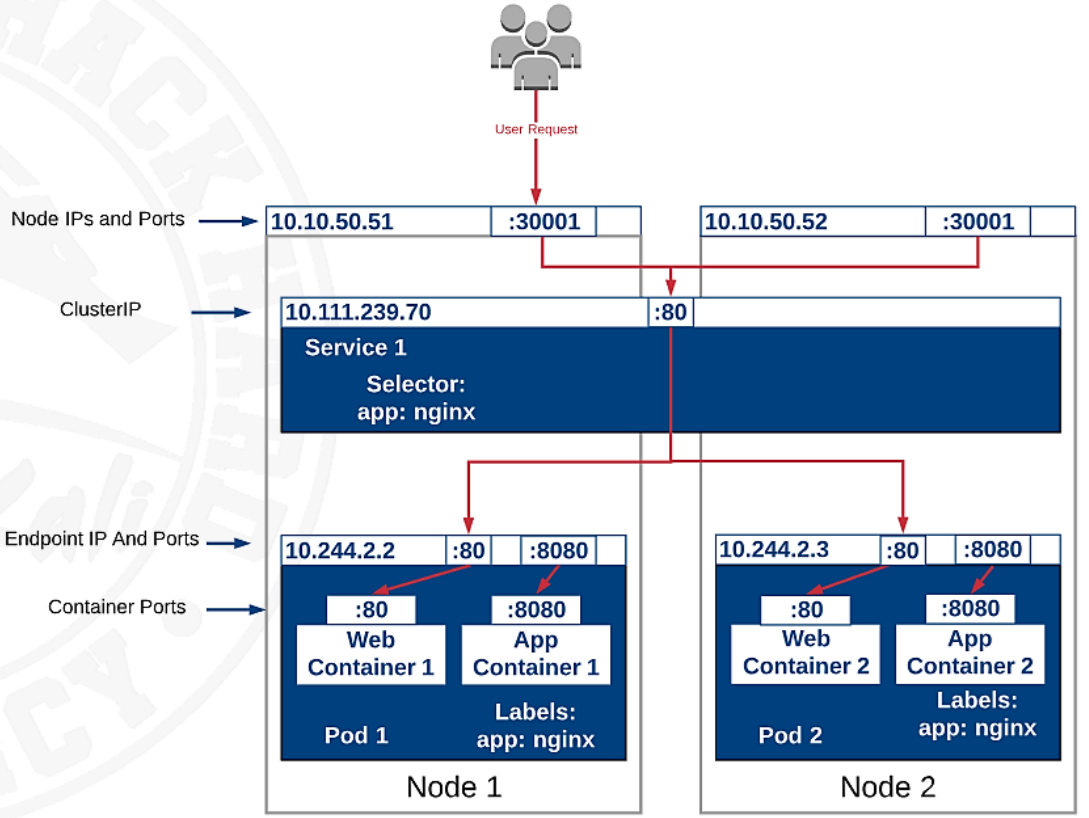


Aplicación con dos capas en forma de servicios. Fuente: www.eksworkshop.com/docs/introduction/getting-started/microservices

SERVICIOS Y ETIQUETAS

José Manuel
Redondo López

- **Para asociar Pods a Services y rutas de acceso a los mismos se usan etiquetas**
  - Aunque más que los PODs **se usan sus deployments asociados**
- **Kubernetes permite**
  - Asociarles IPs fijas
  - Asociarles nombres DNS
  - Hacerles balanceo de carga con política round-robin si hay varios pods asociados a una IP
  - También proporciona una forma de descubrir y exponer servicios



Servicios desplegados y las etiquetas de sus PODs. Fuentes:
https://theithollow.com/2019/02/05/kubernetes-service-publishing,
https://theithollow.com/2019/01/31/kubernetes-services-and-labels

SERVICIOS Y ETIQUETAS

José Manuel
Redondo López

- **Aunque cada pod tenga una dirección IP única, esta nunca se expone fuera del clúster sin un servicio**
  - Aunque podemos exponerla a través de un proxy, como ya vimos
- **Las etiquetas son la solución que permite a los servicios y a los pods comunicarse de forma débilmente acoplada**
  - **Debemos asociar a cada pod una etiqueta** (“frontend”, “backend”, “release”…)
  - Y será el **nombre lógico** que usemos para referirnos a él (¡**nunca su IP**!)
  - Y de esta forma **hacer consultas** que nos permitan buscar entre la infraestructura desplegada
  - Si tenemos muchos pods en muchos nodos (segunda imagen de la página anterior), contar con una herramienta que nos permite localizarlos es una ayuda muy importante
  - ¡Siempre que los hayamos etiquetado debidamente!
- **El conjunto de pods asociado con un servicio se determina con una LabelSelector (selector de etiquetas)**

SERVICIOS Y TRÁFICO DE RED

José Manuel
Redondo López

- **Los servicios permiten a las aplicaciones ejecutadas en sus pods recibir y transmitir información vía red exponiéndolas de distintas formas**
  - **ClusterIP** (por defecto): Expone el servicio en una IP interna del clúster Kubernetes
    - Permitiendo comunicarse con él servicio **solo dentro del clúster**
    - Ideal para capas de la aplicación **que no deban verse** desde el exterior
  - **NodePort**: Expone el servicio **en el mismo puerto** en cada nodo del clúster seleccionado con NAT
    - Haciendo que sea **accesible desde fuera** del mismo, es decir, usable por otros clústers / usuarios
  - **LoadBalancer**: Crea un balanceador de carga externo en la nube actual (si está soportado)
    - Y le asigna una **IP fija externa** al servicio que permite acceder a él con LB (tema 1)
  - **ExternalName**: Expone el servicio mediante un nombre DNS asociado al mismo, devolviendo un registro `CNAME` con el mismo sin falta de usar un proxy
    - Necesita la versión 1.7 o superior de `kube-dns`
    - No lo veremos en esta presentación, pero permite ofrecer servicios con nombre DNS, no sólo IP
- **Si usamos los servicios y no sus pods (lo recomendado), según como se exponga el servicio obtendremos un comportamiento u otro**

PROBANDO SERVICIOS Y ETIQUETAS

José Manuel
Redondo López

- Vamos a asociar un servicio con el pod que ejecuta la aplicación que creamos anteriormente
- Primero comprobamos que la aplicación está activa en su pod: `kubectl get pods`

```
redondo@miw:~$ sudo kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
kubesample1-8649c7d75c-4qpjv   1/1     Running   0          8h
```

- Luego comprobamos qué servicios tenemos: `kubectl get services`
  - Vemos que Kubernetes en sí se oferta como un servicio también, pero no hay más

```
redondo@miw:~$ sudo kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   9h
```

PROBANDO SERVICIOS Y ETIQUETAS

José Manuel
Redondo López

- **Creamos un servicio vinculado al pod anterior**
  - A través del deployment asociado al mismo que se le creó automáticamente
    - `kubectl expose deployment/kubesample1 --type="NodePort" --port 80` (1)
  - Lo asociamos refiriéndonos a él con el formato `[tipo]/[nombre]` (`deployment/kubesample1`)
  - Exponemos el servicio con tipo **NodePort** y el puerto 80,
    - Es donde está el servidor del contenedor `httpd:2.4`

```
redondo@miw:~$ sudo kubectl expose deployment/kubesample1 --type="NodePort" --po
rt 80
service/kubesample1 exposed
```

1

- **Si ejecutamos `kubectl get services` de nuevo, veremos lo siguiente (2)**
  - Ahora tenemos **2 servicios**: el básico de Kubernetes y la aplicación anterior expuesta como un servicio
    - Tipo **NodePort** (accesible desde el exterior)
  - El servicio nuevo usa un puerto aleatorio, pero mapeado contra el 80 de los pod asociados

```
redondo@miw:~$ sudo kubectl get services
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP        9h
kubesample1   NodePort    10.96.250.123   <none>        80:30636/TCP   28s
```

2

Probando servicios y etiquetas

José Manuel
Redondo López

- **Al usar NodePort, el servicio podrá ser usado desde fuera mediante NAT**
  - Ej.: Desde la máquina donde ejecutamos `minikube`
- **¡Pero la IP de NAT y el puerto son difíciles de recordar y pueden cambiar!**
  - Para facilitar el trabajo, podemos guardar el puerto del servicio en una variable de entorno **(1)**
    - `export NODE_PORT=$(sudo kubectl get services/kubesample1 -o go-template='{{(index .spec.ports 0).nodePort}}')`
    - `echo NODE_PORT=$NODE_PORT` **(1) (2)**
- **Ahora tenemos el puerto en `NODE_PORT`**
  - Y podemos comunicarnos con el servicio solamente usando variables
  - `curl $(sudo minikube ip):$NODE_PORT` **(3) (4)**

```
redondo@miw:~$ export NODE_PORT=$(sudo kubectl get services/kubesample1 -o go-te
mplate='{{(index .spec.ports 0).nodePort}}')
```

1

```
redondo@miw:~$ echo NODE_PORT=$NODE_PORT
NODE_PORT=30636
```

2

```
redondo@miw:~$ curl $(sudo minikube ip):$NODE_PORT
<html><body><h1>It works!</h1></body></html>
```

3

4

PROBANDO SERVICIOS Y ETIQUETAS

José Manuel
Redondo López

- Al crear el pod al desplegar la aplicación, el sistema le asignó una etiqueta automáticamente
- Podemos ver cuál fue mediante: `kubectl describe deployments`
- En este caso `run=kubesample1`
- Por tanto, no tenemos que hacer nada para crear una etiqueta asociada a un pod

```
redondo@miw:~$ sudo kubectl describe deployments
Name:                   kubesample1
Namespace:              default
CreationTimestamp:      Wed, 15 Jul 2020 05:37:42 +0000
Labels:                 app=kubesample1
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=kubesample1
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailab
le
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=kubesample1
  Containers:
   httpd:
    Image:        httpd:2.4
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   kubesample1-8649c7d75c (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  9h    deployment-controller  Scaled up replica set kub
esample1-8649c7d75c to 1
```

PROBANDO SERVICIOS Y ETIQUETAS

José Manuel
Redondo López

- **Podemos listar las etiquetas de servicios y pods con `--show-labels` (1)(2)**
- **Obtener pods que tienen la misma etiqueta:**

  `kubectl get pods -l run=kubesample1` **(3)**

  - `-l` hace una petición a una etiqueta concreta
  - También se puede aplicar a nivel de servicio además de pod individual: `kubectl get services -l run=kubesample1`
- **Añadir etiquetas a pods o servicios permite hacer consultas más complejas**
  - Conviene usarlas para dar un nombre descriptivo de la aplicación (lo que hace), su versión y su propósito
  - Así podemos buscar todos los pods pertenecientes a una aplicación (`app=mitienda`), versión (`app=v2`) o capa (`frontend`, `backend`)
  - Añadiéndoles más etiquetas a cada pod implicado

```
redondo@miw:~$ sudo kubectl get services --show-labels
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   LABELS
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        9h    compon
ent=apiserver,provider=kubernetes
kubesample1  NodePort    10.96.250.123   <none>        80:30636/TCP   18m   app=ku
besample1
```

1

```
redondo@miw:~$ sudo kubectl get pods --show-labels
NAME                          READY   STATUS    RESTARTS   AGE   LABELS
kubesample1-8649c7d75c-4qpjv  1/1     Running   0          9h    app=kubesample1,p
od-template-hash=8649c7d75c
```

2

```
redondo@miw:~$ sudo kubectl get pods -l app=kubesample1
NAME                          READY   STATUS    RESTARTS   AGE
kubesample1-8649c7d75c-4qpjv  1/1     Running   0          9h
redondo@miw:~$ sudo kubectl get services -l app=kubesample1
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubesample1  NodePort    10.96.250.123   <none>        80:30636/TCP   21m
```

3

PROBANDO SERVICIOS Y ETIQUETAS

José Manuel
Redondo López

- Para cambiar las etiquetas de un pod, 1º usamos una variable de entorno para guardar su nombre
  - Comprobándolo 1º con `kubectl get pods` **(1)**
- Guardamos el nombre
  - `POD_NAME=kubesample1-6f886b668-x8btw` **(2)**
- Y luego le añadimos una etiqueta nueva `app=version1`
  - `kubectl label pod $POD_NAME version=1.0` **(3)**
- Finalmente, comprobamos que se ha añadido a las etiquetas asociadas actualmente al pod
  - `kubectl describe pods $POD_NAME` **(4)**

```
redondo@miw:~$ sudo kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
kubesample1-8649c7d75c-4qpjv   1/1     Running   0          8h
```

1

```
redondo@miw:~$ POD_NAME=kubesample1-8649c7d75c-4qpjv
redondo@miw:~$ echo $POD_NAME
kubesample1-8649c7d75c-4qpjv
```

2

```
redondo@miw:~$ sudo kubectl label pod $POD_NAME version=1.0
pod/kubesample1-8649c7d75c-4qpjv labeled
```

3

```
redondo@miw:~$ sudo kubectl describe pods $POD_NAME
Name:         kubesample1-8649c7d75c-4qpjv
Namespace:    default
Priority:     0
Node:         miw/10.0.2.15
Start Time:   Wed, 15 Jul 2020 05:37:42 +0000
Labels:       app=kubesample1
              pod-template-hash=8649c7d75c
              version=1.0
Annotations:  <none>
Status:       Running
IP:           172.17.0.3
IPs:
  IP:           172.17.0.3
Controlled By:  ReplicaSet/kubesample1-8649c7d75c
Containers:
  httpd:
    Container ID:   docker://b1267e127c6a481bacd2faa15a139e1e86e8ffdb9b737e334f7c26
35bae26864
    Image:          httpd:2.4
    Image ID:       docker-pullable://httpd@sha256:387f896f9b6867c7fa543f7d1a686b0e
be777ed13f6f11efc8b94bec743a1e51
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Wed, 15 Jul 2020 05:37:43 +0000
    Ready:          True
```

4

Eliminando servicios

José Manuel
Redondo López

- **Si estamos haciendo pruebas, puede convenirnos eliminar servicios a los que ya no le demos uso**
  - `sudo kubectl delete service -l run=kubesample1`
  - Y **verificar** que ya no esté disponible con: `kubectl get services`
  - También conviene comprobar que la IP del servicio ya no responde
    - `curl $(sudo minikube ip):$NODE_PORT`
- **Esto nos permite deshacer el trabajo de esta segunda parte**
- **Pero eliminar un servicio no elimina la aplicación**
  - Que debería seguir disponible si accedemos directamente a su pod
  - `sudo kubectl exec -ti $POD_NAME curl localhost`
  - Solo se elimina esa vía de acceso a la misma

EVENTOS

José Manuel
Redondo López

- **Podemos ver todo lo que ha pasado en nuestra sesión de trabajo con `kubectl get events`**
  - Admite ordenación y búsqueda por distintos criterios
  - **Ejemplos**: https://www.learnitguide.net/2023/05/kubectl-get-events-sort-by-time.html

```
kalsoom@kalsoom-VirtualBox:~$ kubectl get events --sort-by=.metadata.creationTi
mestamp -A
NAMESPACE     LAST SEEN   TYPE      REASON           OBJECT
            MESSAGE
kube-system   22m         Normal    Pulled           pod/kube-apiserver-minikub
e             Container image "k8s.gcr.io/kube-apiserver:v1.20.7" already presen
t on machine
kube-system   22m         Normal    Pulled           pod/kube-scheduler-minikub
e             Container image "k8s.gcr.io/kube-scheduler:v1.20.7" already presen
t on machine
kube-system   18m         Normal    Started          pod/kube-controller-manage
r-minikube    Started container kube-controller-manager
kube-system   18m         Normal    Created          pod/kube-controller-manage
r-minikube    Created container kube-controller-manager
kube-system   18m         Normal    Pulled           pod/kube-controller-manage
r-minikube    Container image "k8s.gcr.io/kube-controller-manager:v1.20.7" alrea
dy present on machine
kube-system   22m         Normal    Created          pod/kube-apiserver-minikub
e             Created container kube-apiserver
kube-system   22m         Normal    Started          pod/kube-apiserver-minikub
e             Started container kube-apiserver
kube-system   22m         Normal    Created          pod/kube-scheduler-minikub
e             Created container kube-scheduler
kube-system   21m         Warning   Unhealthy        pod/kube-scheduler-minikub
e             Startup probe failed: Get "https://127.0.0.1:10259/healthz": dial
tcp 127.0.0.1:10259: connect: connection refused
```

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

- *¿Has entendido el propósito de los servicios y el problema que resuelven?*
- *¿Comprendes cómo se divide una aplicación basada en microservicios?*
- *¿Has entendido la importancia de las etiquetas a la hora de asociar servicios y los **pods** que implementan sus funciones, y poder así localizarlos?*
- *¿Has entendido las 4 formas que hay de exponer un servicio en K8s, y los casos de uso que modelan para adaptarse a distintas necesidades?*
- *¿Has entendido el proceso descrito para exponer un servicio con **NodePort**, cómo se crean etiquetas asociadas al mismo y cómo se puede acceder a él y a su información?*
- *¿Entiendes que eliminar servicios no elimina los **pods** asociados al mismo, sino una forma de acceder a ellos?*
- *¿Entiendes que los eventos son como un log de todo lo hecho con **K8s**?*

< Ir al Índice

Fuente: Bing Chat AI

Deployments, etc. >

Volúmenes >

ConfigMaps, Secrets >

Network Policies >

Desplegando en Kubernetes con código

IaC en Kubernetes

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿No dijiste que esto era IaC? ¿Dónde está el código?

José Manuel
Redondo López

- **Durante toda la asignatura estamos hablando de las ventajas del IaC y lo cierto es que en Kubernetes aún no lo hemos usado**
  - Todos los despliegues han sido interactuando manualmente con el API del clúster usando `kubectl`
- **Vamos a ver cómo usar ficheros YAML para arrancar los elementos que hemos visto**
  - ¡Era necesario entender los conceptos antes de poder hacerlo! ☺
  - Esto nos permitirá arrancar aplicaciones en un clúster Kubernetes de forma más sencilla
- **Existen diversas formas de crear clústeres con `kubectl`**
  - https://kubernetes.io/docs/concepts/overview/working-with-objects/object-management/

Deployments y Services

Ficheros para desplegar Deployments y servicios Kubernetes

Creando un deployment por código

José Manuel
Redondo López

- **En este fichero**
  - Creamos un deployment llamado `web-deployment`
  - Le asignamos una **etiqueta** llamada `app:apache2` que lo identifique
  - Estará compuesto por 3 pods (`replicas`)
  - Dentro de cada pod habrá un contenedor de la imagen `httpd:latest`
  - Estos contenedores dejan accesible su puerto 80
- **¡Todos los conceptos vistos, resumido y automatizados en un fichero!**
- **Todos los ficheros de configuración de Kubernetes tienen campos obligatorios**
  - `apiVersion`
  - `kind`
  - `metadata`

SampleDeployment.yml

```yaml
apiVersion: apps/v1
# Tipo de objeto Kubernetes: Deployment
kind: Deployment
metadata:
  # Nombre del deployment
  name: web-deployment
spec:
  selector:
    matchLabels:
      app: apache2
  # Indica al controlador que ejecute 3 pods
  replicas: 3
  template:
    metadata:
      labels:
        # Etiqueta con el nombre de la aplicación
        # desplegada
        app: apache2
    spec:
      containers:
      - name: httpd
        # Nombre de la imagen a partir de la que se crean
        # los contenedores
        image: httpd:latest
        ports:
        # Puerto expuesto de los contenedores
        - containerPort: 80
```

CREANDO UN DEPLOYMENT POR CÓDIGO

José Manuel
Redondo López

- Para crear el **deployment** basta con hacer
  - `kubectl apply -f <fichero yml>`
- Y para comprobar que se ha creado usamos la **matchlabel** del fichero
  - `kubectl get pods -l app=apache2`
- Con esa información podremos preguntar detalles de cada **pod** creado (3 en total)
  - `kubectl describe pod <nombre de pod>`
- Si cambiamos el fichero por algún motivo basta con repetir el mismo proceso para que el **deployment** **se actualice**
  - Más **pods**, distinta versión de **Apache**…
  - Se crearán **pods** nuevos con la nueva configuración y se borrarán los antiguos
- Y no olvides que podemos borrarlo
  - `kubectl delete deployment web-deployment`

Asociando Services a Deployments

José Manuel
Redondo López

- Hemos visto que la forma de exponer deployments “hacia afuera” es con servicios
- Se puede crear un servicio y vincularlo a un deployment por código
- Se usa como selector la label del deployment incluida en la sección `metadata` del deployment
- Si volvemos a aplicar el fichero, tendremos un nuevo servicio

SampleService.yml

```yaml
apiVersion: apps/v1
# Tipo de objeto Kubernetes: Deployment
kind: Deployment
metadata:
  # Nombre del deployment
  name: web-deployment
spec:
  selector:
    matchLabels:
      app: apache2
  # Indica al controlador que ejecute 3 pods
  replicas: 3
  template:
    metadata:
      labels:
        # Etiqueta con el nombre de la aplicación
        # desplegada
        app: apache2
    spec:
      containers:
      - name: httpd
        # Nombre de la imagen a partir de la que se crean
        # los contenedores
        image: httpd:latest
        ports:
        # Puerto expuesto de los contenedores
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
    name: frontend
spec:
    type: NodePort
    selector:
        app: apache2
    ports:
        - port: 80
          targetPort: 80
          nodePort: 30002
```

Volúmenes persistentes

Salvaguardando nuestros datos

VOLÚMENES PERSISTENTES

José Manuel
Redondo López

- Uno de los principales problemas de trabajar con aplicaciones en Kubernetes es que los pods sean efímeros
  - Kubernetes puede regenerar los pods en caso de pérdida o reinicio
  - Pero toda la información que hayamos guardado en sus contenedores **se perderá**
  - Son contenedores al fin y al cabo...
- Para evitarlo es necesario usar un **volumen persistente**
  - https://kubernetes.io/docs/concepts/storage/persistent-volumes/
- Todo lo que guardemos en él se mantendrá a salvo de reinicios y caídas accidentales de pods
- El API de Kubernetes separa el almacenamiento de la computación: se mantienen en entidades separadas

Volúmenes persistentes

José Manuel
Redondo López

- **Existen dos tipos de recursos asociados con volúmenes**
  - **Los PersistentVolumes**: se usan para **definir** un volumen de almacenamiento en el sistema
    - Su ciclo de vida es independiente de los **pods** que lo usan
    - **En otras palabras**: existen sin importar que sus usuarios se creen, mueran o se reinicien
    - Hay muchas implementaciones gracias a **plugins**
      - NFS, GlusterFS, CephFS...
    - También con proveedores de nube
      - GCEPersistentDisk (Google Compute Engine), AWSElasticBlockStore (Amazon Web Services), AzureFile, AzureDisk...
  - **Los PersistentVolumeClaims**: **Peticiones** que los **pods** hacen para **obtener el acceso** a un volumen
    - Una vez hecha la petición, esos volúmenes **se montarán en un path concreto** del **pod**
    - Y se proporcionará una abstracción del mismo al sistema de almacenamiento del **pod**
    - Cada uno de estos **claims** debe especificar un atributo `storageClassName` para obtener un **PersistentVolume** que sea adecuado para las necesidades del **pod**
    - Si el **claim** emite un `storageClassName` que no coincide con la de ningún **PersistentVolume** existente, el clúster **intentará crear uno nuevo** dinámicamente (¡**cuidado**!)
    - Si no hay volúmenes que satisfagan las características del **claim**, este quedará en estado **pending**

Definición de un volumen y su claim

José Manuel
Redondo López

- En este fichero `storage.yml` definimos un `PersistentVolume` y su `PersistentVolumeClaim` asociado
- Tanto el `PersistentVolume` como su `PersistentVolumeClaim` se definen con el mismo `storageClassName`
  - https://kubernetes.io/docs/concepts/storage/storage-classes/
  - Ambos **pertenecen a la misma clase de almacenamiento** y por tanto se pueden vincular
    - Kubernetes busca un volumen que cumpla los parámetros del claim entre los que pertenecen a la misma clase de almacenamiento
  - Se pueden crear clases de almacenamiento con diferentes características (tema más avanzado)
- Más información
  - https://kubernetes.io/docs/concepts/storage/volumes/
  - https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/
  - https://kubernetes.io/docs/concepts/storage/persistent-volumes/
  - https://www.paradigmadigital.com/dev/kubernetes-almacenamiento-persistencia-datos/

```yaml
---
kind: PersistentVolume
apiVersion: v1
metadata:
  name: datasafe
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 50M
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  labels:
    app: apache2
  name: datasafe-claim
spec:
  storageClassName: manual
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 50M
```

Aplicación de un volumen y su claim

José Manuel
Redondo López

- Una vez definido el volumen, debemos incorporarlo a nuestro sistema **Kubernetes** con `kubectl apply -f storage.yml`
- Tras aplicar este fichero se creará un nuevo `PersistentVolume` con una capacidad de 50Mb
- Usar el modo de acceso (`accessMode`) `ReadWriteOnce` permite a un único nodo a la vez escribir y leer en el mismo
  - Esto **evita problemas de corrupción de datos** si lo que vamos a leer no está preparado para lecturas / escrituras concurrentes (caso más probable)
- También se define con la política `Retain`
  - Permite al `PersistentVolume` mantenerse tal y como está
  - Incluso aunque su `PersistentVolumeClaim` se libere
  - Es el comportamiento por defecto

INCORPORACIÓN DE VOLÚMENES A DEPLOYMENTS

José Manuel
Redondo López

- **Ahora podemos modificar la especificación de nuestro Deployment para añadirle un volumen**
  - Añadimos al fichero el código para obtener el volumen gracias a su `persistentVolumeClaim`
- **Lo montamos en una ruta (`volumeMounts`)**
  - Si aplicamos de nuevo el fichero del deployment, este se actualizará con la nueva configuración
- **Aunque borremos pods o se caigan, la información del volumen seguirá ahí**
  - Siempre que se use el mismo `persistentVolumeClaim` para obtener el `PersistentVolume` donde se guardaron los datos
  - **No se pierden**, aunque ahora se ejecute otro contenedor
- **Otro ejemplo**
  - https://www.adictosaltrabajo.com/2019/01/03/desplegar-una-aplicacion-web-con-minikube/

```
… (resto del fichero SampleService.yml anterior)
  spec:
   containers:
    - name: httpd
      image: httpd:latest
      ports:
      - containerPort: 80
      volumeMounts:
      - mountPath: /mnt/data
        name: datasafe
  volumes:
  - name: datasafe
    persistentVolumeClaim:
      claimName: datasafe-claim
```

ConfigMaps y Secrets

Pasando configuración y secretos a nuestros deployments

ConfigMaps y Secrets

José Manuel
Redondo López

- Una de las formas que hay de suministrar datos a una aplicación es con variables de entorno (`env`)
- Se pueden especificar en el fichero que configura su **deployment**
  - Pero **puede exponer datos secretos en texto plano**, lo que es una falta de seguridad enorme
    - ¡En el ejemplo exponemos una clave!
  - También facilita la **duplicación de variables de entorno** en distintos ficheros `.yml`
  - Esto hace el mantenimiento más complejo
- Los **ConfigMaps** y los **Secrets** son la forma adecuada de gestionar estos casos

```yaml
kind: Deployment
metadata:
  name: nginx
spec:
  ... (definición de contenedor, etc.)
      env: # Variables de entorno
      - name: DB
        value: my_web_db
      - name: DB_USER
        value: admin
      - name: DB_PASSWORD
        value: willyrex
```

¿Hardcodeando claves? NOOOO
¿Qué pasa si subes este fichero a GitHub ahora?

ConfigMaps y Secrets

José Manuel
Redondo López

- **Son dos formas similares de sacar parámetros de configuración de los ficheros que definen los Deployments**
  - Esto los hace más portables, mantenibles y seguros
  - Pero tienen dos diferencias principales
- **Diferente propósito**
  - Los Secrets son la forma recomendada de **guardar información privada**
    - Como tokens, API keys, claves...
  - Los ConfigMaps son para valores de configuración en general
- **Formato de los datos que guardan**
  - Los datos dentro de los secrets se deben guardar codificados en base64
    - Esto **permite guardar datos binarios**, al poder codificarse en ese formato
  - Los ConfigMaps solo permiten guardar texto plano

Definiendo un secreto

José Manuel
Redondo López

- Como decíamos, lo primero es codificar el dato en **Base64**
  - Codifiquemos la contraseña `"willyrex"` por ejemplo: `echo -n "willyrex" | base64 -`
  - Devuelve: `dXJsX3Nob3J0ZW5lcl9kYg==`
- Ahora ya podemos definir el secreto en un fichero como el adjunto (`secrets.yml`)
  - Le damos el nombre `secret-config`
- Y luego incorporar su contenido
  - `kubectl create -f secrets.yml`

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secret-config
type: Opaque
data:
  password: dXJsX3Nob3J0ZW5lcl9kYg==
```

Base64 no es un método de cifrado, pero una vez incorporado el clúster con `create` podemos borrar su contenido, y este quedará cifrado dentro del clúster

USANDO UN SECRETO DESDE UN DEPLOYMENT

José Manuel
Redondo López

- Con el secreto definido y creado, ya podemos adaptar un **deployment** para usarlo
- Obteniendo el valor de la entrada `password` asociada al nombre `secret-config` anterior
- Usamos para ello el par de directivas `valueFrom` / `secretKeyRef`
- Los valores de los secretos estarán disponibles para la aplicación en forma de variables de entorno
  - En este caso debemos buscar la variable de entorno `password`

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: webserver
spec:
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: httpd
        image: httpd:latest
        ports:
        - containerPort: 80
        env:
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: secret-config
              key: password
```

Léeme el valor password del archivo de secretos `secret-config`
¿Cuál es el valor de `password`? No lo sabes, y en el fichero ya no está ☺
(¡Esto ahora si se puede subir a GitHub!)

DEFINIENDO CONFIGMAPS

José Manuel
Redondo López

- **Se pueden definir en forma clave/valor dentro del fichero del deployment**
- **Pero es mejor tratar de desacoplarlo del mismo cargándolo desde un fichero externo** (`config.yaml`)
  - Creamos el fichero con este contenido

    ```
    db: my_web_db
    db_user: admin
    ```
  - Incorporamos el ConfigMap a partir de él
    - `kubectl create configmap app-config --from-file=./config.yaml`
- **En el ejemplo, cargamos el fichero en un nuevo volumen**
  - Ahora podemos acceder a toda la configuración desde la aplicación desplegada en el pod
  - Habrá un `config.yaml` en `/etc/config`
- **Más opciones para definir ConfigMaps**
  - https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/

```
… (resto del fichero anterior)
  volumeMounts:
  - mountPath: /mnt/data
    name: datasafe
  - name: config-volume
    mountPath: /etc/config
  env:
  - name: DB_PASSWORD
    valueFrom:
      secretKeyRef:
        name: database-secret-config
        key: password
volumes:
- name: datasafe
  persistentVolumeClaim:
    claimName: datasafe-claim
- name: config-volume
  configMap:
    name: app-config
```

Cargamos las variables `db` y `db_user` así

Network Policies

Controlando quién accede a qué en nuestro clúster Kubernetes

Network Policies

José Manuel
Redondo López

- Permiten especificar con qué **endpoints** se puede comunicar un **pod** vía red
- Se aplican sobre los **pods** tanto para el tráfico de salida (**egress**) como de entrada (**ingress**) y no tienen efecto en otras conexiones de red
- Las entidades con las que un **pod** se puede comunicar (o establecer restricciones) son una combinación de
  - **Otros pods** permitidos (un **pod** no puede bloquear el acceso a si mismo)
  - **Espacios de nombres** (**namespaces** de K8s) permitidos
  - **Bloques de IPs** (aunque el tráfico de un **pod** a su nodo siempre se permite, sino no se podría manejar)
- Debe estar configurado un proveedor de red que las soporte
  - El que viene por defecto en **minikube** **no lo hace** y aunque pueden aplicarse sin error, no funcionan
  - Hay una serie de proveedores disponibles que lo permiten: https://kubernetes.io/docs/tasks/administer-cluster/declare-network-policy/
  - Uno muy sencillo de usar es **Cilium**: https://kubernetes.io/docs/tasks/administer-cluster/network-policy-provider/cilium-network-policy/

Network Policies

José Manuel
Redondo López

- **Por defecto un pod no tiene restricciones al tráfico saliente ni entrante**
- **Para establecerlas, es necesario que exista un fichero con una Network Policy**
  - Que seleccione el pod como target y tenga la palabra Egress o Ingress en su campo `policyType`
  - Veremos un ejemplo en la siguiente transparencia
- **Cuando hay una entrada así, las únicas conexiones permitidas desde / hacia el pod son a la lista de endpoints que aparecen bajo la lista `egress` / `ingress`**
- **Puede haber varias Network Policies definidas para un pod**
  - Son aditivas (las restricciones finales son **la unión** de todas las aplicables)

Creando una NetworkPolicy

José Manuel
Redondo López

- `spec`: **La especificación del estado deseado de la NetworkPolicy**
  - Define toda la información para crear un NetworkPolicy concreta en un namespace dado
  - https://github.com/kubernetes/community/blob/master/contributors/devel/sig-architecture/api-conventions.md#spec-and-status
- `podSelector`: **Selecciona el conjunto de pods al que se aplica la política**
  - Es obligatorio para una NetworkPolicy
  - En el ejemplo serían todos los que tengan la **etiqueta** "`role=webserver`"
  - Si se deja vacío, **se aplica a todos los pods** del namespace
- `policyTypes`: `Ingress`, `Egress` **o ambos**
  - Si no se especifica, entonces las reglas especificadas se aplican a `Ingress`
  - Solo se aplican a `Egress` si alguna lleva esa etiqueta

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: servidor_pruebas
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: webserver
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - ipBlock:
            cidr: 172.17.0.0/16
            except:
              - 172.17.1.0/24
        - namespaceSelector:
            matchLabels:
              project: miproyecto
        - podSelector:
            matchLabels:
              role: frontend
      ports:
        - protocol: TCP
          port: 80
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
      ports:
        - protocol: TCP
```

Creando una network policy: policy types

José Manuel
Redondo López

- `ingress`: **Reglas de tráfico entrante**
  - Cada regla permite el tráfico desde los **endpoints** cubiertos por las secciones `from` y `ports`
  - En el ejemplo se permite el tráfico al puerto 80 desde cualquiera de los tres orígenes vistos
    - Una red de contenedores (`ipBlock`)
    - Los **endpoints** de un espacio de nombres (**miproyecto**, `namespaceSelector`)
    - Desde los contenedores de un **pod** llamado **frontend** (`podSelector`)
    - Otras opciones podrían ser aquellos **pods** que tengan una determinada etiqueta (`matchlabels`)
      - Ej.: `app: apache2`
- `egress`: **Reglas de tráfico saliente**
  - Admite los mismos elementos que el caso anterior
  - En el ejemplo se permite la conexión hacia el puerto 8080 de cualquier **endpoint** en la red (`ipBlock`) `10.0.0.0/24`

EJEMPLO

José Manuel
Redondo López

- **En resumen, esta política restringe los pods con la etiqueta "`role=webserver`" en el espacio de nombres "`default`"**
  - Tanto para tráfico saliente como entrante
  - **Reglas de entrada**: Permite conexiones al puerto 80 de todos los pods en el espacio de nombres "`default`" con la etiqueta "`role=webserver`" desde:
    - Cualquier pod en el espacio de nombres "`default`" con la etiqueta "`role=frontend`"
    - Cualquier pod en un namespace con la etiqueta "`project=miproyecto`"
    - Direcciones IP en el rango `172.17.0.0–172.17.0.255` y `172.17.2.0–172.17.255.255`
      - Es decir, todo `172.17.0.0/16` menos `172.17.1.0/24`
  - **Reglas de salida**: Permite las conexiones desde cualquier pod en el namespace "`default`" con la etiqueta "`role=webserver`" a la red `10.0.0.0/24` en el puerto TCP `8080`

EJEMPLOS MÁS SENCILLOS

José Manuel
Redondo López

Denegar por defecto todo el tráfico entrante en un namespace (selecciona todos los pods, pero no establece ninguna regla de Ingress)

```yaml
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

Permitir todo el tráfico entrante (cuidado: ninguna política podrá bloquear conexiones a estos pods con esta aplicada)

```yaml
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress
spec:
  podSelector: {}
  ingress:
  - {}
  policyTypes:
  - Ingress
```

Denegar por defecto todo el tráfico saliente (selecciona todos los pods, pero no especifica ninguna política de Egress)

```yaml
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-egress
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

Bloquear todo el tráfico saliente y entrante

```yaml
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Permitir todo el tráfico saliente (de nuevo, ninguna otra política podrá bloquear tráfico saliente)

```yaml
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-egress
spec:
  podSelector: {}
  egress:
  - {}
  policyTypes:
  - Egress
```

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

- **Deployments y servicios**
  - *¿Entiendes cómo crear un deployment vía código en un fichero YAML?*
  - *¿Entiendes cómo se combina esto con la opción apply de kubectl y cómo podemos actualizar el deployment si cambiamos su fichero?*
  - *¿Has entendido como asociar un servicio a un deployment por código y la importancia de nuevo de las etiquetas a la hora de seleccionar que deployment corresponda a cada servicio?*
- **Volúmenes**
  - *¿Entiendes qué nos proporcionan los volúmenes a la hora de manejar un clúster Kubernetes?*
  - *¿Entiendes la diferencia entre un volumen y el claim a un volumen, y para que se usan ambos conceptos?*
  - *¿Crees que puedes reproducir el proceso de asociar un volumen y su claim a la hora de incorporarlo a un deployment?*

¿Crees que lo has entendido todo? ¡Autoevalúate!

José Manuel
Redondo López

?

ConfigMaps y Secrets

- *¿Entiendes por qué usar ConfigMaps y Secrets es mejor que usar variables de entorno?*
- *¿Has entendido las principales diferencias entre ConfigMaps y Secrets?*
- *¿Recuerdas como definir un secret y como acceder a su valor dentro de tu aplicación K8s?*
- *¿Recuerdas como definir un ConfigMap en un fichero externo e incorporarlo a un deployment?*
- *¿Entiendes como vincular ConfigMaps a volúmenes?*

Network Policies

- *¿Entiendes que son entidades de K8s que te permiten hacer las veces de firewall?*
- *¿Comprendes cómo eso suple la imposibilidad de hacer estas operaciones dentro de los propios contenedores?*
- *¿Recuerdas que necesitas instalar un proveedor de red de 3ºs para soportarlos en minikube?*
- *¿Distingues correctamente entre tráfico ingress y egress?*
- *¿Has comprendido como definir restricciones de tráfico egress e ingress y asociarlos con un pod usando etiquetas?*
- *¿Controlas los valores por defecto si no especificas parámetros en las Network Policies?*
- *¿Entiendes las entidades que puedes restringir?*

< Ir al Índice

Fuente: Bing Chat AI

Manual >

Auto-escalado >

Escalando Aplicaciones

¡Hasta el infinito, y más allá!

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **Este apartado va a lidiar con la forma de hacer que los clústeres de K8s puedan con más carga de trabajo, escalando cuando sea necesario**
  - **De forma manual**, especificando nosotros el nº de pods que queremos
  - **De forma automática**, respondiendo dinámicamente a la demanda de peticiones que le llegue

Escalado manual de aplicaciones

Cómo escalar una aplicación a su desired state

Escalado manual de aplicaciones

José Manuel
Redondo López

- **Se trata de que el sistema cree más pods que contienen nuestra aplicación**
  - Y los **planifique automáticamente** en los nodos disponibles
- **Para ello, le indicamos a Kubernetes cuál es el desired state del que hablábamos**
  - Cuántos pods queremos
- **Kubernetes hace el resto**
  - **Crea** automáticamente múltiples instancias de la aplicación (**contenedores de sus imágenes**)
  - **Distribuye** el tráfico a todos los pods (**balanceo de carga**)
  - **Monitoriza** cada pod para comprobar si está disponible y envía tráfico a los que respondan
- **Es decir, nos implementa toda la gestión de un clúster**
  - Controlando cuántos nodos tiene y distribuyendo automática y óptimamente el tráfico
  - ¡Sin que tengamos que hacer prácticamente casi nada!

ESCALANDO MANUALMENTE UNA APLICACIÓN

José Manuel
Redondo López

- **Para escalar una aplicación primero hay que asegurarse de**
  - Arrancar/elegir la aplicación que queremos escalar
  - Comprobar que está desplegada correctamente: `kubectl get deployments`
  - En dicha comprobación vemos los siguientes valores:
    - `READY`: El valor que aparece ahí se llama `CURRENT STATE/DESIRED STATE`
    - `UP_TO_DATE`: N.º de replicas actualizadas para alcanzar el **desired state**
    - `AVAILABLE`: Cuántas replicas están disponibles para trabajar
- **¿Y luego?**
  - Decirle al sistema que quieres N réplicas de un **deployment** (**desired state**) y que éste se encargue de llevar el nº de réplicas actual (**current state**) al indicado
    - `kubectl scale deployments/kubesample1 --replicas= <nº de pods que quieres>`
  - Es decir, le decimos manualmente cuántas réplicas de un servicio queremos
  - Inicialmente, solo tenemos una réplica activa y como **desired state**. ¡vamos a por más!

```
redondo@miw:~$ sudo kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
kubesample1   1/1     1            1           9h
```

ESCALANDO MANUALMENTE UNA APLICACIÓN

José Manuel
Redondo López

- **Con las réplicas en marcha, el sistema hará balanceo de carga de peticiones**
  - Para comprobarlo, haremos un “truco”
  - Nos conectamos al contenedor en marcha en estos momentos y cambios la web que sirve **antes de hacer las réplicas (1) (2)**
  - Las réplicas **se añaden a partir de imágenes de contenedores**
    - El contenedor en ejecución se mantiene
  - En consecuencia, cuando hagamos el escalado...
    - Algunas peticiones mostrarán la web modificada (si la sirve el contenedor que estaba en marcha)
    - Otras la web original (la que tiene la imagen desde la que se crearon nuevos contenedores)
    - Eso quiere decir que nos ha contestado una réplica creada posteriormente
    - Y es **señal de que las peticiones se distribuyen**

```
redondo@miw:~$ sudo kubectl exec -ti $POD_NAME -- bash
root@kubesample1-8649c7d75c-4qpjv:/usr/local/apache2# pwd
/usr/local/apache2
root@kubesample1-8649c7d75c-4qpjv:/usr/local/apache2# ls
bin  build  cgi-bin  conf  error  htdocs  icons  include  logs  modules
root@kubesample1-8649c7d75c-4qpjv:/usr/local/apache2# cd htdocs
root@kubesample1-8649c7d75c-4qpjv:/usr/local/apache2/htdocs#
```

1

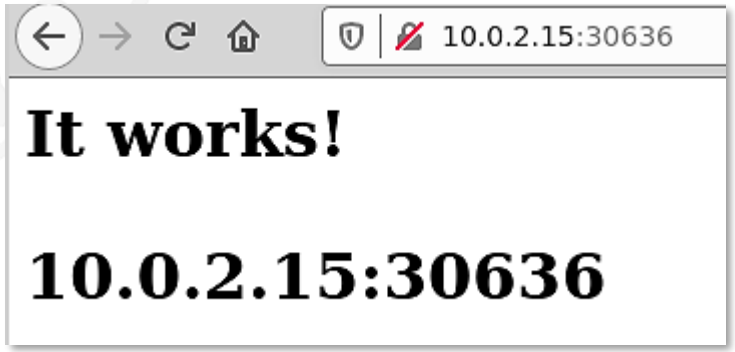
```
 GNU nano 3.2                    index.html                    Modified

<html>
<body>
<h1>
It works!<p id="ip"></p>

<script>document.getElementById("ip").innerHTML = location.host</script>

</h1>
</body>
</html>
```

2



ESCALANDO MANUALMENTE UNA APLICACIÓN

José Manuel
Redondo López

- Podemos escalar una aplicación en marcha de nuevo con `scale`
  - `kubectl scale deployments/kubesample1 --replicas=4` **(1)**
  - Con él incrementamos el nº de réplicas a 4
  - Kubernetes ahora hará **balanceo de carga** entre ellas de cualquier petición a la aplicación
  - ¡No tienes que preocuparte de nada!
- Si ahora vemos el despliegue de la aplicación vemos que hemos generado 4 pods (2) (3)
- El siguiente comando se usa para obtener más información de las réplicas generadas
  - `kubectl describe deployments/kubesample1` **(4)**

```
redondo@miw:~$ sudo kubectl scale deployments/kubesample1 --replicas=4
deployment.apps/kubesample1 scaled
```

1

```
redondo@miw:~$ sudo kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
kubesample1   4/4     4            4           9h
```

2

```
redondo@miw:~$ sudo kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
kubesample1-8649c7d75c-4qpjv   1/1     Running   0          9h
kubesample1-8649c7d75c-95qvs   1/1     Running   0          49s
kubesample1-8649c7d75c-qxdk9   1/1     Running   0          49s
kubesample1-8649c7d75c-zn75l   1/1     Running   0          49s
```

3

```
redondo@miw:~$ sudo kubectl describe deployments/kubesample1
Name:                   kubesample1
Namespace:              default
CreationTimestamp:      Wed, 15 Jul 2020 05:37:42 +0000
Labels:                 app=kubesample1
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=kubesample1
Replicas:               4 desired | 4 updated | 4 total | 4 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=kubesample1
  Containers:
   httpd:
    Image:        httpd:2.4
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
```

4

Escalando manualmente una aplicación: Balanceo de carga

José Manuel
Redondo López

- La gran ventaja del escalado es el balanceo de carga automático de las peticiones entrantes
  - Kubernetes se asegura de que las peticiones no sean atendidas siempre por el mismo pod
    - Sino **que se vayan distribuyendo** entre ellos
  - Para probar esto podemos intentar varios métodos
  - Uno de ellos es averiguar las IPs internas de los pods escalados **(1)**
  - Y mandar peticiones manuales a cada una de ellas **(2)**, **pero no se recomienda**

```
redondo@miw:~$ sudo kubectl describe services
Name:              kubernetes
Namespace:         default
Labels:            component=apiserver
                   provider=kubernetes
Annotations:       <none>
Selector:          <none>
Type:              ClusterIP
IP:                10.96.0.1
Port:              https  443/TCP
TargetPort:        8443/TCP
Endpoints:         10.0.2.15:8443
Session Affinity:  None
Events:            <none>


Name:                     kubesample1
Namespace:                default
Labels:                   app=kubesample1
Annotations:              <none>
Selector:                 app=kubesample1
Type:                     NodePort
IP:                       10.96.250.123
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30636/TCP
Endpoints:                172.17.0.3:80,172.17.0.4:80,172.17.0.5:80 + 1 more...
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

1

```
redondo@miw:~$ curl 172.17.0.3
<html>
<body>
<h1>
It works!<p id="ip"></p>

<script>document.getElementById("ip").innerHTML = location.host</script>

</h1>
</body>
</html>
redondo@miw:~$ curl 172.17.0.4
<html><body><h1>It works!</h1></body></html>
redondo@miw:~$ curl 172.17.0.5
<html><body><h1>It works!</h1></body></html>
redondo@miw:~$ curl 172.17.0.6
<html><body><h1>It works!</h1></body></html>
```

2

Escalando manualmente una aplicación: Balanceo de carga

José Manuel
Redondo López

- **Esto desvirtúa el propósito de Kubernetes**
  - Tener un clúster escalable de forma transparente
- **Por ello es importante recordar los valores que usamos anteriormente cuando desplegamos un pod**
  - NodePort: Puerto al que mandamos la petición HTTP
  - Endpoint: La IP NAT de los pods escalados
- **El pod individual y el replicado serán accesibles de la misma forma**
  - Invocamos `curl` sobre el puerto: `curl $(sudo minikube ip):$NODE_PORT`
  - Y vemos que algunas peticiones son atendidas por la instancia original y el resto por sus réplicas

```
redondo@miw:~$ curl $(sudo minikube ip):$NODE_PORT
<html>
<body>
<h1>
It works!<p id="ip"></p>

<script>document.getElementById("ip").innerHTML = location.host</script>

</h1>
</body>
</html>
redondo@miw:~$ curl $(sudo minikube ip):$NODE_PORT
<html><body><h1>It works!</h1></body></html>
redondo@miw:~$ curl $(sudo minikube ip):$NODE_PORT
<html>
<body>
<h1>
It works!<p id="ip"></p>

<script>document.getElementById("ip").innerHTML = location.host</script>

</h1>
</body>
</html>
redondo@miw:~$ curl $(sudo minikube ip):$NODE_PORT
<html><body><h1>It works!</h1></body></html>
redondo@miw:~$ curl $(sudo minikube ip):$NODE_PORT
<html><body><h1>It works!</h1></body></html>
```

Escalando manualmente una aplicación: Eliminando réplicas

José Manuel
Redondo López

- **También podemos escalar a la baja el nº de pods, si disminuye el nº de peticiones**
  - Podemos crear todas las réplicas que queramos, siempre que el nodo aguante la carga de trabajo
  - Se usa el mismo comando anterior
    - `kubectl scale deployments/kubesample1 --replicas=2` **(1)**

```
redondo@miw:~$ sudo kubectl scale deployments/kubesample1 --replicas=2
deployment.apps/kubesample1 scaled
redondo@miw:~$ sudo kubectl get pods
NAME                           READY   STATUS        RESTARTS   AGE
kubesample1-8649c7d75c-4qpjv   1/1     Running       0          9h
kubesample1-8649c7d75c-95qvs   1/1     Running       0          7m17s
kubesample1-8649c7d75c-qxdk9   0/1     Terminating   0          7m17s
kubesample1-8649c7d75c-zn75l   0/1     Terminating   0          7m17s
```

1

- **Kubernetes destruirá los pods necesarios para ajustarse al nuevo desired state**
  - 2 de los 4 pods anteriores pasarán al estado terminating y finalmente se destruirán **(2)**
  - Podemos ejecutar `kubectl get deployments` para ver si ya se ha alcanzado el nuevo desired state **(2)**

```
redondo@miw:~$ sudo kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
kubesample1-8649c7d75c-4qpjv   1/1     Running   0          9h
kubesample1-8649c7d75c-95qvs   1/1     Running   0          7m43s
redondo@miw:~$ sudo kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
kubesample1   2/2     2            2           9h
```

2

Escalando manualmente una aplicación: Mantener el estado

José Manuel
Redondo López

- **El desired state da a los clústeres capacidades de "self-healing"**
  - Si los pods fallan aparecen otros en su lugar que realizan el mismo trabajo que los "fallecidos"
  - Ocurre si un pod se para manualmente o por un fallo en la aplicación o sus componentes
- **Se puede comprobar borrando manualmente un pod y viendo cómo reacciona Kubernetes**
  - Lo primero es obtener una lista de pods: `kubectl get pods` **(1)**
  - Luego asignamos uno de los pods listados a una variable de entorno
    - `POD_TO_KILL=kubesample1-6f7886b668-x8btw`
    - `echo POD_TO_KILL=$POD_TO_KILL` **(2)**

```
redondo@miw:~$ sudo kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
kubesample1-8649c7d75c-4qpjv   1/1     Running   0          9h
kubesample1-8649c7d75c-95qvs   1/1     Running   0          8m47s
redondo@miw:~$ POD_TO_KILL=kubesample1-8649c7d75c-95qvs
```

1

```
redondo@miw:~$ echo POD_TO_KILL=$POD_TO_KILL
POD_TO_KILL=kubesample1-8649c7d75c-95qvs
```

2

Escalando manualmente una aplicación: Mantener el estado

José Manuel
Redondo López

- Y así podemos borrarlo fácilmente: `kubectl delete pod $POD_TO_KILL` **(1)**
- Al comprobar el estado de los pods, veremos un pod en estado terminating
  - Este estado cambia rápidamente, por lo que seguramente no sea fácil verlo
- Si esperamos podemos ver que, aunque se ha borrado un pod, otro ha aparecido automáticamente
  - Se puede comprobar preguntando por los pods: nos encontramos con uno con un nombre distinto al que eliminamos y un `AGE` mucho menor **(2)**
- Es decir, que la capacidad de self-healing de Kubernetes ha funcionado

```
redondo@miw:~$ sudo kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
kubesample1-8649c7d75c-4qpjv   1/1     Running   0          9h
kubesample1-8649c7d75c-95qvs   1/1     Running   0          10m
redondo@miw:~$ sudo kubectl delete pod $POD_TO_KILL
pod "kubesample1-8649c7d75c-95qvs" deleted
```

1

```
redondo@miw:~$ sudo kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
kubesample1-8649c7d75c-4qpjv   1/1     Running   0          9h
kubesample1-8649c7d75c-8m4gh   1/1     Running   0          23s
```

2

Escalando manualmente una aplicación: Problemas

José Manuel
Redondo López

- **Pero esta forma de escalar tiene limitaciones**
  - Tenemos que saber cuántas replicas queremos tener en cada momento
  - Si la carga cambia, tenemos que monitorizarlo para identificarlo y reaccionar de manera acorde
- **Por tanto**
  - Debes se capaz **predecir la carga media de la aplicación en todo momento**
  - ¿Y si es más? ¿O menos? Es decir, ¿y si nos equivocamos?
  - O lo que es peor, ¿**Y si tenemos un pico de carga**?
    - ¿Y si la aplicación realmente solo tiene unos pocos "picos de carga" y el resto del tiempo consume mucho menos?
  - Tened en cuenta que algunos servicios de nube **cobran por uso**...
    - Más pods implica más coste, y si además no se están usando realmente…
- **Es cuando entra en juego la cuarta y última parte de esta introducción: el auto-escalado**

Auto-escalado de Aplicaciones

Escalado horizontal y escalado elástico

¿Por qué auto-escalar?

José Manuel
Redondo López

- El **desired state** es una estrategia adecuada si queremos tener control sobre el consumo máximo de recursos de la aplicación
- Pero no es capaz de responder a una situación bastante común: **Picos inesperados de tráfico**
  - Son anomalías que pueden ser más o menos frecuentes según el propósito de la aplicación dada
    - Por ejemplo, es bastante común en aplicaciones de venta de tickets
  - O también aplicaciones a las que se le haga una denegación de servicio
- Debe usarse con cuidado en proveedores de nube
  - Cuidado con el máximo nº de pods a escalar, para que no se dispare el coste demasiado
  - Una vez pasado el pico de tráfico, el clúster debe volver a su estado original: **escalado elástico**
  - Esto último también debería ocurrir en escenarios de muy bajo tráfico según horario
  - ¡Es muy útil para optimizar los costes si se configura bien!

¿Cómo se auto-escala?

José Manuel
Redondo López

- **El concepto de auto-escalado en Kubernetes se denomina Horizontal auto-scaling (hpa)**
  - Implica que Kubernetes puede escalar automáticamente el nº de pods que necesitamos
- **Para ello, usa el parámetro de la utilización de la CPU en los nodos**
  - Aunque puede cambiarse para usar otros valores
- **Consiste en dos partes: recursos y un controlador**
  - El controlador mira la utilización (o la métrica que se use) para comprobar que el nº de replicas encaja con la especificada
    - Si es necesario, aumenta o disminuye el nº de pods
    - Lo comprueba cada 15 segundos, aunque se puede configurar con el flag `--horizontal-pod-autoscaler-sync-period`
- **El nº de replicas generadas se calcula así**
  - `desiredReplicas = ceil[currentReplicas * (currentMetricValue / desiredMetricValue)]`

AUTO-ESCALANDO UNA APLICACIÓN

José Manuel
Redondo López

- No debe usarse el desired state, que solo tiene sentido en escalados manuales
- Para hacer auto-escalado necesitamos dar valor a 2 cosas
  - **min/max**: definimos un mínimo y un máximo de pods deseados
  - **CPU**: Fijamos un porcentaje de uso de la CPU que usamos como límite
    - Si supera ese valor indicado, inicia el escalado
- También necesitamos habilitar una serie de addons
  - Para saber los que tenemos habilitados hacemos: `minikube addons list`

```
redondo@miw:~$ sudo minikube addons list
|-----------------------------|----------|--------------|
|         ADDON NAME          | PROFILE  |    STATUS    |
|-----------------------------|----------|--------------|
| ambassador                  | minikube | disabled     |
| dashboard                   | minikube | disabled     |
| default-storageclass        | minikube | enabled 27 05|
| efk                         | minikube | disabled     |
| freshpod                    | minikube | disabled     |
| gvisor                      | minikube | disabled     |
| helm-tiller                 | minikube | disabled     |
| ingress                     | minikube | disabled     |
| ingress-dns                 | minikube | disabled     |
| istio                       | minikube | disabled     |
| istio-provisioner           | minikube | disabled     |
| kubevirt                    | minikube | disabled     |
| logviewer                   | minikube | disabled     |
| metallb                     | minikube | disabled     |
| metrics-server              | minikube | disabled     |
| nvidia-driver-installer     | minikube | disabled     |
| nvidia-gpu-device-plugin    | minikube | disabled     |
| olm                         | minikube | disabled     |
| pod-security-policy         | minikube | disabled     |
| registry                    | minikube | disabled     |
| registry-aliases            | minikube | disabled     |
| registry-creds              | minikube | disabled     |
| storage-provisioner         | minikube | enabled 27 05|
| storage-provisioner-gluster | minikube | disabled     |
|-----------------------------|----------|--------------|
```

Prerequisitos para hacer auto-escalado

José Manuel
Redondo López

- **Necesitamos Metrics server**
  - Habilita la monitorización y análisis de rendimiento del clúster
  - Proporciona métricas a través de una API
  - Es necesario para que el escalador horizontal de pods pueda leerlas
- **Si no estuviera activo, hacemos**
  - `minikube addons enable metrics-server`
- **Habilitamos las custom metrics arrancando minikube con el mismo driver que la primera vez**
  - `minikube --vm-driver=docker --extra-config kubelet.EnableCustomMetrics=true start`
  - ¡**No hace falta pararlo** si ya está en marcha!
- **¡Listos para auto-escalar!**

```
redondo@miw:~$ sudo minikube addons enable metrics-server
  The 'metrics-server' addon is enabled
```

```
redondo@miw:~$ sudo minikube --vm-driver=none  --extra-config kubelet.EnableCustomM
etrics=true start
😄  minikube v1.12.0 on Ubuntu 18.04 (vbox/amd64)
    Using the none driver based on existing profile
    Starting control plane node minikube in cluster minikube
    Updating the running none "minikube" bare metal machine ...
ℹ   OS release is Ubuntu 18.04.4 LTS
    Preparing Kubernetes v1.18.3 on Docker 19.03.12 ...
    ▪ kubelet.EnableCustomMetrics=true
    ▪ kubelet.resolv-conf=/run/systemd/resolve/resolv.conf
    Configuring local host environment ...

    The 'none' driver is designed for experts who need to integrate with an existin
g VM
    Most users should use the newer 'docker' driver instead, which does not require
 root!
    For more information, see: https://minikube.sigs.k8s.io/docs/reference/drivers/
none/

    kubectl and minikube configuration will be stored in /home/redondo
    To use kubectl or minikube commands as your own user, you may need to relocate
them. For example, to overwrite your own settings, run:

    ▪ sudo mv /home/redondo/.kube /home/redondo/.minikube $HOME
    ▪ sudo chown -R $USER $HOME/.kube $HOME/.minikube

    This can also be done automatically by setting the env var CHANGE_MINIKUBE_NONE
 USER=true
    Verifying Kubernetes components...
    Enabled addons: default-storageclass, metrics-server, storage-provisioner
    Done! kubectl is now configured to use "minikube"
redondo@miw:~$
```

AUTO-ESCALANDO UNA APLICACIÓN

José Manuel
Redondo López

Desplegar una aplicación

- Este paso no es necesario si ya tenemos funcionando la anterior (`kubesample1`)
- Conviene hacer `kubectl get pods` y `kubectl get services` para ver si siguen en marcha

Aplicar auto-escalado

- Para ello hacemos: `kubectl autoscale deployment kubesample1 --cpu-percent=50 --min=1 --max=10`
- Con esto aplicamos el auto-escalado al despliegue indicado con un **porcentaje de CPU** del 50% y valores **min** y **max** 1 y 10 respectivamente
- Esto puede leerse así: "*Si la carga de la CPU es mayor del 50%, crea un nuevo pod, pero siempre y cuando no haya más de 10. Si la carga de la CPU es pequeña, entonces vuelve gradualmente a un pod*"

```
redondo@miw:~$ sudo kubectl autoscale deployment kubesample1 --cpu-percent=50 --min
=1 --max=10
horizontalpodautoscaler.autoscaling/kubesample1 autoscaled
```

AUTO-ESCALANDO UNA APLICACIÓN

José Manuel
Redondo López

- **"Bombardeo" de peticiones**
  - Para probar el auto escalado tenemos que generar un gran nº de peticiones sobre nuestro despliegue y ver si Kubernetes es capaz de responder adecuadamente
  - Pero lo primero es comprobar el estado del Horizontal pod auto-escaler (`hpa`)
    - `sudo kubectl get hpa`
    - La columna `TARGETS` muestra el uso actual de la CPU/valor que inicia el escalado
    - La columna `REPLICAS` indica el nº de copias activas en ese momento
  - ¡Pero es muy importante que **no muestre <unknown> como en la imagen**!

```
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                TARGETS         MINPODS   MAXPODS   REPLICAS
   AGE
kubesample1   Deployment/kubesample1   <unknown>/50%   1         10        2
   26s
```

Auto-escalando una aplicación: Problemas de métricas

José Manuel
Redondo López

- **Si muestra `unknown`, significa que Kubernetes es incapaz de leer el valor de % de uso de la CPU**
  - Si no es capaz de hacerlo, no sabrá cuando está siendo sobre utilizada
  - Y, si no lo sabe, tampoco sabrá cuando tiene que auto-escalar
  - Por tanto, el auto-escalado **no funcionará** ☹
- **¿Qué ha pasado?**
  - El problema es que en nuestro despliegue actual no está configurado el tiempo que debe pasar entre peticiones que lean el % de uso de la CPU
  - Es decir, que el componente que obtiene este valor no lo hace porque no se le ha especificado la periodicidad con la que debe hacerlo
- **¿Cómo arreglarlo?**

AUTO-ESCALANDO UNA APLICACIÓN

José Manuel
Redondo López

- **Debemos editar la configuración del deployment actual**
  - Se puede hacer mientras está funcionando con `sudo kubectl edit deployment kubesample1` **(1)**
  - Aquí veremos el fichero `.yaml` del despliegue que hemos hecho
  - Podremos modificarlo y en unos segundos el deployment se actualizará con nuestros cambios
- **Debemos añadir la sección de (2)**
- **El editor usado es `vi`**
  - https://www.cs.colostate.edu/helpdocs/vi.html
  - Tenemos que ir a la posición de editar y pulsar `i`
  - Escribimos el texto
  - Grabamos con Escape y `:wq`
  - Para salir sin grabar Escape y `:q!`

1

```
operario@server1804:~/.minikube$ sudo kubectl edit deployment kubesample1
```

```
    matchLabels:
      app: kubesample1
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: kubesample1
    spec:
      containers:
      - image: httpd:2.4
        imagePullPolicy: IfNotPresent
        name: httpd
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: "2020-07-15T05:37:42Z"
    lastUpdateTime: "2020-07-15T05:37:44Z"
    message: ReplicaSet "kubesample1-8649c7d75c" has successfully progressed.
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
```

```
      resources:
        requests:
          cpu: 200m
```

2

AUTO-ESCALANDO UNA APLICACIÓN

José Manuel
Redondo López

- **Es extremadamente importante que nos aseguremos que el módulo hpa puede obtener métricas de CPU**
  - En caso contrario, no funcionarán las operaciones siguientes
- **Por tanto, debemos esperar hasta que `sudo kubectl get hpa` devuelva algo como esto**
  - Puede tardar un poco de tiempo (minutos) en cambiar de `<unknown>` a `0%`

```
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   0%/50%    1         10        1          7m5
6s
```

AUTO-ESCALANDO UNA APLICACIÓN

José Manuel
Redondo López

- Tenemos que “atacar” nuestro servicio, y lo podemos hacer fácilmente desde el propio host
- Comprobamos que podemos acceder al **deployment** que va a auto-escalar desde el host
- **Ahora usamos la herramienta `ab` (Apache Benchmark) para “machacar” el cluster a peticiones**
  - Ej.: `ab -n 50000 http://192.168.49.2:30180/`
  - Debemos poner la IP del clúster y el `NODE_PORT` directamente llamando a los comandos correspondientes
  - Las peticiones continuarán hasta que pulsemos CTRL+C
  - Es posible que necesites más peticiones y más paralelismo para hacer autoescalar al clúster, según la potencia de tu máquina

AUTO-ESCALANDO UNA APLICACIÓN

José Manuel
Redondo López

- **Si lo dejamos un tiempo, al ejecutar `kubectl get hpa` desde el 1er terminal veremos los resultados del auto escalado**
  - **La carga va creciendo** a medida que abrimos más terminales con ese script **(1)**
  - En la imagen **(2)** vemos que `TARGETS` **excede el valor umbral del 50%** fijado pasado un tiempo
    - Fuerza al sistema a crear otra réplica (`REPLICAS` = 2)
    - El sistema ha creado automáticamente 2 réplicas del pod por el aumento de peticiones **(3)**
  - **La carga sigue subiendo** (abrimos más terminales) y los nodos replicados vuelven a estar al límite **(4)**
  - Nuevamente **superan el límite** y se genera una réplica nueva
    - Bajando la carga media de los nodos otra vez **(5)**
  - El sistema tarda un tiempo en adaptarse a la subida de volumen: **los cambios no son inmediatos**

```
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   0%/50%    1         10        1          28m
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   38%/50%   1         10        1          28m
```

1

```
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   38%/50%   1         10        1          29m
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   71%/50%   1         10        1          29m
```

2

```
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   71%/50%   1         10        2          30m
```

3

```
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   48%/50%   1         10        2          30m
```

4

```
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   35%/50%   1         10        3          41m
```

5

AUTO-ESCALANDO UNA APLICACIÓN

José Manuel
Redondo López

- **Kubernetes tarda minutos en volver a su estado inicial al disminuir la carga**
  - Podemos ver cómo van disminuyendo el nº de pods y la carga de trabajo si ejecutamos periódicamente `kubectl get pods` **(1) (2)**
  - Hasta llegar al estado original **(3)**
  - La vuelta al estado original es paulatina y lenta (minutos) y no repentina
- **Se puede eliminar totalmente el auto-escalado de un deployment con**
  - `sudo kubectl delete hpa kubesample1`

1

```
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                 TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   31%/50%   1         10        3          43m
```

2

```
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                 TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   0%/50%    1         10        3          45m
```

3

```
redondo@miw:~$ sudo kubectl get hpa
NAME          REFERENCE                 TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
kubesample1   Deployment/kubesample1   0%/50%    1         10        1          49m
```

¿CREES QUE LO HAS ENTENDIDO TODO? ¡AUTOEVALÚATE!

José Manuel
Redondo López

?

- ***¿Comprendes que, sea cual sea el modo de escalado, K8s se encarga de toda la distribución de peticiones por el clúster de manera transparente?***
- **Escalado manual**
  - *¿Entiendes que el escalado manual consiste en decirle tú al clúster Kubernetes cuál es el desired state que quieres que tenga? (nº de pods activo)*
  - *¿Has comprendido cómo funciona el self-healing de un clúster Kubernetes?*
  - *¿Comprendes los problemas que tiene este modo de escalado?*
- **Escalado automático**
  - *¿Entiendes la diferencia entre el escalado manual y el automático, y el problema que resuelve?*
  - *¿Comprendes como un escalado automático mal configurado puede disparar tus costes?*
  - *¿Has entendido en qué consiste el concepto de "escalado elástico"?*
  - *¿Has entendido como configurar y desplegar el escalado?*
  - *¿Comprendes cómo tener métricas adecuadas de la CPU es vital para que funcione?*
  - *¿Estás seguro de haber comprendido cómo reacciona un clúster K8s que auto-escala ante un incremento o disminución de peticiones?*

< Ir al Índice

Fuente: Bing Chat AI

Herramientas >

Más allá... >

Para ampliar conocimientos...

¿Qué más cosas puedo mirar si quiero seguir aprendiendo?

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ VAMOS A VER EN ESTE APARTADO?

José Manuel
Redondo López

- **Este apartado de cierre de la presentación contiene dos cosas**
  - Una mención a **herramientas adicionales** que te permiten mejorar la gestión del clúster K8s y hacer más cosas sobre el
  - **Enlaces para estudiar más** sobre K8s

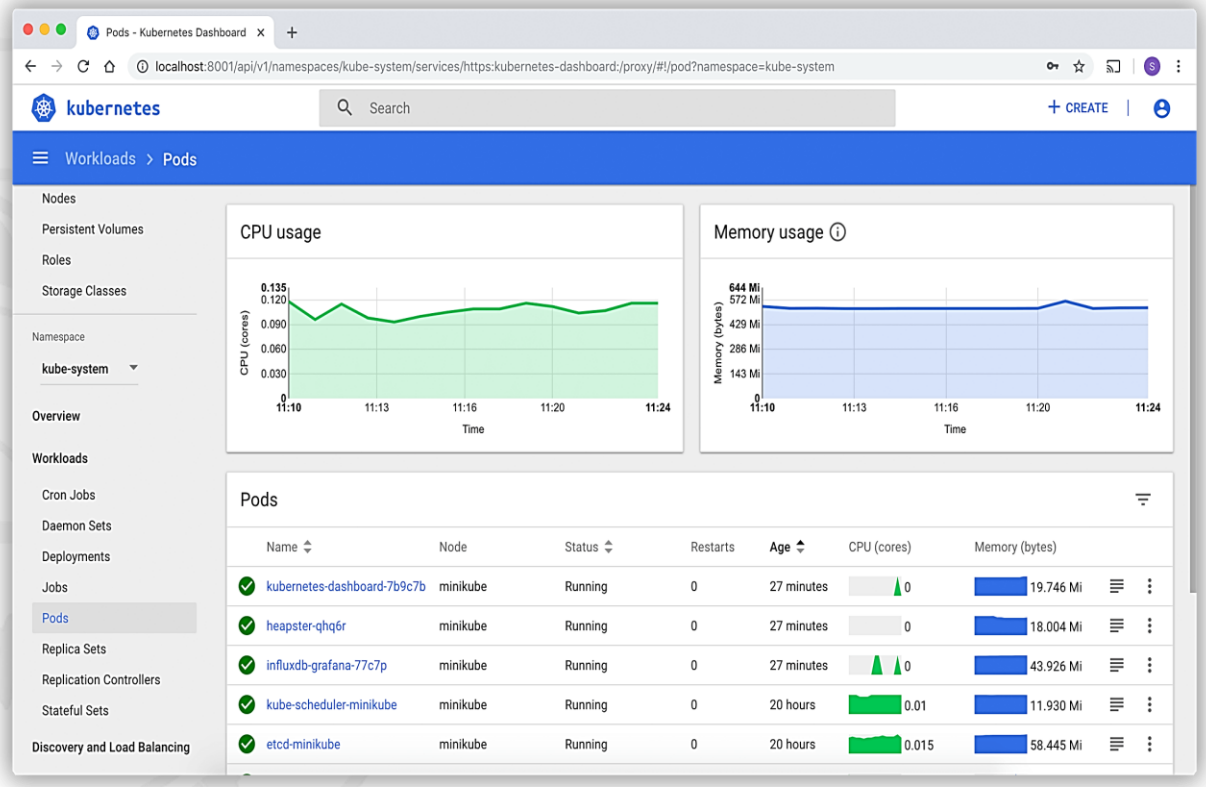
Herramientas complementarias a K8s

Distintas herramientas que nos pueden facilitar más la gestión de K8s

DASHBOARD

José Manuel
Redondo López

- **Es un interfaz gráfico vía web para Kubernetes que permite**
  - Desplegar aplicaciones contenerizadas en un clúster
  - Ver posibles errores que se produzcan
  - Gestionar los recursos del clúster
  - Ver información de las aplicaciones ejecutándose
  - Crear o modificar elementos de un clúster Kubernetes, como los deployments vistos
- **Con esta interfaz se simplifica el realizar operaciones típicas**
  - Escalado de deployments, reiniciar pods, desplegar aplicaciones…
- **Se accede a él con `minikube dashboard`**
- **Más información**
  - https://minikube.sigs.k8s.io/docs/handbook/dashboard/



Fuente: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

Helm

José Manuel
Redondo López

- **Es un administrador de paquetes para Kubernetes, es decir, una herramienta para administrar aplicaciones que se ejecutan en él**
  - Proporciona un conjunto de operaciones que son útiles para administrar aplicaciones: inspeccionar, instalar, actualizar y eliminar
  - Se compone de dos partes, el cliente (Helm) y el servidor (Tiller)
- **Tiene Helm Charts: Aplicaciones listas para desplegar**
  - Paquetes de recursos de Kubernetes preconfigurados que describen cómo administrar una aplicación
  - Tienen metadatos que describen la aplicación y la infraestructura necesaria para operarla con primitivas de Kubernetes
  - Cada uno hace referencia a una o varias imágenes de contenedor de aplicaciones que contienen el código de aplicación que se va a ejecutar
  - Contienen al menos estos dos elementos:
    - Una descripción del paquete (`chart.yml`)
    - Una o más plantillas, que contienen archivos de manifiesto de Kubernetes

Helm

José Manuel
Redondo López

- **Usar Helm en profundidad excede el ámbito de esta asignatura, pero los siguientes materiales os pueden ayudar a iniciaros con él**
  - Podéis probar sobre nuestro clúster minikube local
  - Las prácticas contienen un ejemplo de despliegue simple para "pillar el feeling" ☺
- **Más información**
  - **Web oficial**: https://helm.sh/
  - Estructura de un chart: https://helm.sh/docs/topics/charts/
  - Ejemplo: https://medium.com/htc-research-engineering-blog/a-simple-example-for-helm-chart-fbb5c7208e94
  - Crear un repo con un chart: https://helm.sh/docs/topics/chart_repository/

SKAFFOLD

José Manuel
Redondo López

- **Skaffold es una herramienta de línea de comandos que facilita el continous development de aplicaciones Kubernetes**
  - Recorre las fuentes de la aplicación local y lo ejecuta en clústeres de Kubernetes locales o remotos
  - Skaffold controla el flujo de trabajo para crear, insertar e implementar la aplicación
- **Algunas de sus funciones son**
  - **Desarrollo rápido local** de Kubernetes: Detecta los cambios en el código fuente y controla el proceso de compilación, subida y despliegue
  - Sus proyectos **funcionan en todas partes**: Es sencillo compartirlos con otras personas
  - **Configuración declarativa única** para un proyecto (skaffold.yaml)
  - Skaffold puede **descubrir los archivos necesarios** y generar su propio archivo de configuración (skaffold init)
- **Más información**
  - https://skaffold.dev/docs/
  - https://skaffold.dev/docs/quickstart/
  - https://github.com/kameshsampath/skaffold-quarkus-helloworld/blob/master/README.adoc

KUSTOMIZE

José Manuel
Redondo López

- **Gestor de configuración nativo de Kubernetes**
  - Implementa una forma de **personalizar la configuración de aplicaciones sin plantillas**
  - Esto simplifica el uso de aplicaciones
  - Se integra en `kubectl` como `apply -k`
- **Características**
  - Enfoque **puramente declarativo** para la personalización de la configuración
  - Gestiona un número arbitrario de configuraciones de Kubernetes distintas
  - Cada artefacto que usa Kustomize es un fichero YAML y puede ser validado y procesado como tal
  - Kustomize fomenta un flujo de trabajo fork/modify/rebase estilo GitHub
- **Más información**
  - https://kustomize.io/
  - https://github.com/kubernetes-sigs/kustomize/blob/master/examples/helloWorld/README.md

KUBE-DNS

José Manuel
Redondo López

- **Kubernetes DNS incorpora un pod y un DNS en el clúster**
  - Lo mencionamos antes en el apartado de cómo visibilizar servicios
  - Configura los kubelets para indicar a los contenedores individuales que usen la dirección IP de ese servicio DNS para resolver sus nombres DNS
  - A cada servicio definido del clúster (incluido el propio DNS) se le asigna un nombre DNS
- **Por defecto las búsquedas DNS de un pod incluyen el propio espacio de nombres del pod y el dominio por defecto del clúster**
  - Si un servicio `serv1` está en el namespace `miservicio`, un pod que se ejecuta en el mismo namespace puede buscar este servicio haciendo una consulta DNS para `serv1`
  - Un pod que se ejecuta en el namespace `misotrosservicios` puede buscar este servicio mediante una consulta DNS para `miservicio.serv1`
- **Más información**
  - https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/

Transición a la nube

José Manuel
Redondo López

- **Minikube** es una implementación de **Kubernetes** local
- Pero nos permite hacer una transición a clúster **Kubernetes** en distintos proveedores de nube (que admiten varios nodos) sencilla
  - Se sigue usando `kubectl`, sólo cambia la infraestructura que hace el **hosting** de contenedores
  - El paso de **minikube** a GKE o un clúster **on-premises** es trivial, simplemente cambiando el método para exponer servicios (`LoadBalancer` o `NodePort`)
- Existen guías que facilitan la transición a los principales proveedores de nube que admiten **Kubernetes**
  - **Get Started with the Google Kubernetes Engine (GKE)**
    - https://docs.bitnami.com/google/get-started-gke/
    - https://cloud.google.com/kubernetes-engine/
  - **Get Started with the Azure Kubernetes Service (AKS)**
    - https://docs.bitnami.com/azure/get-started-aks/
  - **Get Started with the Amazon Elastic Container Service for Kubernetes (EKS)**
    - https://docs.bitnami.com/aws/get-started-eks/

Más allá de la introducción

Para saber más...

¿Sólo Docker?

José Manuel
Redondo López

- **Aunque en esta presentación (y tradicionalmente) se use Docker como plataforma de contenedores, ya hemos dicho que se ha dejado de soportar en favor de otras**
  - Cri-O: Es un entorno para la creación de contenedores de bajo consumo de recursos
    - https://cri-o.io/
  - Containerd: Otro entorno para la creación de contenedores que destaca por su robustez, portabilidad y por haber sido certificado Cloud Native Computing Foundation
    - https://containerd.io/
- **Aunque no vayamos a verlos en la asignatura por falta de tiempo, ¡saber que existen siempre es bueno de cara al futuro! ☺**

BUILDAH

José Manuel
Redondo López

- **Es una herramienta de línea de comandos para crear imágenes que sigan los estándares de la Open Container Initiative (OCI)**
  - Se puede utilizar con Docker, Podman, Kubernetes y otros frameworks de contenedores compatibles
  - Se utiliza para crear, compilar, administrar, ejecutar imágenes y contenedores
- **Con Buildah se puede crear un contenedor a partir de una imagen o desde cero**
  - También se puede crear una imagen desde un contenedor que esté en ejecución o mediante una Dockerfile
  - Puede crear las imágenes en formato OCI o en el formato Docker, así como modificar y eliminar contenedores e imágenes
- **No necesita un daemon para funcionar, por lo que es especialmente útil en entornos CI/CD**
- **Más información**
  - https://buildah.io/
  - **Tutorial básico de uso**: https://linuxhandbook.com/buildah-basics/

Istio

José Manuel
Redondo López

- **Las plataformas en nube proporcionan muchos beneficios a las organizaciones que las usan**
- **Pero incrementan la dificultad de manejar despliegues grandes basados en microservicios**
  - Especialmente en implementaciones híbridas y multi-nube extremadamente grandes
  - Para las cuales es típico usar también Terraform (siguiente tema)
- **`Istio` reduce la complejidad de este tipo de tareas facilitando conectar, proteger, controlar y observar servicios**
  - Dicho de otra forma, permite controlar los servicios de una aplicación distribuida de manera transparente
  - Tiene además un API que permite integrar sus funciones con cualquier plataforma de log o telemetría
- **Más información**
  - https://istio.io/docs/concepts/what-is-istio/
  - https://github.com/istio/istio/tree/master/samples/helloworld

Más información

José Manuel
Redondo López

- Cheatsheet de **Kubernetes**
  - https://kubernetes.io/docs/reference/kubectl/cheatsheet/
- Descripción detallada del funcionamiento de **Kubernetes**
  - https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
- Alternativas a `minikube` como **Kubernetes** local
  - **Microk8s:** https://www.adictosaltrabajo.com/2018/09/27/kubernetes-en-local-con-microk8s/
  - **Kind**: https://kubernetes.io/docs/setup/learning-environment/kind/
- **CI/CD: How To Set Up a CD Pipeline with Spinnaker on DigitalOcean Kubernetes**
  - https://www.digitalocean.com/community/tutorials/how-to-set-up-a-cd-pipeline-with-spinnaker-on-digitalocean-kubernetes
- **Azure Kubernetes Service**
  - Descripción de **Kubernetes**, sus partes y cómo funciona: https://azure.microsoft.com/en-gb/topic/what-is-kubernetes/?wt.mc_id=devto-blog-chnoring
  - Cuenta gratuita: https://azure.microsoft.com/en-gb/free/?wt.mc_id=devto-blog-chnoring
  - https://azure.microsoft.com/en-gb/services/kubernetes-service/?wt.mc_id=devto-blog-chnoring
  - https://docs.microsoft.com/en-gb/azure/aks/?wt.mc_id=devto-blog-chnoring
  - Mejores prácticas: https://docs.microsoft.com/en-us/azure/aks/best-practices?wt.mc_id=devto-blog-chnoring

Más información

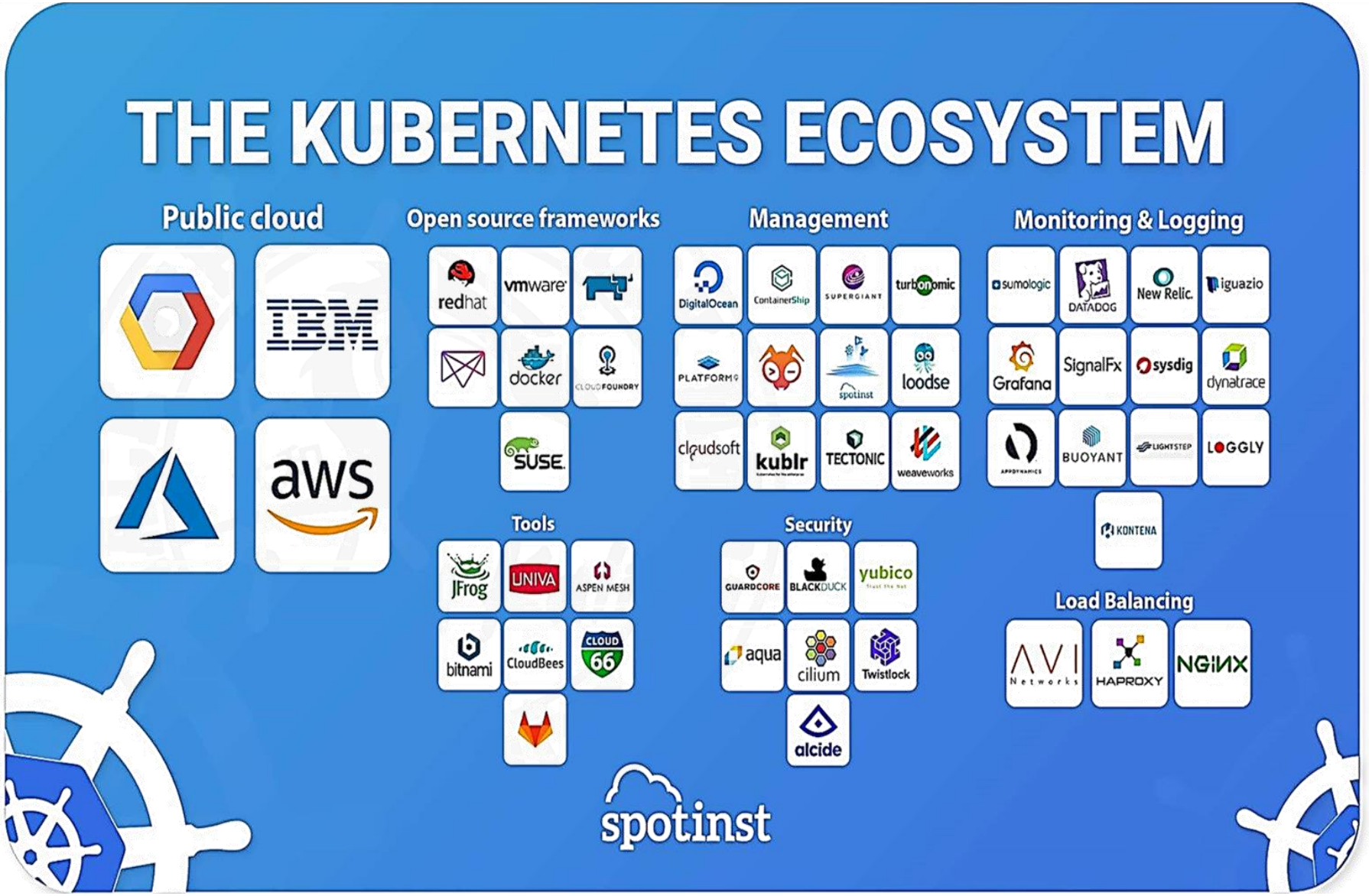
José Manuel
Redondo López

- Aplicación sin estado con **deployments** (incluye crear y aplicar ficheros YAML)
  - https://kubernetes.io/es/docs/tasks/run-application/run-stateless-application-deployment/
- Introducción básica de la tecnología
  - http://www.javiergarzas.com/2016/02/kubernetes-for-dummies-explicado-en-10-minutos.html
- Tutoriales
  - Oficiales (incluye un "Hello World!"): https://kubernetes.io/docs/tutorials/
  - Creación de clúster básicos: https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/
- Asignación de recursos de CPU a contenedores y pods
  - https://kubernetes.io/docs/tasks/configure-pod-container/assign-cpu-resource/
- Creación de clúster Kubernetes con KubeADM
  - https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/
- How to deploy a Kubernetes cluster with Multipass
  - https://www.techrepublic.com/article/how-to-deploy-a-kubernetes-cluster-with-multipass/
- Curso completo de Kubernetes
  - https://www.edx.org/es/learn/kubernetes/the-linux-foundation-introduction-to-kubernetes

Pero esto es un ecosistema mucho más grande...

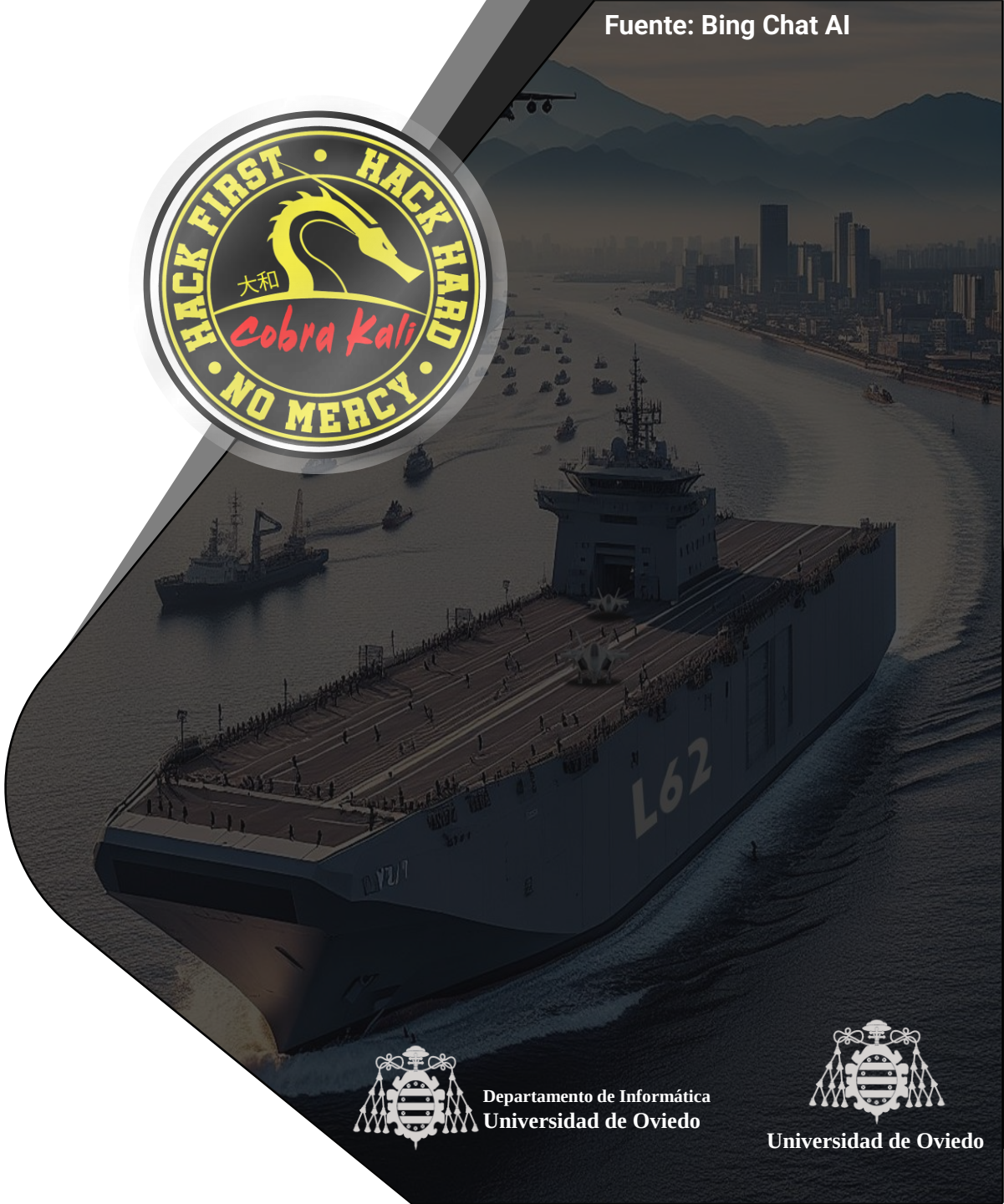
José Manuel
Redondo López

- ¡Creíste que ibas a ser un experto en K8s, but it was me, Dio!
- ¡Ser un experto en Kubernetes requiere estudiar mucho! ☺



Fuente: https://spot.io/blog/kubernetes-ecosystem

INTRODUCCIÓN A KUBERNETES



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo