Kubernetes

1. Arquitectura

Índice

Objetivos

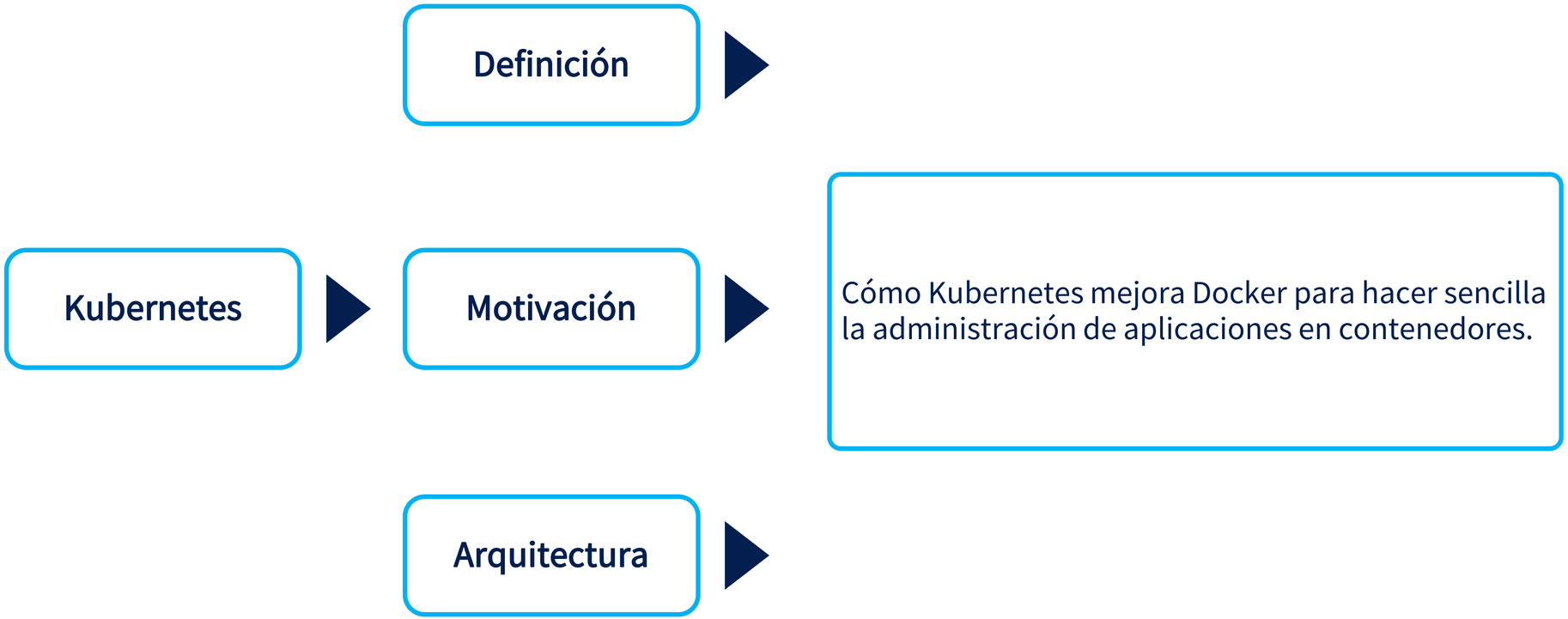
Conocer qué es Kubernetes

Situarlo en el ecosistema de Docker

Comenzar rápido y desplegar nuestras primeras apps

Visualizar las ventajas de utilizar esta tecnología

Mapa Conceptual



Qué es Kubernetes

Orquestador de contenedores

- Administra el despliegue de contenedores
- Preparado para entornos de producción

Kubernetes dispone de herramientas para levantar contenedores:

- En alta disponibilidad
- De forma segura
- Aislados de otras cargas de trabajo
- Conectados entre sí
- Expuestos a Internet
- Conectados con servicios en la nube

Qué es Kubernetes

En definitiva, es una plataforma que automatiza la operación de contenedores en entornos de producción.

Si por ejemplo tenemos una imagen Docker con el código para un blog, necesitaremos:

- Poder actualizarlo sin pérdida de servicio
- Protegernos frente a errores en la aplicación
- Exponerlo a Internet de forma segura
- Proteger las máquinas en las que se ejecuta el blog
- Poder volver a versiones anteriores
- Adaptar los recursos a la carga de visitas

Todo esto es lo de lo que nos abstrae Kubernetes. Hacerlo simplemente con Docker sería mucho más complejo.

Algunos datos

Comenzó dentro de Google, inspirado en su sistema Borg, que lleva en producción más de 15 años, ejecutando todos los servicios de Google.

Una primera versión fue liberada en 2015. A partir de ahí el proyecto ha crecido gracias a la comunidad open source.

Una gran cantidad de empresas utiliza y contribuye al desarrollo de Kubernetes hoy en día. Más de 2400 colaboradores individuales.

Se ha creado un ecosistema de herramientas enorme a su alrededor. Tanto Kubernetes como muchos otros proyectos relacionados son administrados por Cloud Native Computing Foundation (CNCF).

Kubernetes significa timonel en griego antiguo. Es decir, “el que controla el barco que transporta los contenedores”.

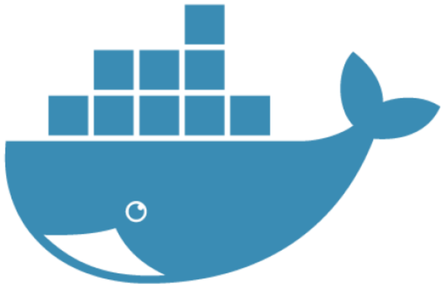
Kubernetes & Docker

Kubernetes es sólo el orquestador. Necesita una capa por debajo que sea el motor que ejecuta contenedores. Este motor en general es Docker

No puede ejecutar ninguna carga de trabajo que no esté definida como un contenedor. Es la base sobre la que se levanta Kubernetes.

Las ventajas de usar contenedores son principalmente

- Movilidad (poder ejecutarlo en cualquier sistema, con el mismo Dockerfile)
- Compartición de recursos (poder ejecutar muchos contenedores aislados en la misma máquina, sin necesidad de tener una máquina por servicio).



Kubernetes aprovecha estas ventajas para crear un sistema de despliegue de aplicaciones en producción seguro y adaptable.

Por qué ha tenido éxito

Production-ready: Los clientes grandes necesitan sistemas con garantías en producción. Kubernetes no es sólo para desarrollar un aplicación. Está pensado desde el principio para desplegar grandes cargas de trabajo.

Escalabilidad: Ha demostrado su utilidad tanto en aplicaciones pequeñas como de alcance mundial (por ejemplo, Pokemon Go funcionó tan bien a escala global gracias a Kubernetes).

Extensibilidad: Es sencillo crear nuevas herramientas, lo que hace que su utilidad se multiplique.

Abierto: Todo el código es público, para siempre, lo que ha facilitado su adopción y adaptación a diferentes casos de uso

Comunidad: Gracias a esa extensibilidad y al código abierto (también al impulso inicial de Google) se ha creado una enorme comunidad alrededor de Kubernetes que hace que evolucione y se adapte muy rápido, sin necesidad de contratar soporte.

Ventajas

Aparte de todas las ventajas de utilizar contenedores, utilizando Kubernetes se consigue:

- **Escalado, balanceo entre nodos, gestión de recursos, autoreparación, monitorización, auditoría**, etc. automatizados
- **Sistema declarativo**: Defines las cargas de trabajo y Kubernetes se encarga de realizar todas las acciones necesarias para ejecutarlas. No es necesario programar.
- **Movilidad**: Las definiciones son estándar y estables. Junto a la movilidad de los contenedores, hace que una misma definición se pueda utilizar en cualquier clúster de Kubernetes.
- **Código abierto**: Ahorros en licencias, evitar compromisos con proveedores, posibilidad de desplegarlo uno mismo, posibilidad de adaptar el código si es necesario, etc

Ventajas

- **Ahorro de recursos**: La posibilidad de levantar muchas cargas de trabajo en un solo nodo hace que se aprovechen mejor los recursos (CPU, memoria), ahorrándolos de la factura final. La enorme escalabilidad de Kubernetes también hace que puedas adaptar los recursos a la carga real en cuestión de segundos.

- **Multi-cloud y cloud híbrida**: Las cargas de Kubernetes se pueden ejecutar en cualquier clúster con mínimas adaptaciones. Esto permite que las aplicaciones se puedan mover de un proveedor cloud a otro, o de una nube privada a una pública, sin complicaciones. Además está disponible como servicio en todos los grandes proveedores.

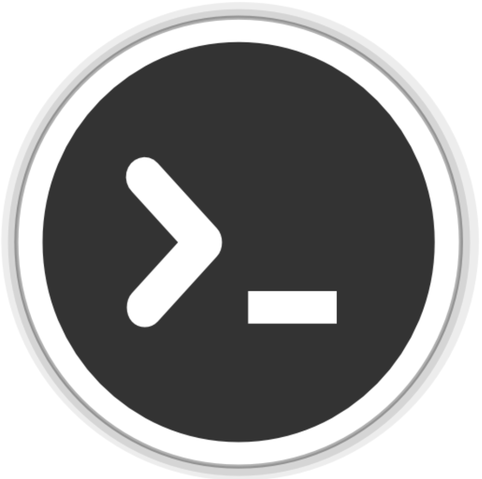
- **Adaptable**: Cualquier tipo de contenedor, con prácticamente cualquier necesidad, puede ser ejecutado en Kubernetes.

- **Un sistema para todo**: aplicaciones, configuraciones, elementos de red, escalado, almacenamiento, credenciales, seguridad… Todo lo relativo a una aplicación puede ser definido en Kubernetes.

Suficiente teoría

https://www.katacoda.com/courses/kubernetes/playground

Los *playgounds* de Katacoda ofrecen entornos ya preparados y gratuitos para ganar experiencia rápidamente en tecnologías nuevas.



1. Click *Start Scenario*
2. Click *launch.sh* (el script ya preparado que pone en marcha el entorno)

¡Listo para empezar a explorar!

Expande la terminal para estar más cómodo . Pulsando *continue* se cierra el entorno.

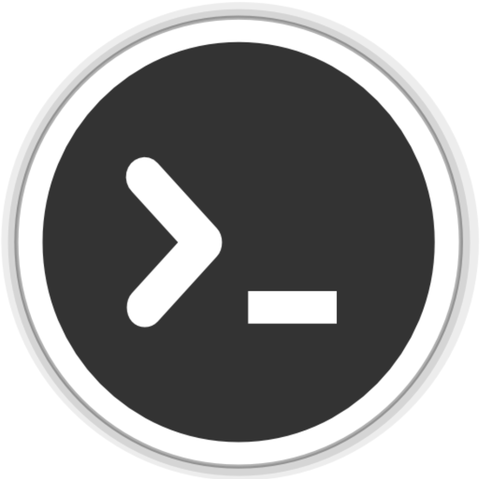
Suficiente teoría

https://www.katacoda.com/courses/kubernetes/playground

En el entorno de prueba verás dos terminales. Cada uno representa un nodo distinto de Kubernetes. En este caso nos ofrece un nodo master (arriba) y otro node01 (abajo).

Vamos a utilizar el terminal de arriba (master) para interactuar con el clúster.

El terminal de abajo, node01, lo usaremos más adelante.



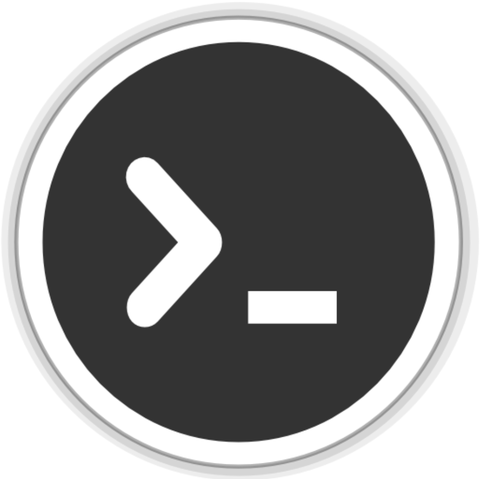
Ejecuta un primer contenedor

kubectl run hello-world --image hello-world --restart Never

Con este comando hemos creado un **Pod**, es el objeto fundamental en el entorno de Kubernetes.

kubectl es el CLI con el que interactuamos con el clúster.

Nota: kubectl run es sólo un atajo para principiantes, no lo volveremos a utilizar ;)



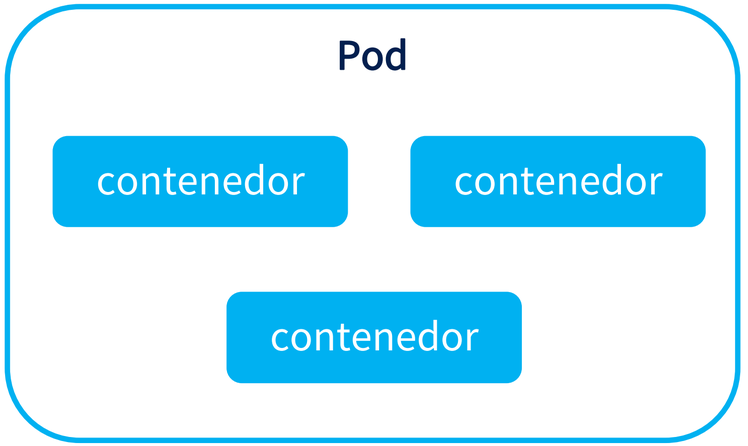
Pod – El objeto fundamental

Un Pod en Kubernetes es un objeto que define una ‘carga de trabajo’.

Es decir, un Pod es uno o varios contenedores que son ejecutados conjuntamente en Kubernetes.

Es el equivalente a docker run

En la definición del Pod están todos los parámetros que definen la ejecución del contenedor: imagen, puertos, variables de entorno, recursos, permisos, volúmenes… Cómo ves, todos los parámetros que le puedes pasar a docker run para configurar tu contenedor.



Explorando nuestro pod

kubectl get pod

Con este comando listamos los pods que están corriendo en Kubernetes.

kubectl logs hello-world

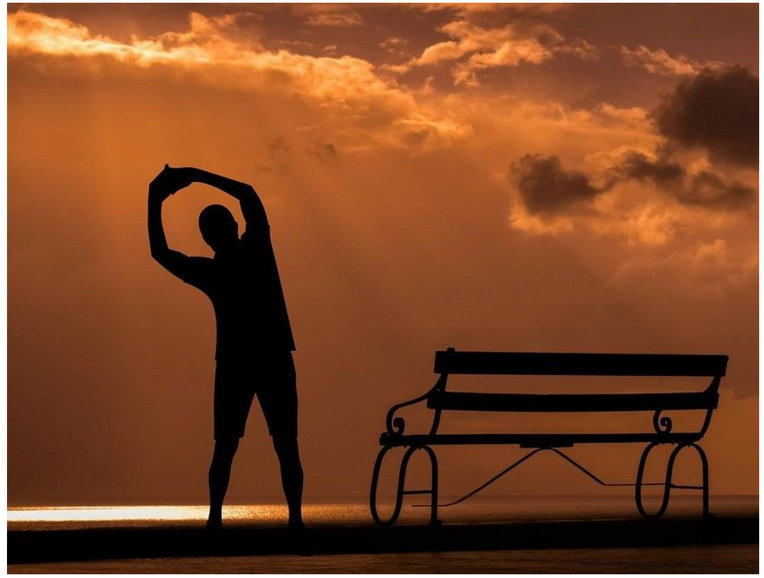
Con este comando obtenemos los logs de ese pod. Es el equivalente a docker log. Quizás tengas que esperar a que se levante del todo el pod (STATUS = Running).

Intenta desplegar un contenedor. más con un servidor Tomcat (--image tomcat) y ver cómo se ejecuta

Descansa 10 minutos

Hasta ahora:

- Hemos definido Kubernetes
- Lo hemos situado en el mapa con respecto a Docker
- Nos hemos motivado para aprenderlo
- Tenemos un entorno en la nube a un click
- Hemos ejecutado el primer contenedor



Hasta el final del módulo:

- Conoceremos cómo definir pods más complejos
- Lo veremos de forma práctica en el entorno de prueba

¿Qué idioma habla Kubernetes? YAML

Nunca se utilizan comandos para desplegar pods. En Kubernetes **todo** se define en archivos de configuración tipo YAML.

- YAML es un lenguaje de *marcado* (*markup language*)
- Puedes llamarlo lenguaje de *configuración*
- Quiere decir que no es propiamente un lenguaje de programación. Sólo sirve para describir configuraciones. No puedes escribir acciones en YAML
- Es el lenguaje básico con el que definir objetos en Kubernetes, que luego serán subidos al clúster
- Kubernetes interpreta estos documentos para ejecutar contenedores, configurar redes, etc

YAML

Así describiríamos el mobiliario de un piso en YAML:

```
catalogo: IKEA 2020
tipo: piso
habitaciones:
- cocina
- dormitorio:
    cama:
        tipo: neiden
        tamaño: 140
- salon:
    libreria: brimnes
    sofa: ektrop
```

Objetos YAML en Kubernetes

Todos los objetos tienen la misma estructura principal:

```
apiVersion:  <La API de mi objeto / el catálogo>
kind:  <El tipo de objeto dentro del catálogo>
metadata:
     name:  <Nombre del objeto>
spec:
  <
  ....
  Todas las variables del objeto
  ....
  >
```

Ejemplo: YAML para un pod

Una explicación de los campos que hemos visto:

- **apiVersion**: Kubernetes se compone de APIs, que son distintas librerías o catálogos que definen objetos (hay una para objetos de almacenamiento, otra para objetos de red, etc.). El objeto pod es fundamental por lo que está en la librería (API) básica, llamada simplemente v1.
- **kind**: Es el tipo de objeto dentro de la librería que vamos a definir. Dentro de v1 hay muchos objetos distintos para definir aplicaciones. En este caso, nuestro YAML define un pod.
- **metadata**: Dentro de este campo se incluyen todos los campos con información extra sobre el objeto. El campo name es obligatorio. Ya supones que especifica el nombre con el que se creará el objeto dentro de Kubernetes.
- **spec**: Aquí dentro va toda la información sobre el objeto, con el formato correcto según la librería y tipo que hayamos especificado. Todas las referencias para todas las librerías están disponibles. En este módulo vamos a ver muchas de ellas.

Ejemplo real: YAML para un pod

Este YAML es correcto y despliega un pod con un contenedor de Nginx que escucha el puerto 80:

```
apiVersion: v1
kind: Pod
metadata:
    name: nginx
spec:
    containers:
    - name: nginx
      image: nginx:latest
      ports:
      - containerPort: 80
```

Desplegando nuestro pod

Volvemos a nuestro entorno de prueba:

https://www.katacoda.com/courses/kubernetes/playground

nano nginx.yaml

nano es un editor de código sencillo en la terminal, que vamos a usar para crear nuestro fichero .yaml en el entorno de prueba.

Copiamos el código de la siguiente diapositiva. Pulsamos ctrl+o para guardarlo y ctrl+x para salir.

nginx.yaml

```
apiVersion: v1
kind: Pod
metadata:
    name: nginx
spec:
    containers:
    - name: nginx
      image: nginx:latest
      ports:
       - containerPort: 80
```

Explorando nuestro pod

Para desplegarlo usamos el comando.

```
kubectl apply -f nginx.yaml
```

apply es el comando básico para desplegar objetos en Kubernetes. con -f estamos especificando el fichero YAML que vamos a desplegar.

```
kubectl get pod
```

Ya conocemos este comando para listar todos los pods. Espera a que el pod esté en estado Running.

Acceder al puerto del pod

kubectl get pod -o wide

Con la opción -o wide listamos los contenedores con más información. En este caso podemos ver la IP y el nodo al que está asignado.

Una vez el pod tiene asignada una IP, podemos realizar una petición al servidor nginx. Ten en cuenta que esta IP es privada. Veremos cómo exponerlo a Internet en otro módulo.

curl http://<IP ADDRESS>:80

Ver los logs del pod

Ejecuta curl unas cuantas veces más.

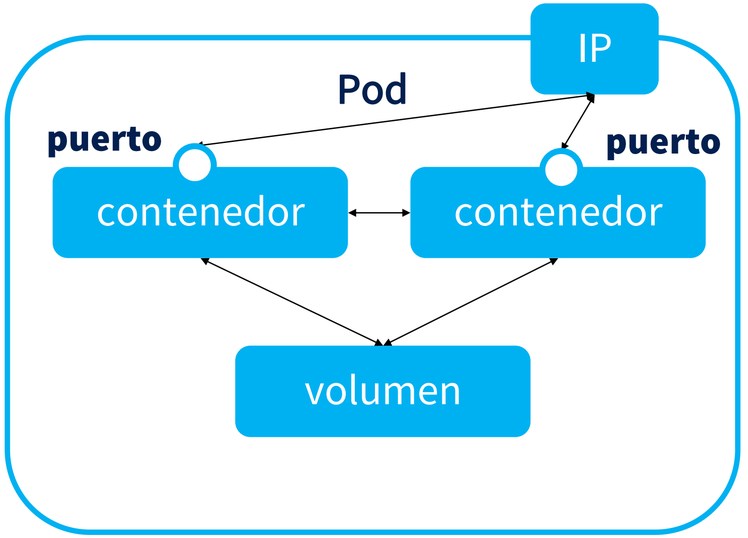
A continuación vamos a inspeccionar los logs del contenedor:

```
kubectl logs nginx
```

Vemos que tenemos varias líneas de log de las peticiones que hemos hecho.

Pods

Un pod es el objeto básico de cargas de trabajo en Kubernetes.



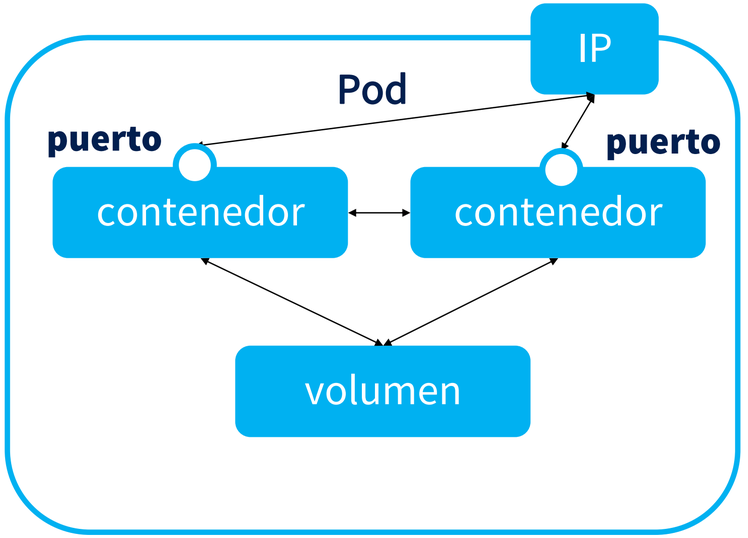
Algunas características

- Los contenedores del mismo pod siempre se ejecutan en el mismo nodo
- Están aislados del resto de contenedores que corren en el mismo nodo
- Los contenedores del mismo pod se pueden comunicar utilizando *localhost* o 127.0.0.1
- Comparten una misma IP para comunicarse con otros pods
- Pueden compartir directorios entre sí. Esto no se puede hacer entre distintos pods

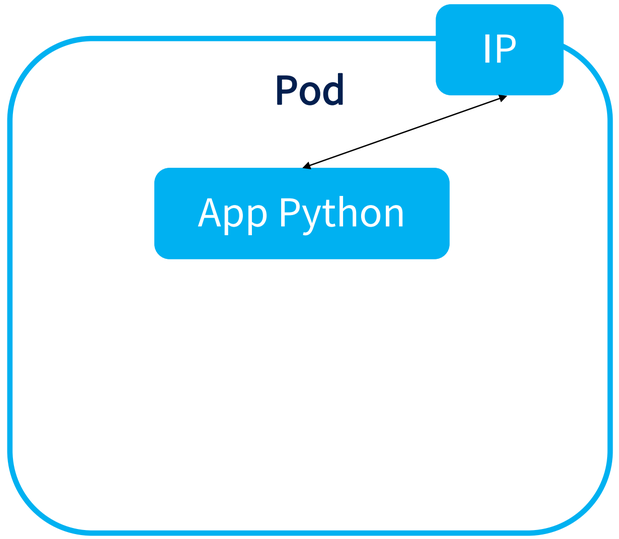
Pods

A nivel técnico, un pod es simplemente un conjunto de contenedores que se ejecutan en el mismo *userspace*. Es decir, que comparten los mismos recursos de red (misma IP, se comunican mediante *localhost*), y de almacenamiento (pueden compartir directorios entre sí).

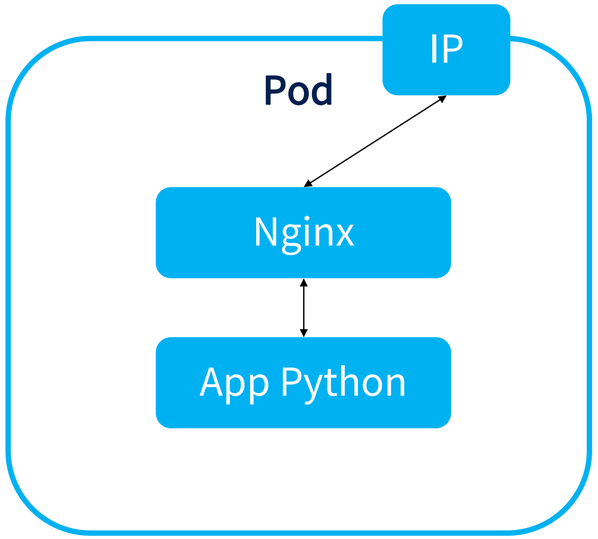
Los pods son cargas de trabajo efímeras. Si un pod falla por cualquier razón, no es levantado de nuevo. Para esto existen otros objetos superiores (ReplicaSets, StatefulSets) que veremos en este módulo.



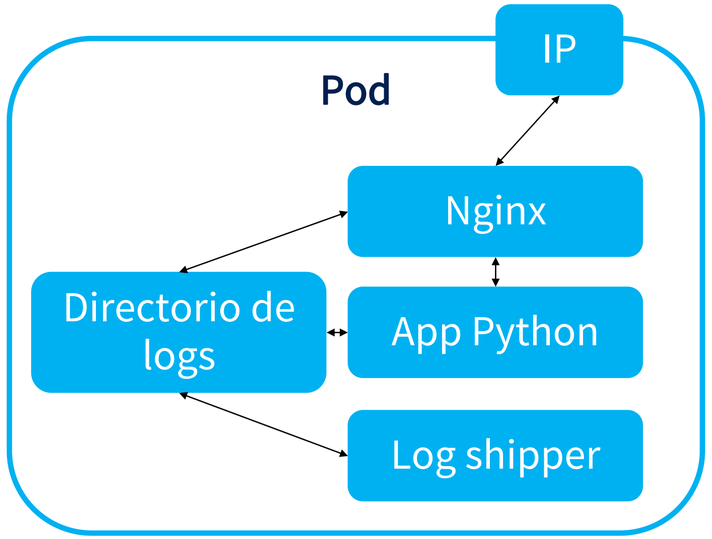
Ejemplos de Pods



Aplicación simple que se comunica directamente con otros pods



Pod con un *reverse proxy* por el que pasan las peticiones a la app



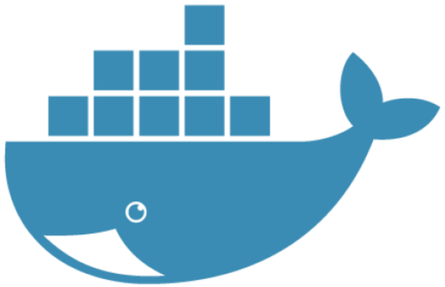
Con un contenedor más que lee los logs de un directorio compartido para gestionarlos mejor

Inspecciona el contenedor

Aunque hemos ejecutado los comandos en el nodo master, el pod con el servidor nginx se ha levantado en el nodo node01 (lo podemos ver con kubectl get pod -o wide).

Kubernetes no aplica los despliegues en el mismo nodo en el que se ejecutan los comandos. Kubernetes trata todos los nodos como un clúster, y decide en qué nodo desplegar los contenedores de la forma más eficiente.

En este caso ha estimado que lo mejor es desplegar nginx en el nodo node01.

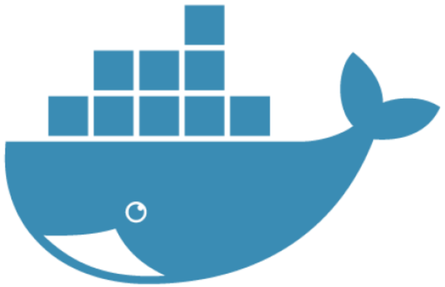


Inspecciona el contenedor

Por tanto podemos observar el contenedor de Docker en ese nodo. Lista todos los contenedores en el terminal de node01 con docker ps. Verás que entre ellos se encuentra uno cuya imagen es precisamente nginx.

Observando el ID de ese contenedor, puedes obtener todos los detalles sobre cómo se está ejecutando con docker inspect <ID del contenedor>.

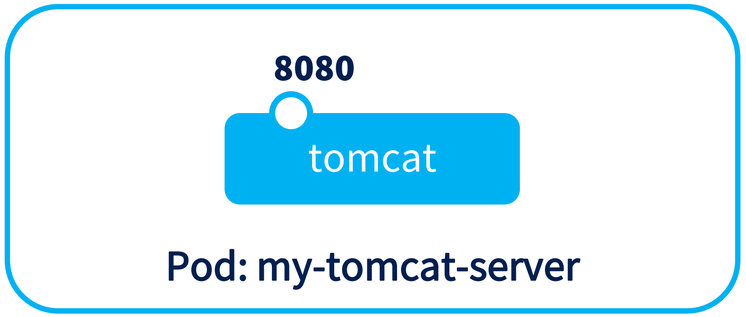
Como ves, el pod de Kubernetes al final se traduce en parámetros para ejecutar contenedores como con docker run. En siguientes módulos profundizaremos más en este tema.



Repetimos con un servidor tomcat

tomcat.yaml

```
apiVersion: v1
kind: Pod
metadata:
    name: my-tomcat-server
spec:
    containers:
    - name: tomcat
      image: tomcat:latest
      ports:
      - containerPort: 8080
```



Repetimos con un servidor tomcat

Recordamos los comandos que ya hemos visto:

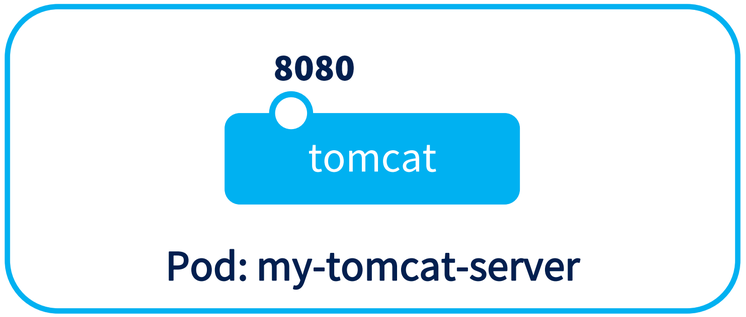
kubectl apply -f tomcat.yaml

kubectl get pod -o wide

curl http://<POD IP>:8080

(veremos una página de error 404, pero es una respuesta normal de tomcat)

kubectl logs my-tomcat-server



¡Fin del módulo!

Desde el comienzo de este primer módulo del curso hemos visto muchas cosas acerca de Kubernetes.

- Aprendido a definirlo y situarlo en el stack de tecnologías
- Visto qué nos ofrece y las razones de su éxito
- Hemos puesto en marcha un entorno de prueba en el navegador
- Hemos interactuado con kubectl y desplegado algunos pods usando YAML
- Hemos visto en primera persona cómo se ejecuta un contenedor en Kubernetes en el nodo.

Extras

Para terminar, dejo algunos recursos interesantes para conocer un poco más a fondo los temas que hemos tratado en este módulo:

Cómic (en inglés)
Guía ilustrada de Kubernetes (en inglés)
Cloud Native Computing Foundation
Post sobre Borg, el predecesor de Kubernetes (en inglés)
Repositorio de código oficial en GitHub
Reportaje de CNCF sobre la adopción de contenedores y Kubernetes

Extras

Páginas de la documentación oficial:

¿Qué es Kubernetes?
Pods
kubectl

Aprende más sobre YAML:
Introducción al lenguaje YAML

Extras

Distintas opciones para levantar un clúster de Kubernetes en local:

Docker en Windows
Docker en Mac
k3s
MicroK8S (mi recomendación para Linux)
MiniKube (solución oficial)

¡Gracias!