Docker y Kubernetes

ADMINISTRACIÓN DE SISTEMAS OPERATIVOS

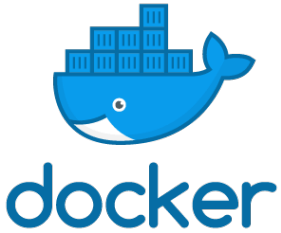
Curso 2020/2021

Cristian Martín Fernández

Contenido

- Docker
- Kubernetes

Docker

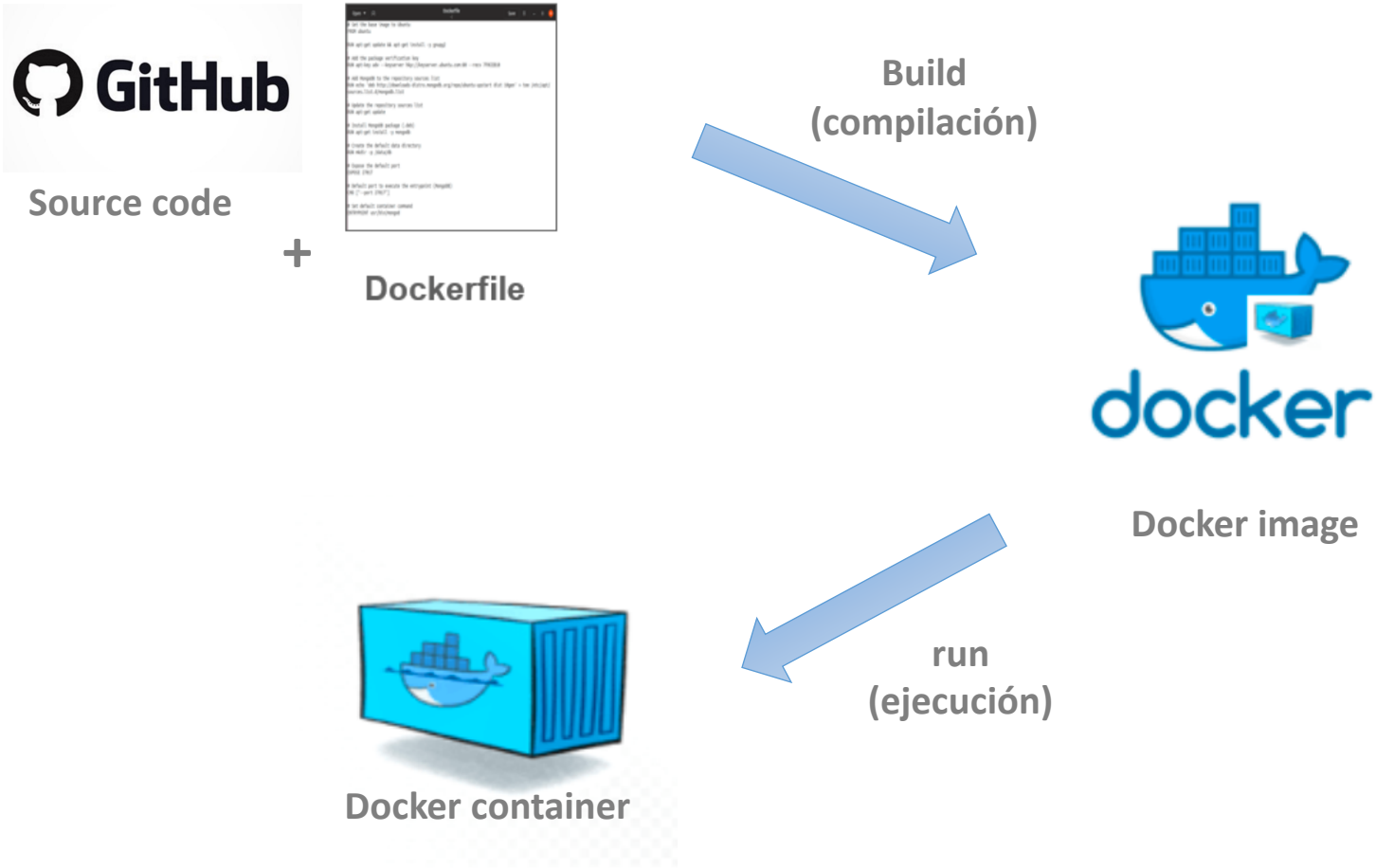


- Docker (www.docker.com) es una tecnología que permite la gestión y despliegue de aplicaciones ligeras basadas en contenedores.

- Los contenedores permiten varios procesos por separado para hacer un mejor uso de su infraestructura y, al mismo tiempo, conservar la seguridad que tendría con sistemas separados.

- Una de las principales ventajas de los contenedores respecto a las tecnologías de virtualización tradiciones (ej., máquinas virtuales) es su rápida puesta en funcionamiento, lo que favorece la instalación y migración de aplicaciones. Además de su ligereza!

Ciclo de una aplicación Docker



Dockerfile

- Un Dockerfile es un documento de texto que contiene todos los comandos para ensamblar una imagen. La imagen debe contener todo lo necesario para ejecutar una aplicación: todas las dependencias, configuración, scripts, binarios, etc.
- Ejemplo de Dockerfile:

FROM tensorflow/tensorflow:2.0.1-py3 **# Partimos como base de una imagen que ya tiene tensorflow y python instalado**

WORKDIR /usr/src/app **# establece el directorio de trabajo**

RUN pip install --upgrade pip **# actualizamos pip**

COPY ./requirements.txt /usr/src/app/requirements.txt **# copiamos el fichero de dependencias**

RUN pip install -r requirements.txt **# instalamos las dependencias**

COPY . /usr/src/app/ **# copiamos el resto del proyecto**

EXPOSE 8000 **# habilitamos el puerto 8000 para que esté disponible el acceso al contenedor**

RUN chmod +x ./start.sh **# damos permiso de ejecución al fichero principal del proyecto**

CMD ["./start.sh"] **# comando principal que se ejecutará cuando se ejecute el contenedor**

Instalación Docker

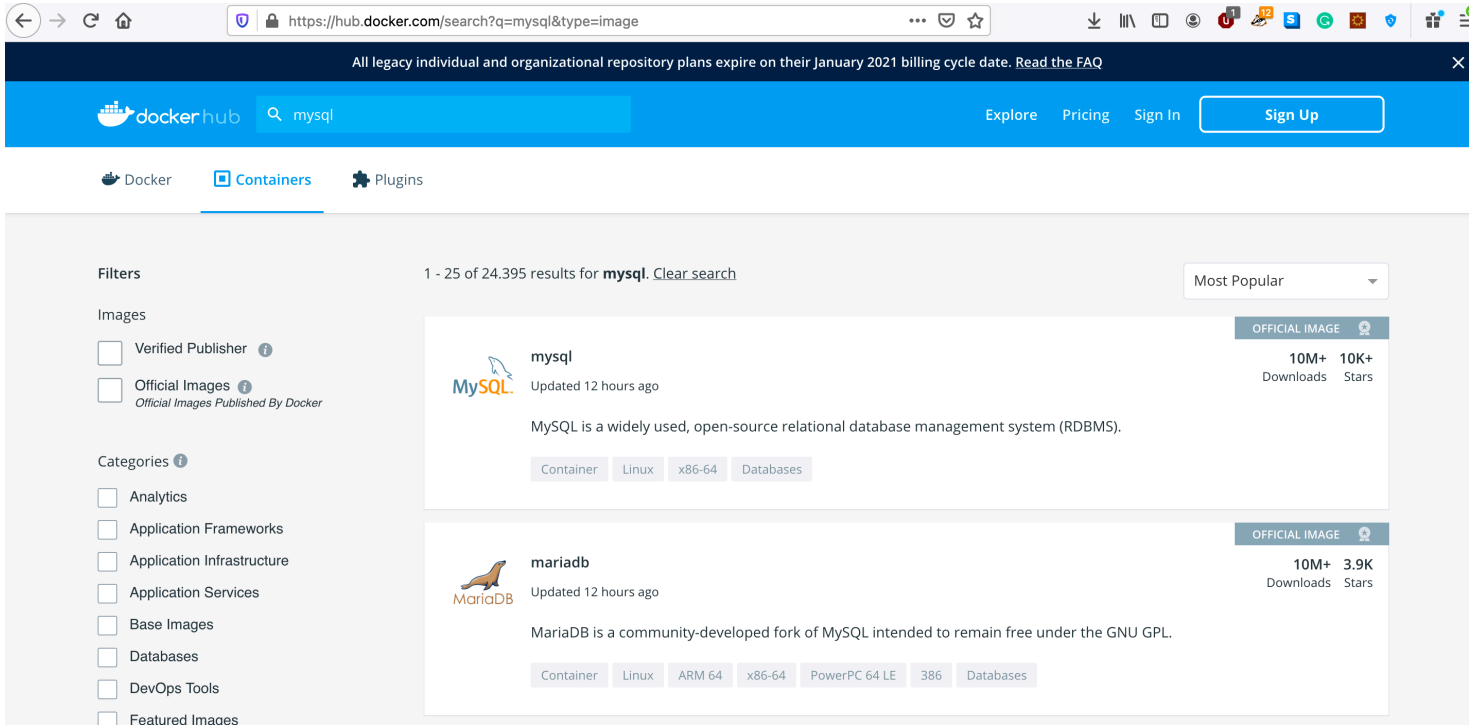
- Actualizamos los repositorios:
  - $ sudo apt-get update
- Instalamos los paquetes para que APT pueda instalar paquetes sobre HTTPS:
  - $ sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
- Añadimos la GPG de Docker:
  - $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
- Mostramos la clave:
  - $ sudo apt-key fingerprint 0EBFCD88
- Añadimos el repositorio:
  - $ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

Instalación Docker en Ubuntu Server

- Instalamos Docker:
  - $ sudo apt-get install docker-ce docker-ce-cli containerd.io
- Para ejecutar sin sudo docker:
  - $ sudo groupadd docker
  - $ sudo usermod -aG docker $USER
  - $ newgrp docker
- Para probar que está funcionado, ejecutamos el contenedor de comprobación de Docker:
  - $ docker run hello-world

Docker Hub

- El sitio donde buscar imágenes de contenedores Docker existentes.
- Las imágenes están organizadas por etiquetas indicando su versión y el SO sobre el cual pueden desplegarse.



Ejecución de un contenedor

- Ejecutamos el contenedor hello-flask encontrado en DockerHub:
  - $ docker run -d -p 8080:5000 markbenschop/flask-demo:1.5
- La opción –d sirve para ejecutar el contenedor en background. La opción –p para el reenvío de puertos (8000 al exterior y 5000 al contenedor).
- Una vez ejecutándose, podemos ver el log del componente al partir del ID que nos devuelve cuando se ejecuta:
  - $ docker logs <CONTAINER_ID>
- Y hacer una petición a la página web desplegada:
  - $ curl localhost:8080

Comandos útiles Docker

- Para ver el listado de contenedores en ejecución:
  - $ docker ps
- A partir del ID de un contenedor podemos pararlo:
  - $ docker stop <CONTAINER_ID>
- Y borrarlo:
  - $ docker rm <CONTAINER_ID>
- Para ver el listado de imágenes
  - $ docker image ls
- Y para borrar una imagen:
  - $ docker image rm <IMAGE_ID>

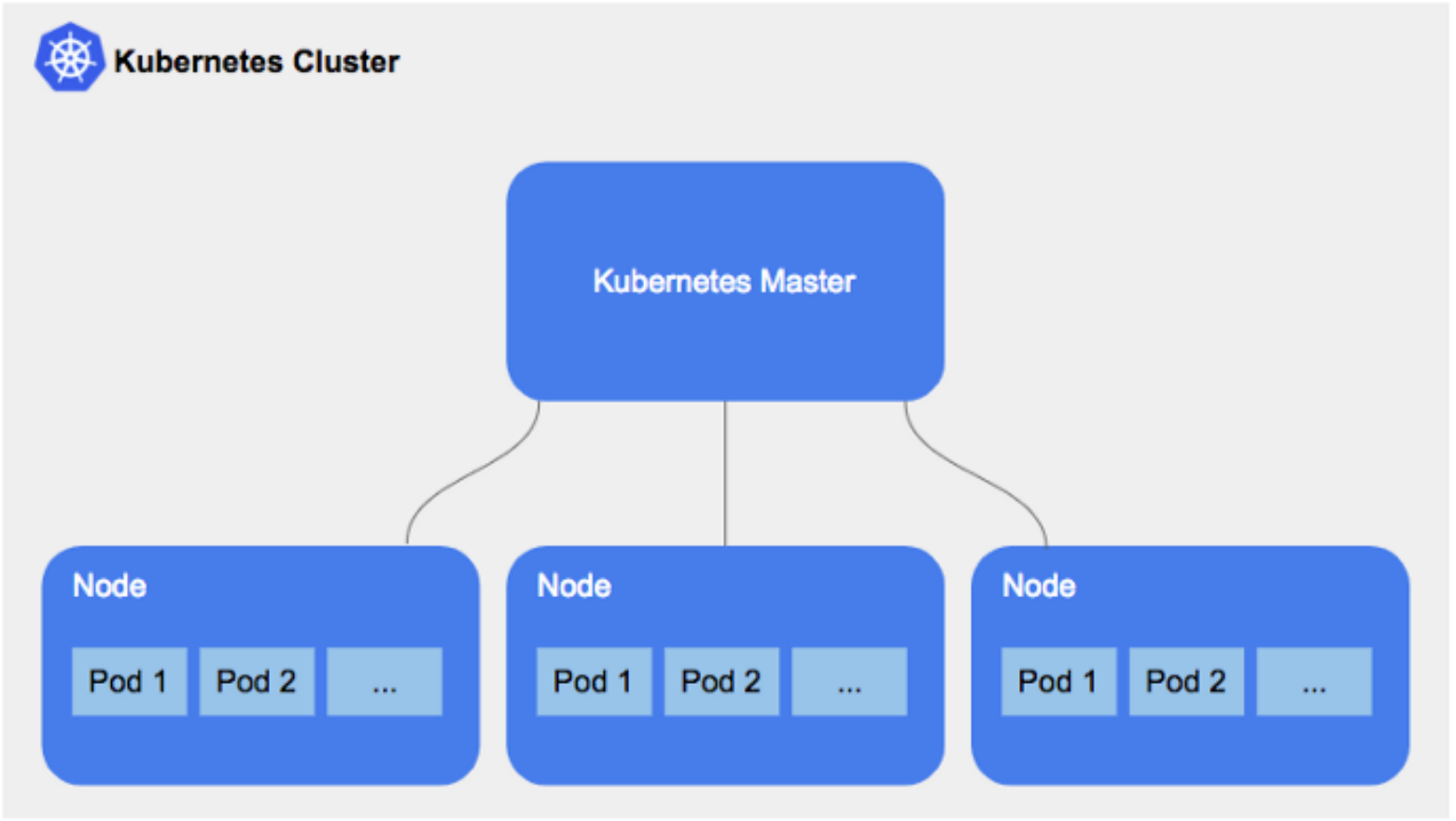
Kubernetes

- Con Docker podemos gestionar aplicaciones ligeras en una sola máquina.

- Cuando queremos funcionalidades típicas de sistemas empresariales como replicas de contenedores (balanceo de carga), tolerancia a fallos, alta disponibilidad, etc., necesitamos un clúster de nodos o servidores que puedan albergar nuestros contenedores y ofrecer el servicio deseado.

- Kubernetes (https://kubernetes.io/) nos sirve para justamente eso: para automatizar la implementación, el escalado y la administración de aplicaciones basadas en contenedores en un clúster de nodos (que también puede ser una única máquina).

Kubernetes


Kubernetes Cluster
Kubernetes Master
Node
Pod 1
Pod 2
...
Node
Pod 1
Pod 2
...
Node
Pod 1
Pod 2
...

Conceptos Kubernetes

- Todos los componentes en Kubernetes se despliegan en ficheros yaml.
- Entre los componentes destacan:
  - **Pods**: unidades de computación desplegables más pequeñas que se pueden crear y gestionar en Kubernetes.
  - **Deployments**: permiten controlar continuamente el estado de un conjunto de Pods para satisfacer características más avanzadas como replicación.
  - **Services**: exponen una aplicación que se ejecuta en un conjunto de Pods como un servicio de red (ofreciendo un solo nombre de DNS para el conjunto de pods, y permitiendo el balanceo de carga entre ellos).

Instalación Kubernetes en Ubuntu Server

- $ sudo apt-get update && sudo apt-get install -y apt-transport-https gnupg2 curl
- $ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
- $ echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list
- $ sudo apt-get update
- $ sudo apt-get install -y kubectl

Creación de cluster

- Se puede crear un clúster de Kubernetes de múltiples formas:

1. Instalando docker-desktop. Opción recomendada para clúster locales (Mac y Windows)
2. Instalando un clúster local con minikube (2 CPUs, 2GB de RAM, 20GB de espacio de disco):
   - curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64
   - deb sudo dpkg -i minikube_latest_amd64.deb
   - minikube start
3. Instalando kubeadm y creando un clúster. Opción recomendada para clústeres en producción.

Ejemplo Kubernetes

- Descargar el proyecto de ejemplo:
  - $ git clone https://github.com/JasonHaley/hello-python.git
- Nos vamos a la carpeta hello-python/app y modificamos el fichero main.py para que muestre la IP que responde:

```
import socket

@app.route("/")
def hello():
    hostname = socket.gethostname()
    local_ip = socket.gethostbyname(hostname)
    return "Hello from: "+str(local_ip)


if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Ejemplo Kubernetes configuration

```
apiVersion: v1
kind: Service
metadata:
name: hello-python-service
spec:
selector:
name: pod-hello-python
ports:
- protocol: "TCP"
port: 8000
targetPort: 5000
type: LoadBalancer

ports:
- containerPort: 5000
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
labels:
name: hello-python
name: hello-python
spec:
replicas: 3
selector:
matchLabels:
name: pod-hello-python
template:
metadata:
labels:
name: pod-hello-python
name: hello-python
spec:
containers:
- name: hello-python
image: hello-python:latest
imagePullPolicy: Never
ports:
- containerPort: 5000
```

Ejemplo Kubernetes

- Compilamos el contenedor para crear la imagen:
  - $ cd hello-python/docker
  - $ docker build -f Dockerfile -t hello-python:latest .
- Ahora podemos desplegar el deployment del contenedor creado:
  - kubectl apply -f kubernetes/deployment.yaml

Comandos útiles Kubernetes

- Para ver el listado de componentes en ejecución:
  - $ kubectl get all
- Para ver el log de un componente utilizamos
  - $ kubectl logs -f <COMPONENT_ID>
- Y para borrar un componente:
  - $ kubectl delete <COMPONENT_ID>

Bibliografía

- Docker: https://docs.docker.com/engine/reference/
- Docker: https://www.redhat.com/es/topics/containers/what-is-docker
- Kubernetes: https://kubernetes.io/es/docs/
- Minikube: https://minikube.sigs.k8s.io/docs/start/
- Kubernetes using Python: https://kubernetes.io/blog/2019/07/23/get-started-with-kubernetes-using-python/