

# Bloque 4 - Desplegando contenedores

Profesor: Iván Garrido Tamarit  
Primera sesión: 2022-10-28  
Segunda sesión: 2022-10-29

## Agenda

- Sesión 1:
  - ¿Qué es Docker?
  - Dockerizando una aplicación.
  - Desplegando mi aplicación con contenedores – Primeros pasos.
  - ¿Cómo trabajamos en Mercadona Tech con contenedores?
- Sesión 2:
  - Estrategias de despliegue con contenedores.
  - Flujo de trabajo con contenedores.
  - ¿Cómo es el flujo de desarrollo en Mercadona Tech?

## Requerimientos

Principalmente vamos a usar **docker**. Si tu sistema operativo es Linux o macOS no tendrás grandes problemas ya que funciona *algo* mejor que en Windows.

Para instalarlo podemos seguir las guías oficiales:

- Windows: <https://docs.docker.com/docker-for-windows/>
- macOS: <https://docs.docker.com/docker-for-mac/install/>
- Linux (ubuntu): <https://docs.docker.com/engine/install/ubuntu/>

También vamos a utilizar Kubernetes dentro de Docker, así que necesitamos tener instalado:

- Golang: <https://go.dev/doc/install>
- Kind (Kubernetes In Docker): <https://kind.sigs.k8s.io/docs/user/quick-start/#installation>

En la segunda sesión tocaremos algo de Kubernetes y para que sea más fácil lo haremos directamente sobre docker. Si no sabes o no lo consigues hacer funcionar, no te preocupes. En la sesión podemos verlo.

Además es muy recomendable tener a mano un buen editor de código con el que nos sintamos cómodos. Personalmente recomiendo **vscode** de Microsoft. Es multiplataforma y tiene infinidad de plugins in configuraciones.

- vscode: <https://code.visualstudio.com/>

Hay ciertos plugins para vscode que nos pueden ayudar pero no son para nada obligatorios:

- Docker plugin: <https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker>
- YAML support: <https://marketplace.visualstudio.com/items?itemName=redhat.vscode-yaml>
- Gremlins tracker, plugin para ver código y carácteres que nos pueden dar problemas:  
<https://marketplace.visualstudio.com/items?itemName=nhoizey.gremlins>

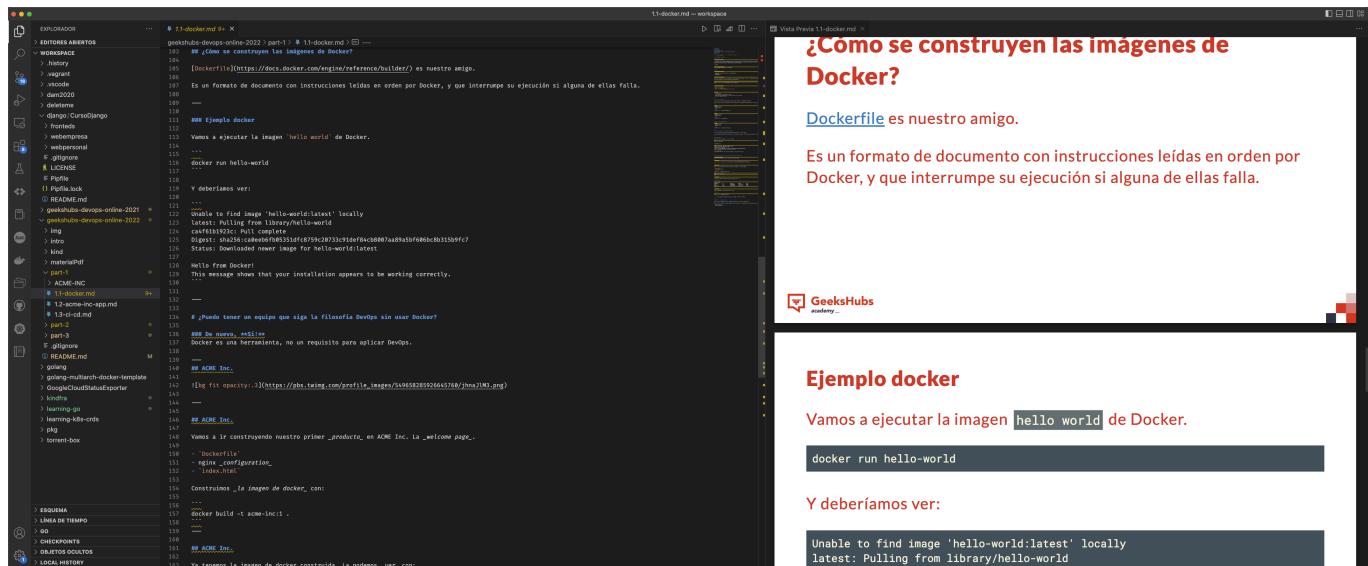
# Material de clase

---

Todo el material del curso lo encontrareis en este repositorio. Las transparencias estan hechas con [Markdown](#). Y usando el motor de renderizado de [Marp](#). Existe también plugin para vscode por lo que si quereis podeis ver las transparencias en la misma ventana que el código y la terminal.

Si teneis curiosidad por la sintaxis de Marp, la tenéis [aquí](#).

Si decidís usar vscode con el plugin de [marp](#) podeis llegar a ver todo el material en una sola ventana.



También encontraréis las transparencias en formato PDF en el directorio [materialPdf](#).

Geekhubs - DevOps Bootcamp 2022 - Iván Garrido

# 4 Desplegando Contenedores

# Hola a todos!

## Mis pasitos en el mundo de IT

- Administrador de redes en Consum Cooperativa
- Investigador en Universitat Politècnica de València
- Administrador de sistemas en Onestic
- Líder del equipo SRE y mánager de ingeniería en Mercadona Tech



# Dinámica de clase

- Presentación del concepto/herramienta
- Debate abierto
- Ejercicio práctico



# 4 Desplegando Contenedores

# ¿Por qué contenedores?

“ Containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run. This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is a private data center, the public cloud, or even a developer’s personal laptop. ”

-- Google



# ¿Por qué contenedores?

Pero, ¿se puede hacer DevOps sin contenedores?

Sí!



# ¿Por qué contenedores?

- Encapsulado de la aplicación junto a sus dependencias.
- Seguridad.
- Inmutabilidad.



# ¿Por qué contenedores?

Ahora, con la irrupción de los contenedores es más sencillo que nunca crear un **pipeline** de desarrollo que permita prácticas más modernas como las que normalmente se utilizan en cultura de equipos DevOps.

```
Code -> Build -> Test -> Deploy Staging -> Deploy Production
```



# ¿Por qué contenedores?

```
docker run -ti python
```

```
docker run -p 8080:80 nginx
```

```
docker run -e POSTGRES_PASSWORD=password postgres
```

...



# Docker

- ¿Qué es Docker?
- ¿Por qué Docker es tan popular hoy en día?
- ¿Puedo tener un equipo que siga la filosofía DevOps sin usar Docker?
- ¿Qué es una imagen de Docker?

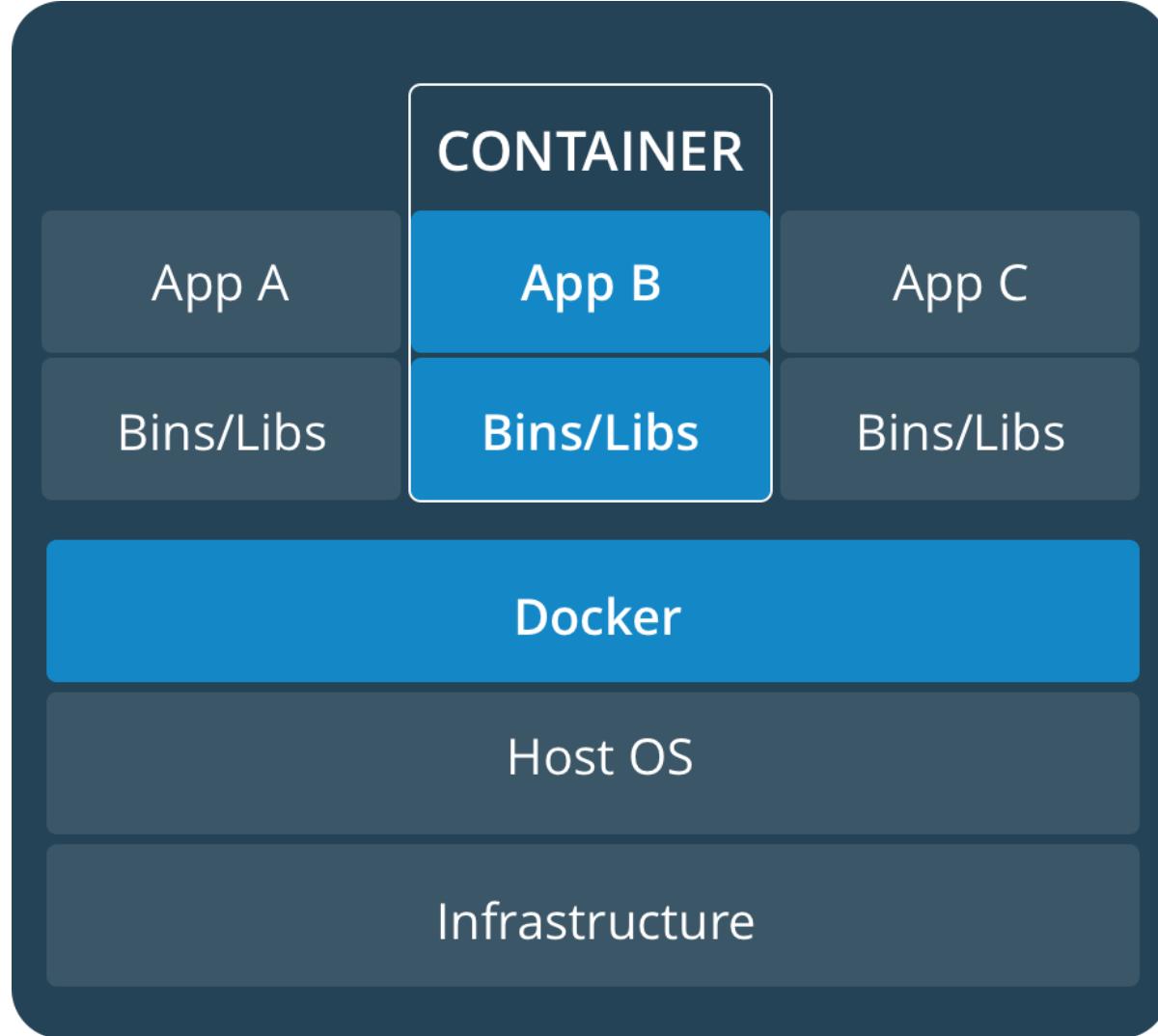


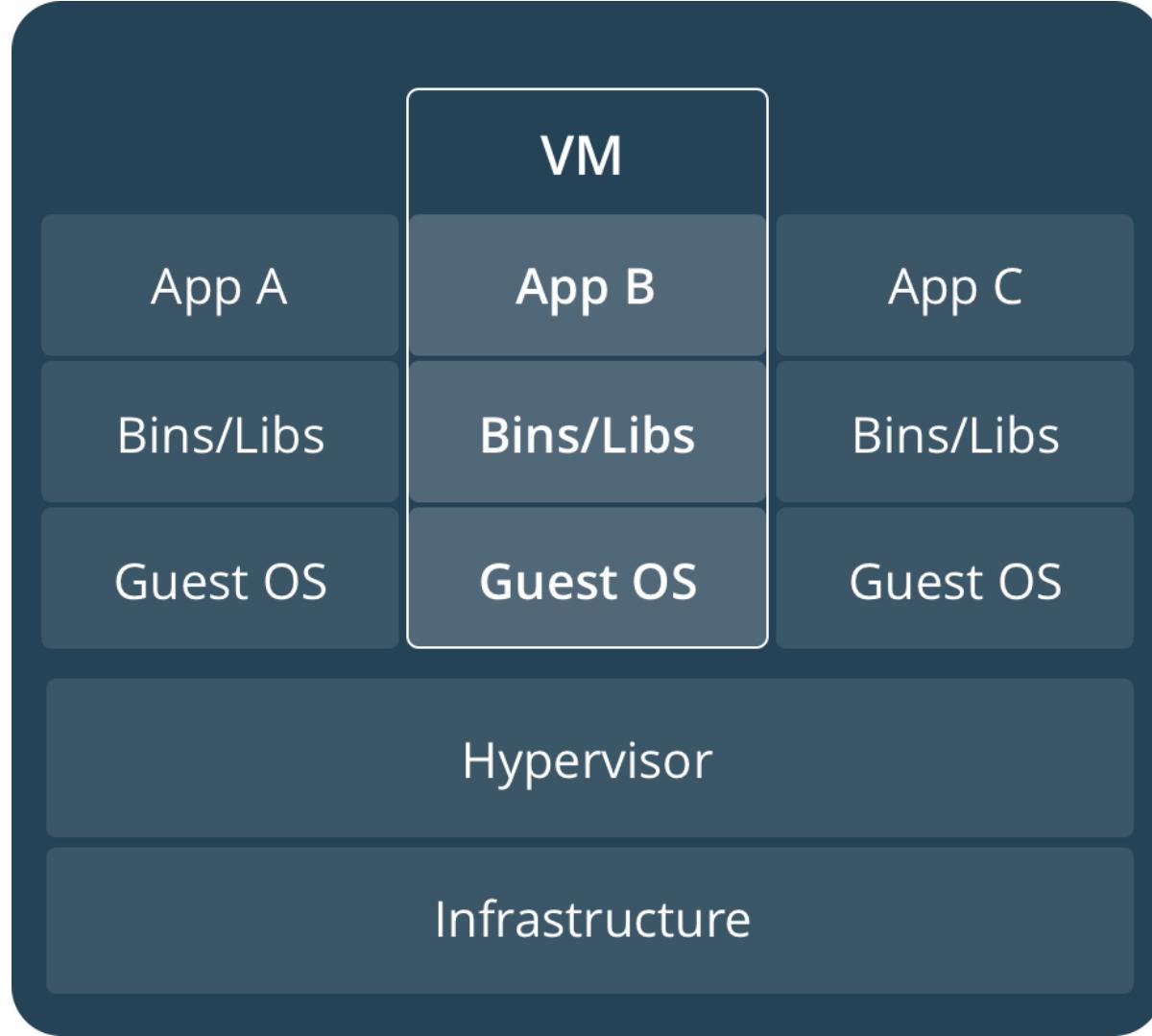
# Pero.. ¿Qué es Docker?

Es el producto comercial multiplataforma que nos permite trabajar con contenedores fácilmente

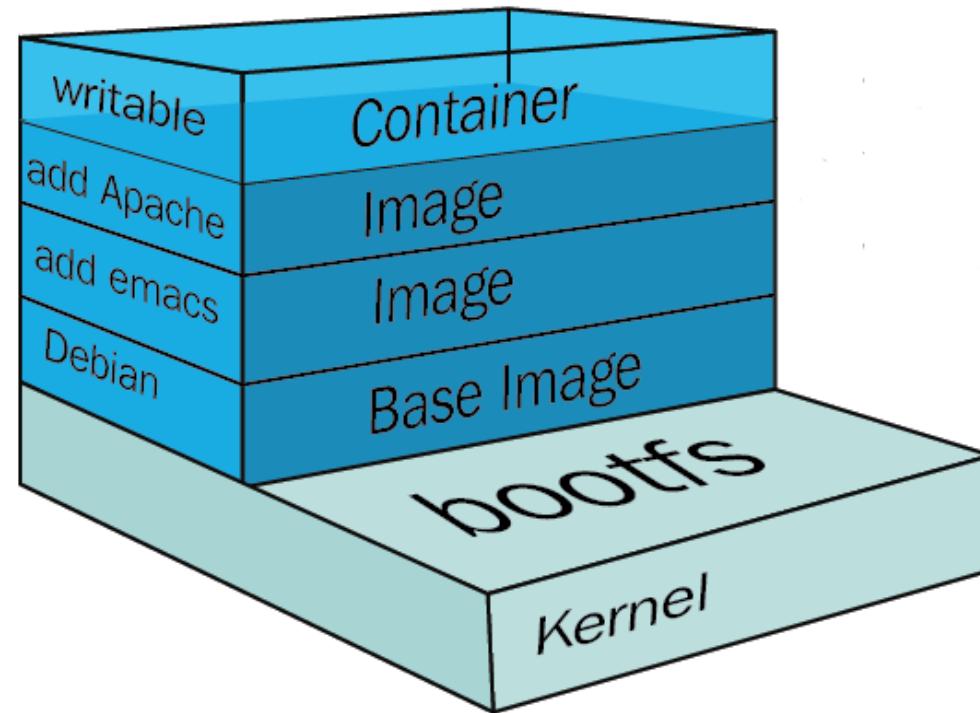
Docker se creó en 2013 por Salomon Hykes. Hoy en día parece que *o despliegas contenedores docker o no eres nadie* :D







# ¿Qué es una imagen de Docker?



# ¿Cómo se construyen las imágenes de Docker?

Dockerfile es nuestro amigo.

Es un formato de documento con instrucciones leídas en orden por Docker, y que interrumpe su ejecución si alguna de ellas falla.



# Ejemplo docker

Vamos a ejecutar la imagen `hello world` de Docker.

```
docker run hello-world
```

Y deberíamos ver:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
Digest: sha256:ca0eeb6fb05351dfc8759c20733c91def84cb8007aa89a5bf606bc8b315b9fc7
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker!

This message shows that your installation appears to be working correctly.



# ¿Puedo tener un equipo que siga la filosofía DevOps sin usar Docker?

De nuevo, Sí!

Docker es una herramienta, no un requisito para aplicar DevOps.



# ACME Inc.



# ACME Inc.

Vamos a ir construyendo nuestro primer *producto* en ACME Inc. La *welcome page*.

- Dockerfile
- nginx configuration
- index.html

Construimos *la imagen de docker* con:

```
docker build -t acme-inc:1 .
```



# ACME Inc.

Ya tenemos la imagen de docker construida. La podemos ver con:

```
docker images
```

\$ docker images	REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
	acme-inc	1	8dcaea4ab748	6 minutes ago	127MB
	nginx	latest	9beeba249f3e	8 days ago	127MB
	hello-world	latest	bf756fb1ae65	4 months ago	13.3kB



# ACME Inc.

Vamos a *desplegar* esta imagen en nuestro *local*.

- docker run

```
docker run --rm -p 8080:80 acme-inc:1
```

- **-p** indica que puertos queremos *mapear* de la imagen a nuestro *localhost*.



# ACME Inc. - Ejercicio práctico

Hemos visto como podemos construir una imagen simple de nginx con nuestra configuración.

Vamos a llevar a ACME Inc. un pasito adelante y vamos a modificar la *welcome page* a nuestro gusto con los siguientes requisitos:

- Página principal
- Página *sobre Acme Inc.*



# 4 Desplegando Contenedores

# ACME Inc.

Vamos a crecer la empresa y lanzar nuestra nueva *web app*. Para ello el equipo de desarrollo está trabajando en una aplicación escrita en *python*.



# ACME Inc. - Dockerizando la aplicación

Hasta ahora la aplicación se lanza con el comando:

```
python3 app.py
```

Pero antes has de asegurarte de tener el interprete correcto y los paquetes de dependencias necesarios.

```
pip install -r requirements.txt
```



# ACME Inc.

**Ejercicio:** Crear el **Dockerfile** de nuestra app. (*Si te atascas tienes uno creado en la carpeta*).

```
FROM python:3-alpine  
[ . . ]
```



# ACME Inc. - Despliegue de la *web app*

Para desplegar nuestra aplicación *dockerizada* manualmente sería:

- ssh a nuestro servidor
- git clone OR git pull desde nuestro servidor
- docker build -t acme-inc-app:miversion .
- docker run -d -p 80:8080 acme-inc-app:miversion

¿Alguna pega a esta forma de desplegar?



# ACME Inc. - Docker-compose entra en juego

Docker-compose es una herramienta que nos permite ejecutar múltiples contenedores de Docker al mismo tiempo.

Nos permite *darle forma y proporcionar conectividad* a nuestro conjunto de contenedores mediante ficheros YAML.



# ACME Inc. - Primer paso a un despliegue más automático

`docker-compose` + *ansible OR terraform*

- Terraform para la VM
- Ansible para *desplegar el* `docker-compose`

Ejercicio: Vamos a crear el `docker-compose.yml` de nuestra *web app.*  
*(Si te bloqueas, encontrarás uno en la carpeta)*



# ACME Inc.

Vamos a *construir* y *levantar* nuestra aplicación.

```
docker-compose -f docker-compose.yml up --build
```



# ACME Inc.

Vuestro equipo de desarrollo quiere dar el siguiente paso. No quiere hacer `build` de la *imagen de docker* en el servidor remoto si no que quiere construirla *1 vez* y *distribuirla* (arquitectura inmutable).

Pero, ¿cómo?

- Docker Hub (public cloud)
- Container Registry (on prem)



# ACME Inc.

Podemos hacer `push` de la imagen a un *registry* y después hacer `pull` en cada uno de los servidores a los cuales queramos descargar la imagen.

Ejemplo: Vamos a *levantar* nuestro `container registry` usando `docker`



# ACME Inc.

```
docker run -d -p 5000:5000 --restart always --name registry registry:2
```

Comprobamos que se la *levantado* correctamente:

\$ docker ps -a	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
	d1e7dc1fc7ab	registry:2	"/entrypoint.sh /etc..."	3 seconds ago	Up 3 seconds	0.0.0.0:5000->5000/tcp	registry



# ACME Inc.

Vamos a *subir* al registry nuestra web app

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
acme-inc-app	1	d85a2cfb5c30	51 minutes ago	94.5MB

Vamos a *retaguear* nuestra imagen

```
docker tag acme-inc-app:1 localhost:5000/acme-inc-app:1
```



# ACME Inc.

Y ahora hacemos `push`

```
docker push localhost:5000/acme-inc-app:1
The push refers to repository [localhost:5000/acme-inc-app]
c2d61e88e95b: Pushed
50aafb14c23d: Pushed
a7d85ab87af8: Pushed
7a82840eb682: Pushed
df897d49f46c: Pushed
a539b76feca4: Pushed
3e207b409db3: Pushed
1: digest: sha256:fb62775e4b57e3f31bd8ec717de107ae83a7fcce23e93de6089baf9856ab9052 size: 1789
```



# ACME Inc.

Ahora ya tenemos nuestra imagen *subida* al registry y podemos descargarla en cualquiera de nuestros servidores con:

```
docker pull localhost:5000/acme-inc-app:1
```

Ejercicio: Modificar el `docker-compose` para que no haga `build`



# ACME Inc.

```
app:  
  image: localhost:5000/acme-inc-app:1  
  stdin_open: true  
  tty: true  
  volumes:  
    - ./src:/app  
  ports:  
    - 8080:8080  
  environment:  
    - ENVIRONMENT=local  
    - DEBUG=True  
    - RELOADER=True  
    - APM_NAME=bootcamp
```



# ACME Inc.

## La prueba del algodón

- Borramos la imagen `docker rmi localhost:5000/acme-inc-app:1`
- Lanzamos docker-compose
  - `docker-compose -f docker-compose.yml up`

```
Pulling app (localhost:5000/acme-inc-app:1)...
1: Pulling from acme-inc-app
Digest: sha256:fb62775e4b57e3f31bd8ec717de107ae83a7fcce23e93de6089baf9856ab9052
Status: Downloaded newer image for localhost:5000/acme-inc-app:1
...
```

# ACME Inc.

Recapitulemos:

- Hemos creado nuestra primera imagen de docker .
- Hemos dockerizado nuestra simple web app.
- Hemos desplegado a mano.
- Hemos mejorado el despliegue con un registry y docker-compose .



# ACME Inc.

...Siguientes pasos?



# 4 Desplegando Contenedores

# ACME Inc.

El equipo ha visto que puede construir una imagen y distribuirla pero sigue siendo un proceso manual (*propenso al fallo*). Ha llegado el momento de *automatizar* el proceso.

Vamos a usar:

- Servidor git
- Jenkins
- Private docker registry



# ACME Inc.

Vamos a usar `docker` para lanzar *localmente* un *stack* completo.

Lanzaremos `Jenkins`, `Gogs` & `docker registry`.

- <https://www.jenkins.io/>
- <https://gogs.io/>
- <https://hub.docker.com/registry>



# ACME Inc.

Dentro de la carpeta servicios lanzaremos el docker-compose en una terminal nueva.

```
docker-compose -f docker-compose.yml up --build
```



# ACME Inc.

Live setup de las nuevas herramientas.

-> Dentro de la carpeta **servicios** teneis una mini guía.



# Jenkinsfile, caliente, que sales

## Jenkinsfile

- ¿Qué es y para qué sirve?



# ACME Inc.

Live coding session.

Vamos a configurar Jenkins para que construya la imagen de docker de nuestra app y la *suba* al registro de imágenes.

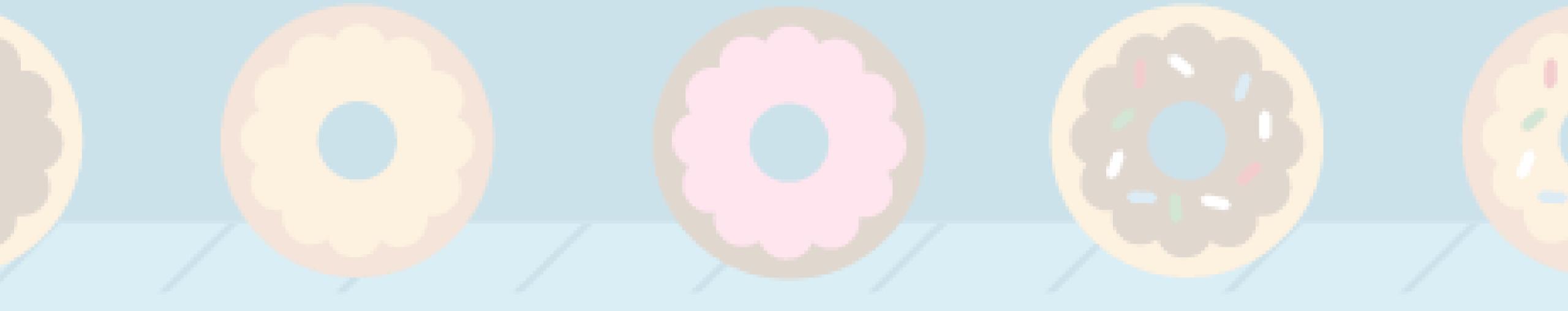




**GeeksHubs**  
academy\_



# Continuous Delivery



# Continuous Delivery Pipeline



# CD

- Manual
- Automático



# CD

Algunas herramientas para automatizar el despliegue desde Jenkins (o nuestro CI/CD):

- Ansible
- Chef
- SaltStack
- Scripts en bash



# CD

“ *Antes de elegir una herramienta hay que tener claro lo que se quiere hacer.* ”

...porque casi todo se puede hacer con todas las herramientas.



# CD

Alguna herramienta específica para la parte de despliegues?

[Spinnaker](#)

[Argo CD](#)



# Estrategias de Despliegue



# 4 Desplegando Contenedores

# Contenido

- Estrategias más comunes de despliegue.
  - Recreate
  - Rolling Update
  - Blue-Green / Red-Black / A/B
  - Shadow
  - Canary



# Cómo

Probaremos y debatiremos acerca de algunas de ellas:

- Manual, docker run
- Manual, docker-compose
- Automática, ansible / chef / puppet / saltstack /etc..
- Automática, kubernetes



# 4 Desplegando Contenedores

# Recreate



GeeksHubs  
academy\_

# Recreate

Reemplazar la actual versión por una nueva.

Apagar y encender.



# Recreate - ¿Cuándo utilizarlo?

- Cuando nuestra aplicación no esté preparada para trabajar con más de una réplica.
- Cuando nuestro despliegue sea manual, e.g. ftp a un directorio.
- En un estado embrionario de nuestro producto nos puede simplificar el despliegue (no producción)



# Recreate

- docker run hello-world

Hello from Docker!

This message shows that your installation appears to be working correctly.

[ ... ]



# Recreate

Construimos el contenedor de nuestra aplicación y le decimos a docker que la ejecute:

```
docker build -t first-app:v1 .
docker run --rm -d -p 8080:80 first-app:v1
```



# Recreate

Podemos comprobar que está desplegada con:

```
curl localhost:8080
```

y como respuesta:

```
<h1>GeeksHubs - DevOps Bootcamp</h1>
<p>Version 1.0</p>
```



# Recreate



**GeeksHubs**  
academy\_



# Recreate

Ahora vamos a por la `v2`.

```
vim index.html
```

Y sustituimos `<p>Version 1.0</p>` por `<p>Version 2.0</p>`

Entonces hacemos `build` de nuestra nueva versión.

```
docker build -t first-app:v2 .
```



# Recreate

Podemos ver las imágenes que tenemos con `docker images`.

Ahora paramos el contenedor con la versión 1.

```
docker stop containerId
```

Y lanzamos la v2:

```
docker run --rm -d -p 8080:80 first-app:v2
```

Y comprobamos de nuevo el output:



**GeeksHubs**

```
curl localhost:8080
```

# Recreate

- Downtime.
- Fácil de implementar.



# Recreate

Ejercicio:

- Construir v3 y desplegarla.



# 4 Desplegando Contenedores

# Rolling Update



**GeeksHubs**  
academy —

# Rolling Update

Reemplazo progresivo de la versión.

Tengo 3 réplicas. Cambio una de las *antiguas* por una de la nueva.  
Luego otra y así hasta tener todas reemplazadas.

```
AAA  
BAA  
BBA  
BBB
```



# Rolling Update

- Si nuestra aplicación está preparada para trabajar con múltiples réplicas.
- Si los cambios introducidos no *rompen* y ambas versiones pueden convivir.



# Rolling Update

- *Sin downtime*
- Cambio incremental, menos riesgo.
- Poder parar el cambio sin afectar al 100% de los usuarios.



# Rolling Update

⚠ Ejecuta el siguiente comando desde la raíz del repositorio ⚠

Creamos un clúster de Kubernetes con Kind

```
./kind/cluster.sh
```

⚠ Cambia al directorio del ejercicio ⚠

```
cd part-2/2.2.2-rolling-update
```



# Rolling Update





**GeeksHubs**  
academy\_





kind

# Kind

- Proyecto de código abierto que nos permite levantar un clúster de Kubernetes completamente funcional dentro de contenedores de Docker.



# Instalación

<https://kind.sigs.k8s.io/docs/user/quick-start/#installation>



# Comandos útiles

- `kubectl` -> Herramienta de control de Kuberntes

Kubectl cheatsheet ->

<https://kubernetes.io/docs/reference/kubectl/cheatsheet/>



# Demo con kind

```
kind create cluster
```



# 4 Desplegando Contenedores

# Rollig Update

Vamos a crear las imágenes de Docker necesarias primero:

- docker build -t localhost:5001/second-lb:v1 -f Dockerfile-second-lb .

- docker build -t localhost:5001/second-app:v1 -f Dockerfile-second-app .

- **Modificamos index.html y:**

- docker build -t localhost:5001/second-app:v2 -f Dockerfile-second-app .



# Rolling Update

Vamos a subir las imágenes de docker al registry local para que kubernetes las "encuentre".

```
docker push localhost:5001/second-1b:v1
docker push localhost:5001/second-app:v1
docker push localhost:5001/second-app:v2
```



# Rolling Update

Es hora de desplegar nuestra aplicación. Primero la v1.

```
kubectl -n default apply -f second-app.yml
```

Y luego exponerla mediante el balanceador:

```
kubectl apply -f second-lb.yml
```



# Rolling Update

```
kubectl get services
```

```
kubectl get pod
```

```
kubectl get deployment
```



# Rolling Update

Ahora vamos a desplegar nuestra versión v2.

- `kubectl edit deploy second-app`

y sustituimos v1 por v2 en el tag de la imagen del contenedor.



# Rolling Update



# Rolling Update

- *Sin downtime*
- Implementación de moderada dificultad.



# Rolling Update

Ejercicio:

- Construir v3 y desplegarla.
- Eliminar los componentes de esta aplicación de Kubernetes



# 4 Desplegando Contenedores

# Blue/Green



# Blue/Green

Consiste en desplegar de forma simultanea 2 versiones distintas y entonces mover el tráfico de Blue a Green.

Una vez pasamos de A a B, si tenemos problemas podemos hacer **roll back** instantáneo.



# Blue/Green

- *Sin downtime*
- Cambio de A a B
- Rollback inmediato

⚠ Cambia al directorio del ejercicio ⚠

```
cd part-2/2.2.3-blue-green
```



# Blue/Green

Vamos a subir las imágenes de docker al registry local para que kubernetes las "encuentre".

- ```
docker build -t localhost:5001/third-lb:v1 -f Dockerfile-third-lb .
```

- ```
docker build -t localhost:5001/third-app:v1 -f Dockerfile-third-app .
```

- **Modificamos `index.html` y:**

- ```
docker build -t localhost:5001/third-app:v2 -f Dockerfile-third-app .
```



# Blue/Green

Vamos a subir las imágenes de docker al registry local para que kubernetes las "encuentre".

```
docker push localhost:5001/third-lb:v1
docker push localhost:5001/third-app:v1
docker push localhost:5001/third-app:v2
```



# Blue/Green

Es hora de desplegar nuestra aplicación. Ahora vamos a desplegar la v1 y la v2 de forma simultanea, pero únicamente la v1 recibe tráfico.

```
kubectl -n default apply -f third-app.yml
```

Y luego exponerla mediante el balanceador:

```
kubectl apply -f third-lb.yml
```



# Blue/Green

Tenemos `v1` recibiendo tráfico. Ahora es momento de cambiar a `v2`.  
Para ello:

- Editamos el `service` para decirle que ahora conecte a los `pods` de `v2`.
- `kubectl edit service third-app`



# Blue/Green

```
selector:  
  app: third-app  
version: v1
```

Se convierte en:

```
selector:  
  app: third-app  
version: v2
```



# Blue/Green



**GeeksHubs**  
academy\_



# Blue/Green

- *Sin downtime*
- Implementación sencilla.
- Requiere un segundo paso en el despliegue (hacer el cambio de versiones).



# Blue/Green

Ejercicio:

- Construir v3 y desplegarla.
- Ir alternando entre v1, 2 y 3



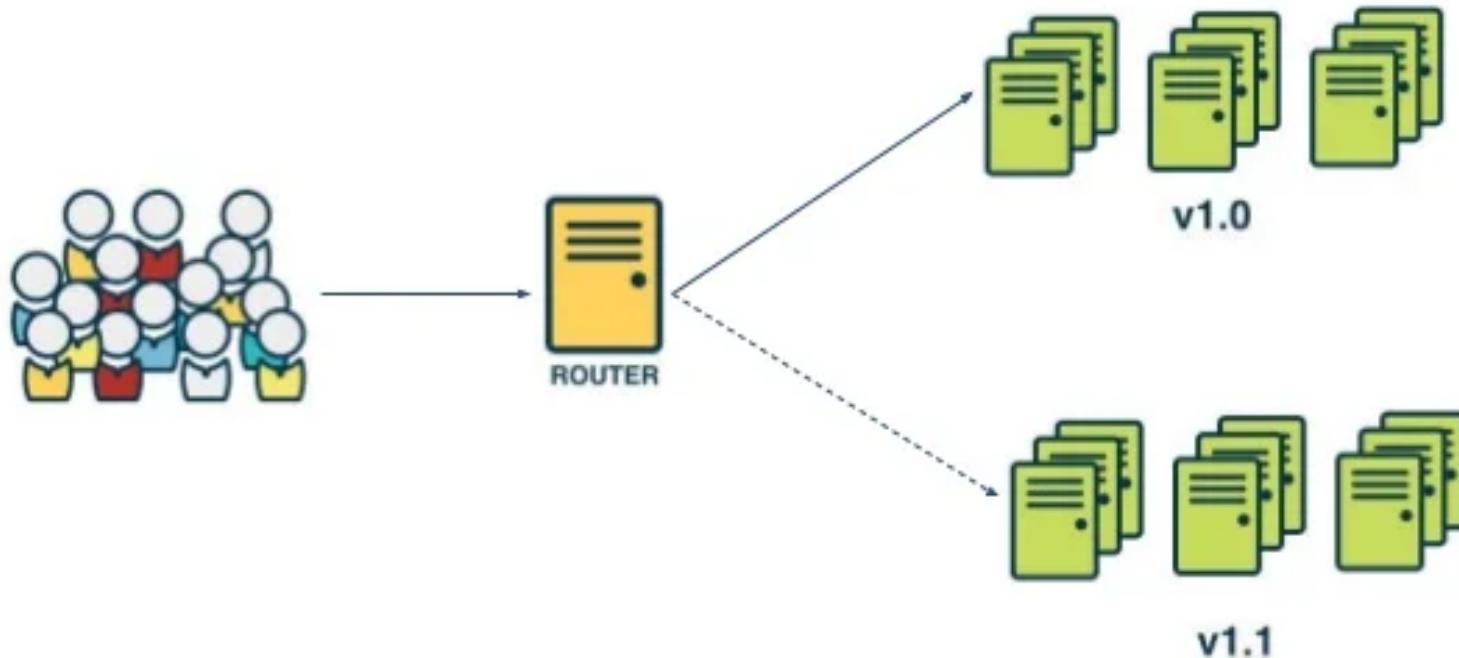
# 4 Desplegando Contenedores

# Shadow



# Shadow

## Traffic Shadowing



# Shadow

- No afecta al tráfico de producción.
- No es exactamente un despliegue.

⚠ Cambia al directorio del ejercicio ⚠

```
cd part-2/2.2.4-shadow/
```



# Shadow

Vamos a crear las imágenes de Docker necesarias primero:

- docker build -t localhost:5001/fourth-lb:v1 -f Dockerfile-fourth-lb .

- docker build -t localhost:5001/fourth-app:v1 -f Dockerfile-fourth-app .

- **Modificamos index.html y:**

- docker build -t localhost:5001/fourth-app:v2 -f Dockerfile-fourth-app .



# Shadow

Vamos a subir las imágenes de docker al registry local para que kubernetes las "encuentre".

```
docker push localhost:5001/fourth-app:v1  
docker push localhost:5001/fourth-app:v2  
docker push localhost:5001/fourth-lb:v1
```



# Shadow

Vamos a desplegar la infraestructura

```
kubectl -n default apply -f fourth-app.yml
```

Y luego exponerla mediante el balanceador:

```
kubectl apply -f fourth-lb.yml
```



# Shadow

Utilizamos `goreplay` como *sidecar* de nginx para hacer *forward* de las requests a `v2`.

```
- name: goreplay
  image: iam404/goreplay:v1.0.0_x64
  args: ["--input-raw", ":80", "--output-http", "fourth-app-v2.default:9000"]
```



# Shadow

Veamos el shadow en acción:

- `kubectl logs -f fourth-app-v2-xxxx`
- `kubectl logs -f fourth-app-v1-xxxx`

y hagamos peticiones:

- `curl -s serviceIP`



# Shadow



**GeeksHubs**  
academy\_



# Shadow

- Para pruebas en las que no queramos comprometer al usuario.
- Implementación algo más elaborada.
- Requiere de software extra (goreplay en nuestro ejemplo).



# Shadow

Ejercicio:

- Construir v3 y hacer shadow de v1 a v2 y v3



# 4 Desplegando Contenedores

# Canary





# Canary

- *Sin downtime*
- Nos permite probar con tráfico real la versión en un % del tráfico.
- Rollback inmediato.

⚠ Cambia al directorio del ejercicio ⚠

```
cd part-2/2.2.5-canary
```



# Canary

Vamos a crear las imágenes de Docker necesarias primero:

- ```
docker build -t localhost:5001/fifth-lb:v1 -f Dockerfile-fifth-lb .
```

- ```
docker build -t localhost:5001/fifth-app:v1 -f Dockerfile-fifth-app .
```

- **Modificamos `index.html` y:**

- ```
docker build -t localhost:5001/fifth-app:v2 -f Dockerfile-fifth-app .
```



# Canary

Vamos a subir las imágenes de docker al registry local para que kubernetes las "encuentre".

```
docker push localhost:5001/fifth-lb:v1
docker push localhost:5001/fifth-app:v1
docker push localhost:5001/fifth-app:v2
```



# Canary

Vamos a desplegar la infraestructura

```
kubectl -n default apply -f fifth-app.yml
```

Y luego exponerla mediante el balanceador:

```
kubectl apply -f fifth-lb.yml
```



# Canary

Tenemos ahora 3 réplicas de v1 y 1 réplica de v2. Esto significa que:

- 75% del tráfico va a recibir v1
- 25% del tráfico va a recibir v2



# Canary

Para ver los pods que están en un **label**.

```
kubectl get pod -l app=fifth-app
```

```
$ kubectl get pod -l app=fifth-app
NAME                  READY   STATUS    RESTARTS   AGE
fifth-app-v1-6d9dff5c6-259dp   1/1     Running   0          3m1s
fifth-app-v1-6d9dff5c6-btx74   1/1     Running   0          3m1s
fifth-app-v1-6d9dff5c6-njndx   1/1     Running   0          3m1s
fifth-app-v2-6c45c99f45-f2w2k   1/1     Running   0          3m1s
```



# Canary

- `kubectl get service fifth-app`
- `watch -d -n 1 'curl -s http://localhost/'`



# Canary

Ahora podemos jugar con los % de cada una de las versiones.

- `kubectl scale deployment fifth-app-v2 --replicas=2`
- `kubectl scale deployment fifth-app-v1 --replicas=2`

Con estos cambios lo balanceamos 50/50



# Canary

Hasta finalmente:

- `kubectl scale deployment fifth-app-v2 --replicas=4`
- `kubectl scale deployment fifth-app-v1 --replicas=0`



# Canary



**GeeksHubs**  
academy\_



# Canary

- *Sin downtime*
- Implementación más laboriosa.
- Requiere madurez en el equipo para hacerlo de forma eficiente y teniendo en cuenta feedback sobre la versión nueva.
  - Ratio de errores
  - Rendimiento
  - Latencias
- Riesgo bajo y controlado frente a fallos.



# Canary

Ejercicio:

- Construir v3 y desplegarla.
- Transicionar desde v1 a v2 y de v2 a v3 sin downtime.
- Hacer *downgrade* de la versión v3 a la v2 sin downtime.



# Construir un Devops pipeline

## Levantar un nuevo clúster de Kubernetes (con Kind)

⚠️ Lanza el comando desde la raíz del repositorio ⚠️

```
./kind/cluster.sh
```

### Requisitos del ejercicio

Vamos a imitar un entorno de producción con múltiples dominios para las aplicaciones del stack. Si tienes gestor de DNS en tu red privada, ¡genial! aprovechalo. Si no, puedes añadir los siguientes dominios en el fichero `/etc/hosts`:

```
127.0.0.1 www.acme.com gogs.acme.com jenkins.acme.com
```

⚠️ Cambia al directorio del ejercicio ⚠️

```
cd part-3/
```

1. `kubectl apply -f manifests/ingress.yml`
2. `kubectl apply -f manifests/gogs.yml`
3. `kubectl apply -f manifests/jenkins-sa.yml`
4. `kubectl apply -f manifests/jenkins.yml`

## Configurar GOGS

URL -> <http://gogs.acme.com/>

Seguir el wizard y crear un repositorio.

- Seleccionar SQLite con ruta absoluta
- Configurar la URL de la interfaz web igual al dominio <http://gogs.acme.com/>

## Configurar Jenkins

URL -> <http://jenkins.acme.com/>

1. `kubectl get pods`
2. `kubectl logs -f jenkins-xxxxx` Extraer el token

Seguir el wizard.

Plugins adicionales:

- Kubernetes

- Blue Ocean (opcional)
- Web for Blue Ocean (opcional)

## Configurar Kubernetes

Jenkins > Configuration

Add cloud:

- Kubernetes Namespace: default
- Jenkins URL: http://jenkins.default:8000
- Crear credenciales de tipo Kubernetes

## Crear job

Crear nuevo **Multibranch pipeline** apuntando al repositorio de Gogs

## Un poco de limpieza

¡Hemos llegado al final! Y por el camino hemos construido y utilizado un montón de herramientas y contenedores. Para eliminarlos, ejecuta los siguientes comandos:

```
kind delete cluster --name produccion  
docker rm -f kind-registry
```

⚠️ Estos comandos son un poco más "delicados" ya que eliminarán todas las imágenes, redes y volúmenes de docker ⚠️

```
docker system prune -fa  
docker volume prune
```