UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Puesta en producción segura

- Tema 3. Implantación de sistemas seguros de despliegue de software

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Índice

- Objetivos
- Virtualización y contenedores (Docker)
- Orquestación de contenedores (Kubernetes)
- DevOps CI/CD (Jenkins)
  - Sistemas de control de versiones (Git y GitHub)
  - Sistemas de automatización de construcción (build)
  - Herramientas de simulación de fallos

Objetivos

- Conocer las principales diferencias entre máquinas virtuales y contenedores.
- Conocer la arquitectura y componentes de un contenedor (Docker).
- Comprender los principales usos de los contenedores en el ciclo de desarrollo software.
- Conocer qué es un orquestador de contenedores y sus principales funciones (Kubernetes).
- Conocer las distintas fases del ciclo de vida de DevOps y cómo se aplica ésta filosofía en el ciclo de desarrollo software.
- Conocer qué son los sistemas de integración continua y cómo se relacionan con los sistemas de control de versiones y de automatización de construcción.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 3. Implantación de sistemas seguros de despliegue de software

- Virtualización y contenedores

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Virtualización y contenedores

Arquitectura máquinas virtuales

- Creación o despliegue de algún tipo de recurso tecnológico a través de software (SO, almacenamiento, aplicaciones).
- Son capaces de ejecutar software como si se tratase de una máquina física real.
- Una máquina virtual o "**guest**" cuenta con un SO completo que se ejecuta de manera aislada al "**host**".
  - Los procesos del "guest" no tienen relación con los del "host" y están limitados en cuanto a recursos.
- Su uso proporciona ciertas ventajas: portabilidad, escalabilidad, optimización del uso de los recursos físicos, etc.

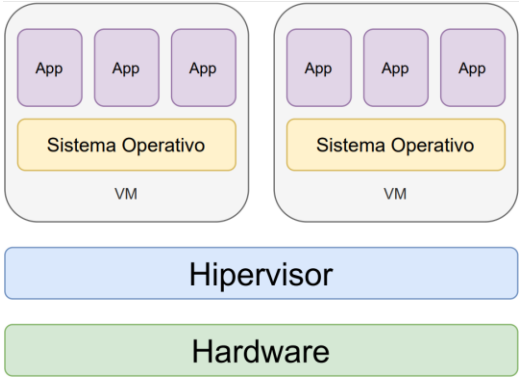
Virtualización y contenedores

Arquitectura máquinas virtuales

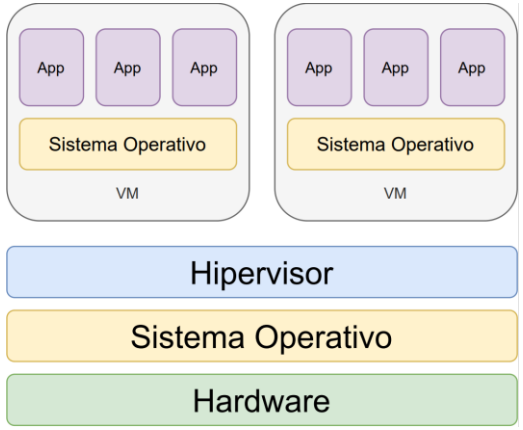
- Un **hipervisor** es un software específico encargado de gestionar máquinas virtuales dentro de una máquina real.

- Permite la abstracción del hardware físico y la asignación de recursos a cada máquina virtual.

- En general, existen dos tipos de hipervisores:
  - **Tipo 1 (bare metal)**: se ejecutan directamente sobre el hardware del sistema → KVM, Hyper-V, ESXi…
  - **Tipo 2 (hosted)**: se ejecutan sobre un SO "host" que a su vez se ejecuta sobre el hardware → VirtualBox, QEMU, VMware Workstation…



Hipervisor tipo 1

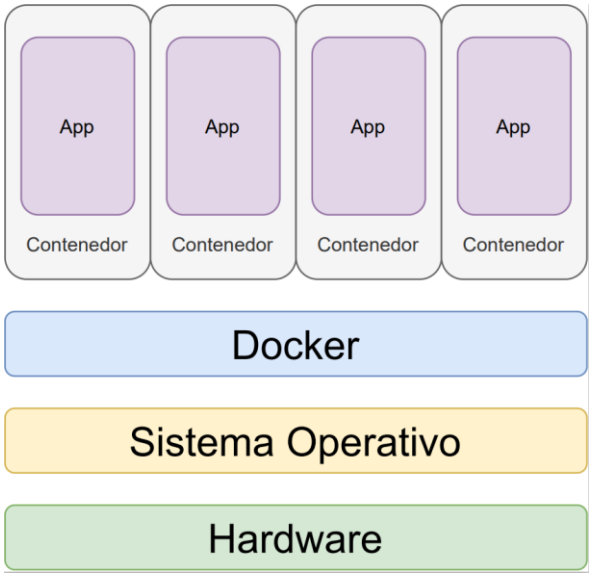


Hipervisor Tipo 2

Virtualización y contenedores

Arquitectura contenedores

- Proporcionan un entorno de ejecución ligero para la ejecución de aplicaciones → Docker, LXD, CRI-O…
- Sólo almacenan las aplicaciones, bibliotecas, ficheros, etc. que son necesarios para el correcto funcionamiento de la aplicación.
  - No contienen un SO completo.
- Comparten/utilizan los recursos del propio SO “host”.
- Permiten la creación de entornos replicables, consistencia entre entornos de prueba y de producción, reducir costes, etc.
- Es una arquitectura enfocada a microservicios.



Virtualización y contenedores

Máquina virtual vs contenedor

- **Máquinas virtuales**
  - Se ejecutan sobre un hipervisor.
  - Crean un SO completo → se consumen más recursos.
  - El estado de la máquina depende de su ejecución y no es inmutable.
  - Limitaciones para la creación de múltiples máquinas virtuales simultáneas.
- **Contenedores**
  - Se ejecutan sobre un motor de contenedores.
  - Sólo contienen lo necesario para ejecutar la aplicación.
  - Son inmutables.
  - Poca sobrecarga. En el “host” pueden ejecutarse miles de contenedores.

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA
Escuela Superior Ingeniería Informática
DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS
XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE
CFR FERROL

Tema 3. Implantación de sistemas seguros de despliegue de software

- Docker

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro
GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Docker

- Es un proyecto open source que permite la creación y despliegue de contenedores.

- Está escrito en el lenguaje de programación Go y se sirve de características del kernel de Linux para proporcionar su funcionalidad.

- Principalmente está formado por los siguientes elementos:
  - **Namespaces:** utiliza los nombres de espacio del kernel para proporcionar un entorno aislado de procesos → contenedor.
  - **Cgroups:** hace uso de los grupos de control del kernel para la asignación compartida de recursos y aislamiento de los contenedores.
  - **UnionFS/OverlayFS:** permite combinar distintos sistemas de archivos formando un único sistema (ejecución de múltiples instancias de contenedor).

- Hace uso de **AppArmor**, **SELinux** y de las capacidades del kernel para añadir seguridad a los contenedores.
  - https://docs.docker.com/engine/security/apparmor

Docker
Componentes

- **Imagen:** paquete que contiene todo lo necesario para ejecutar contenedores (SO base, aplicaciones, dependencias, configuración, etc). Están formadas por capas y sirven como plantilla para instanciar contenedores.

- **Contenedor**: entornos aislados de procesos que comparten el kernel del "host". Suponen una capa de abstracción para la ejecución de aplicaciones. Instancias en ejecución de una imagen.

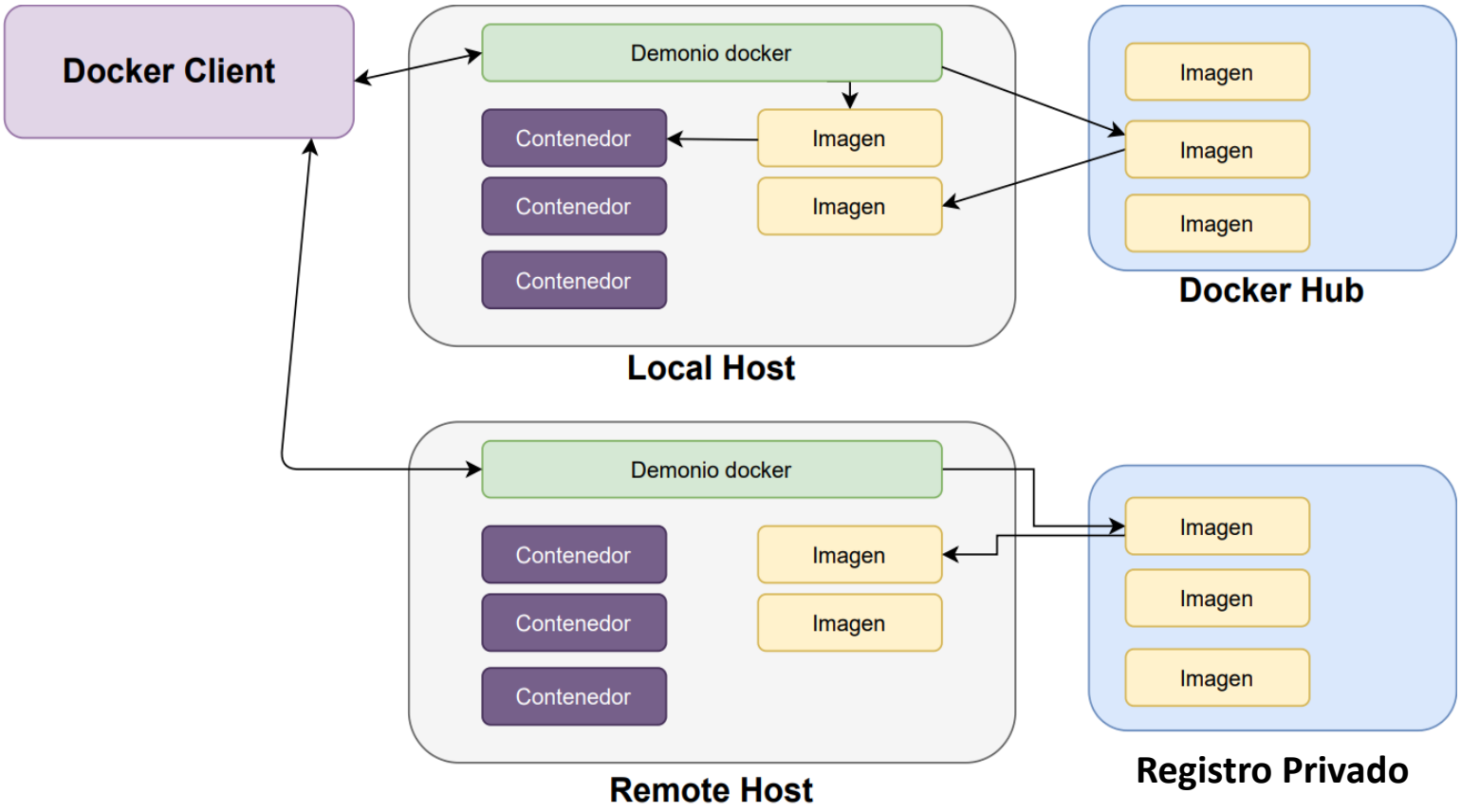
- **Registros de imágenes**: servicio de almacenamiento de imágenes (i.e. repositorios). Pueden ser tanto públicos como privados.

- **Docker daemon (dockerd)**: motor encargado de gestionar todos los componentes relacionados con Docker. Permanece a la escucha de peticiones a la Docker API enviadas desde el cliente.

- **Docker client (CLI)**: interfaz de línea de comandos que nos permite interactuar con el motor de Docker.

Docker

Arquitectura y componentes



Docker
Instalación

- **Docker Desktop (Mac/Windows):** https://docs.docker.com/get-docker
- **Distribuciones Linux**: https://docs.docker.com/engine/install

- Verificar la instalación:
  - *docker*
  - *docker version*
  - *docker info*

Docker

Docker CLI

```
root@ubuntu-20:/home/usuario/T3# docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/root/.docker")
  -c, --context string     Name of the context to use to connect to the daemon (overrides DOCKER_HOST env
                           var and default context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/root/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/root/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/root/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  app*        Docker App (Docker Inc., v0.9.1-beta3)
  builder     Manage builds
  buildx*     Build with BuildKit (Docker Inc., v0.6.1-docker)
  config      Manage Docker configs
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  node        Manage Swarm nodes
```

Docker

Ejemplo: subcomando *container*

```
root@ubuntu-20:/home/usuario/T3# docker container

Usage:  docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
```

Docker

Registros de imágenes

- Almacén de repositorios de imágenes. Pueden ser tanto públicos como privados (al ser open source).

- Por defecto, utiliza el registro público de Docker Hub.
  - *docker pull nginx*
  - *docker images*

- En el mismo Docker Hub podemos encontrar una imagen para desplegar nuestro propio registro de imágenes privado.
  - *docker pull registry*

Docker
Imágenes

- Tienen permiso de solo lectura y representan el estado de la aplicación en un momento específico.

- Son usadas como base/plantilla para construir y ejecutar contenedores.

- Están construidas a base de capas, donde cada capa es una modificación sobre el estado anterior de la imagen.
  - Imagen oficial de PHP.
  - *docker image history php*

- Las imágenes de una misma aplicación o servicio se almacenan ordenadas en repositorios. A cada versión de una imagen específica se le asigna un tag (en formato “nombre:tag”).
  - Imagen oficial de MySQL.
  - *docker pull mysql:5.7*

Docker
Contenedores

- Es un paquete software que incluye todo lo necesario para que se pueda ejecutar la aplicación.

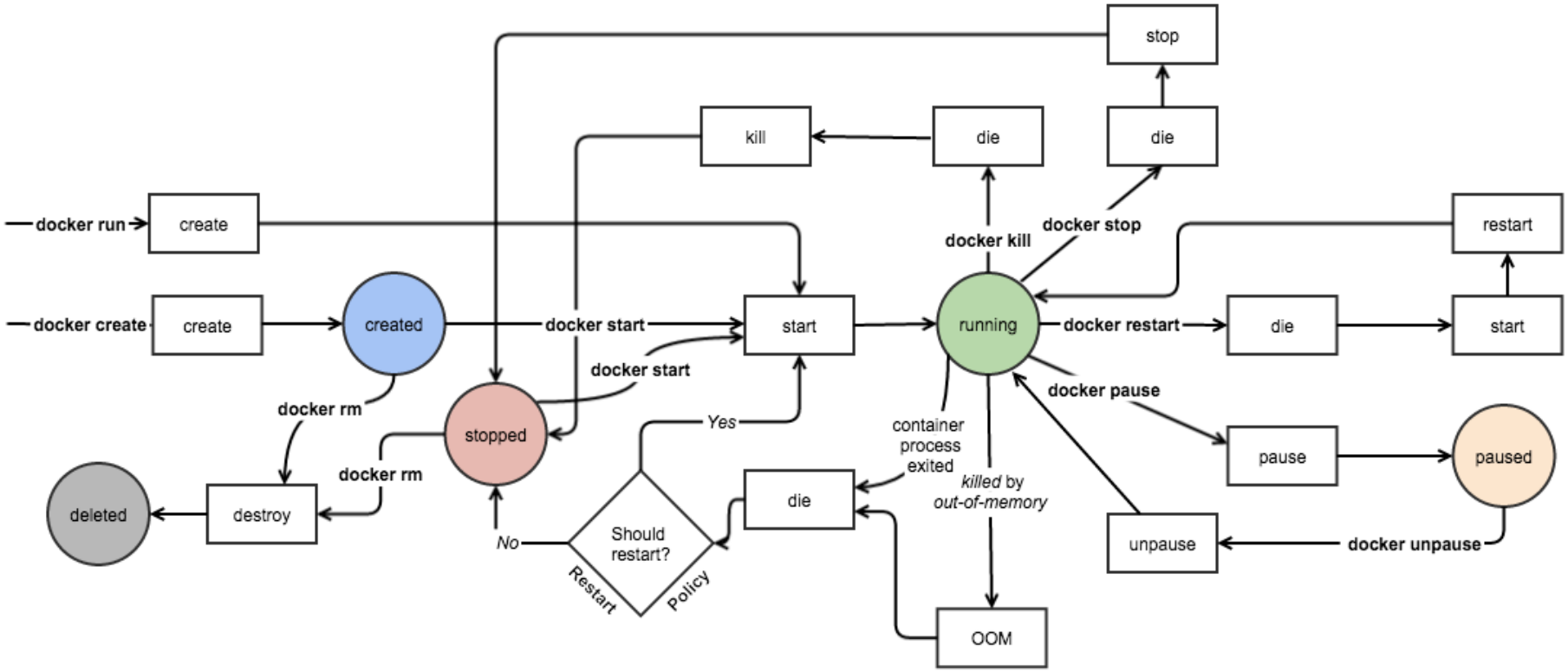
- Cuando se crea un contenedor en Docker a través de una imagen, se añade sobre ésta una nueva capa con permisos de lectura y **escritura**. Los cambios en esta capa se pierden cuando se para o elimina el contenedor.
  - **La imagen prevalece!**

- Un contenedor es un proceso. Aunque es posible que dentro del contenedor se ejecuten varios procesos se recomienda que sólo ejecute un proceso/servicio.

Docker

Ciclo de vida de los contenedores

- Comandos, acciones internas y estados:



Docker

Ciclo de vida de los contenedores

- La diferencia entre los comandos *docker stop* y *docker kill* es la señal que se envía al proceso principal del contenedor al terminar:
  - *docker stop* envía la señal SIGTERM para intentar acabar con el contenedor de forma "pacífica". Después de un timeout, si el contenedor no ha finalizado se envía una señal SIGKILL**.**
  - *docker kill* envía directamente una señal SIGKILL para finalizar el contenedor/proceso de forma abrupta.

- Se recomienda usar *docker stop* para finalizar un contenedor. Con una señal SIGTERM, internamente se realizan tareas de limpieza, actualización de ficheros…

Docker
Contenedores

- **Ejecutar un contenedor:**
  - *docker run -it -p 8080:80 --name mi_servidor1 nginx*

- **Listar contenedores en ejecución:**
  - *docker ps*
  - *docker container ls*
  - En ambos comandos se puede usar la flag "*-a*" para mostrar todos los contenedores (en ejecución y parados).

- **Configurar variables de entorno:**
  - *docker run -e user=admin -e pass=1234 --name mi_servidor2 -d nginx*
  - También se le puede pasar directamente un fichero de variables de entorno mediante la flag "*--env-file*".

Docker
Contenedores

- **Comandos para interactuar con un contenedor:**
  - *docker exec -it mi_servidor2 /bin/bash*
  - *docker cp secret.txt mi_servidor2:/*
  - *docker cp mi_servidor2:/secret.txt .*
  - *docker attach mi_servidor1*

- **Parar o matar un contenedor:**
  - *docker stop mi_servidor1*
  - *docker kill mi_servidor2*

- **Volver a iniciar un contenedor:**
  - *docker start mi_servidor1*
  - *docker restart mi_servidor1* → reinicia el contenedor (da igual si está activo o parado).

Docker
Contenedores

- **Eliminar un contenedor:**
  - *docker rm mi_servidor1* → **ERROR**
  - *docker rm -f mi_servidor1*

- **Eliminar todos los contenedores:**
  - *docker container prune* → elimina los contenedores parados.
  - *docker rm -f $(docker ps -a -q)* → elimina **TODOS** los contenedores.

- Igualmente se pueden eliminar todas las imágenes:
  - *docker rmi -f $(docker images -a -q)*
  - Cuidado con las imágenes que están siendo usadas por contenedores en ejecución → **ERROR**

Docker

Imágenes y contenedores (Ejercicio)

- Vamos a configurar nuestro propio registro privado de imágenes.
  - Pista → entramos a Docker Hub y buscamos “registry”.

Docker

Imágenes y contenedores (Ejercicio)

- *docker pull registry* (opcional)
- *docker run -d -p 5000:5000 registry*
- *docker pull nginx*
- *docker tag nginx localhost:5000/nginx:custom* → crea una nueva imagen con etiqueta personalizada que apunta a nuestro registro privado.
- *docker pull nginx:custom* → **ERROR**
- *docker push localhost:5000/nginx:custom*
- *docker pull nginx:custom* → **ERROR**
- *docker rmi -f $(docker images -a -q)* → ¿?
- *docker pull localhost:5000/nginx:custom*

Docker

Imágenes y contenedores (Ejercicio)

- **Nota**: si queremos utilizar un repositorio público como Docker Hub para subir nuestras imágenes, primero nos registramos en su página web:
  - https://hub.docker.com

- *docker tag nginx DH_USER/nginx:custom* → crea una nueva imagen con etiqueta personalizada que apunta a nuestro usuario de Docker Hub.

- Ya en Docker CLI:
  - *docker login* → introducimos usuario y contraseña.
  - *docker push DH_user/nginx:custom*

Docker

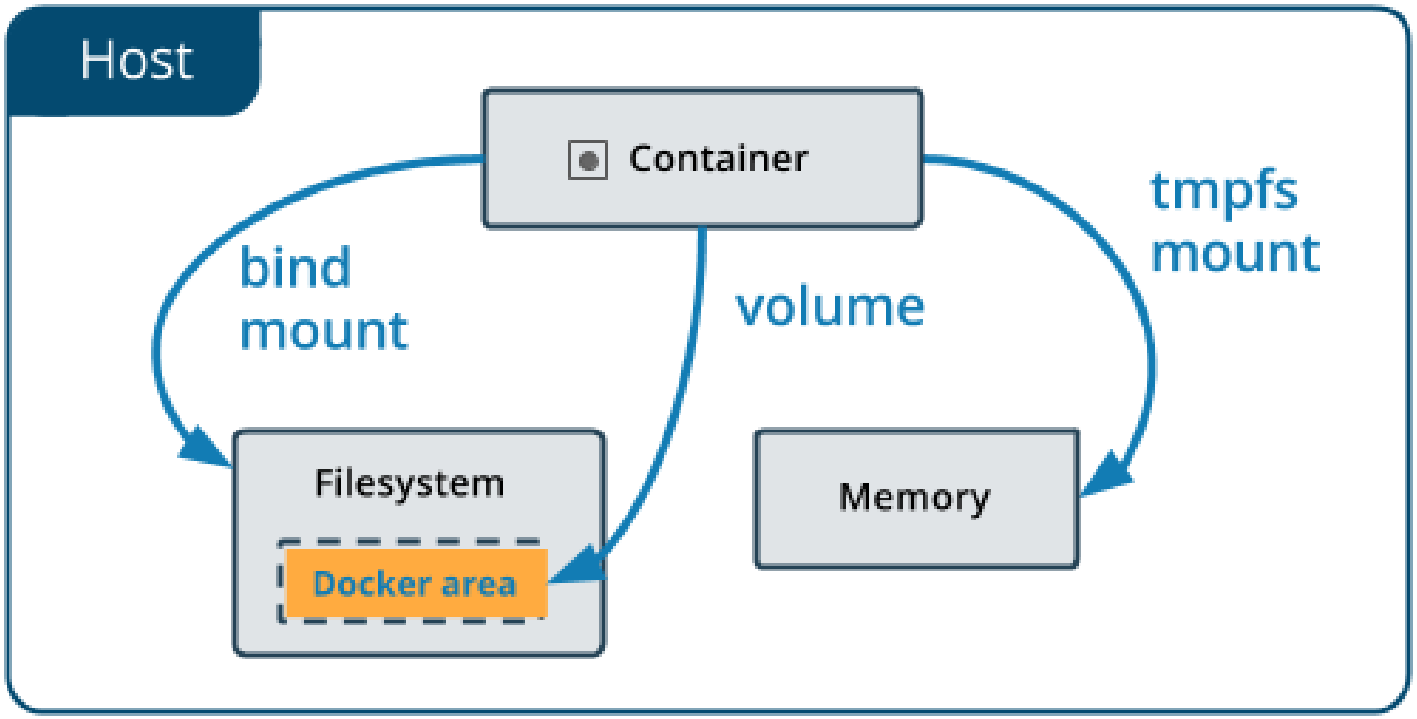
Almacenamiento y persistencia

- Cuando se instancia un contenedor se añade una nueva capa que es de lectura y escritura. Todos los cambios que se producen en el contenedor se almacenan en esta capa.
  - **Al apagar el contenedor se pierden!**

- Existen diferentes formas de que los datos sean persistentes y no se pierdan al apagar el contenedor:
  - **Bind mounts:** monta un directorio del “host” en el contenedor (funcionalidad limitada, confían en el sistema de ficheros del “host”).
  - **Tmpfs mounts:** permiten crear ficheros fuera de la capa de escritura del contenedor. El mount es temporal y se almacena en la memoria RAM del “host”. No es posible compartir este tipo de persistencia entre contenedores.
  - **Volumes:** son objetos gestionados por Docker y pueden ser compartidos entre contenedores. Son almacenados en un directorio especial de Docker.

Docker

Almacenamiento y persistencia



Docker

Almacenamiento y persistencia

- **Crear un volumen:**
  - *docker volume create mi_volumen1*

- **Obtener volúmenes:**
  - *docker volume ls*
  - *docker volume inspect mi_volumen1* → muestra información detallada de un volumen.

- **Iniciar un contenedor asignándole un volumen:**
  - *docker run --mount source=mi_volumen1,target=/mi_app -d nginx*
  - *docker run --mount source=mi_volumen1,target=/app -it php*

- **Eliminar volúmenes:**
  - *docker volume rm mi_volumen2*
  - *docker volume prune* → elimina todos los volúmenes que no están siendo usados por contenedores.

Docker

Almacenamiento y persistencia

- **Iniciar un contenedor asignándole un bind mount:**
  - *docker run --mount type=bind,source="$(pwd)",target=/app -d nginx*

- **Iniciar un contenedor asignándole un tmpfs mount:**
  - *docker run --mount type=tmpfs,target=/app -it nginx /bin/bash* → ¿?

- Hablando de persistencia… Es posible persistir cualquier cambio que se haga en un contenedor (p.e. configuración) creando una nueva imagen a partir del contenedor en ejecución:
  - *docker run -it --name mi_servidor nginx /bin/bash*
  - *docker commit mi_servidor nginx:mi_nginx2.0*
  - *docker images -a*
  - *docker run -it nginx:mi_nginx2.0* → ¿?

Docker
Redes

- Docker posee distintas funciones de red, o **network drivers**, de manera que los contenedores puedan comunicarse entre sí:
  - **Bridge:** se utiliza para permitir la comunicación entre contenedores dentro del mismo “host”. Es el driver por defecto en caso de omitir la configuración de red.
  - **Host:** se utiliza para eliminar el aislamiento de red entre el contenedor y el “host” donde se ejecuta el demonio Docker. Usa la red del “host” directamente.
  - **Overlay:** conecta varios demonios de Docker para crear una red virtual, permitiendo conectar contenedores entre entornos distribuidos que se están ejecutando en diferentes “hosts”.
  - **None:** deshabilita la configuración de red en el contenedor.

Docker
Redes

- **Crear una red:**
  - *docker network create -d bridge mi_red1*

- **Obtener las redes existentes:**
  - *docker network ls* → existen varias redes por defecto ya creadas que no pueden ser eliminadas (“brigde”, “host”, “none”).

- **Obtener información de una red concreta:**
  - *docker network inspect mi_red1*

- **Eliminar una red:**
  - *docker network rm mi_red1*

Docker
Redes

- **Ejecutando un contenedor en una red bridge:**
  - *docker run --network mi_red2 -it ubuntu*
- **Ejecutando un contenedor en una red host:**
  - *docker run --network host -it ubuntu*
- Comprobándolo…
  - *apt-get update*
  - *apt-get install net-tools iputils-ping*
  - *ifconfig*

Docker

Dockerfile

- Archivo de texto que contiene las instrucciones necesarias para construir imágenes de forma automatizada.
- Cada comando del Dockerfile es una capa diferente en la imagen que se va a generar.
- Se utiliza el comando *docker build .* para crear la imagen desde el archivo Dockerfile.
- Se puede crear un fichero **.**dockerignore para excluir ficheros y directorios que se envían al contenedor.

Docker

Dockerfile (Ejemplo)

```
 1 FROM debian:latest as compilacion
 2 RUN apt-get update && apt-get -y install gcc
 3 WORKDIR /app
 4 COPY ./servidor_echo.c .
 5 RUN gcc -static servidor_echo.c -o servidor_echo.o
 6
 7 FROM alpine:latest
 8 WORKDIR /app
 9 COPY --from=compilacion /app/servidor_echo.o /app
10 EXPOSE 8080
11 ENTRYPOINT ["./servidor_echo.o"]
12 CMD ["8080"]
```

Docker
Dockerfile

- **FROM:** directiva que indica el nombre de la imagen en la que se basa el Dockerfile.
  - *FROM ubuntu:latest*
  - *FROM ubuntu:latest AS nombre_etapa_build* → construcciones intermedias que se pueden usar como base en posteriores directivas dentro del mismo Dockerfile.

- **RUN:** ejecuta un comando. Se puede especificar en formato "shell" o en formato "exec".
  - *RUN comando* → formato "shell"
  - *RUN ["ejecutable", "parametro1", "parametro2"]* → formato "exec"

- **LABEL:** añade metadatos a la imagen en formato clave-valor.
  - *LABEL version="1.0"*
  - *LABEL maintainer="Admin"*

- **EXPOSE:** para informar que el contenedor estará escuchando en determinado puerto una vez se encuentre en ejecución.
  - *EXPOSE 8080*

Docker
Dockerfile

- **ENV**: establece variables de entorno a usar por la imagen/contenedor en ejecución.
  - *ENV USER="admin"*

- **ADD:** copia archivos o directorios locales a la ruta indicada dentro de la imagen. También permite descargar archivos remotos mediante URLs y descomprime archivos ".tar" automáticamente en el destino.
  - *ADD archivo /ruta_destino/*

- **COPY:** al igual que la instrucción ADD, sirve para copiar archivos desde el "host" a la imagen (solo permite copiar archivos o directorios locales).
  - *COPY archivo /ruta_destino/*

- **ENTRYPOINT:** indica el ejecutable que se usará cuando se inicie el contenedor → i.e. configurando el contenedor como un ejecutable. El ENTRYPOINT por defecto es "*/bin/sh -c".*
  - *ENTRYPOINT ping 8.8.8.8* → formato "shell"
  - *ENTRYPOINT ["ping", "8.8.8.8"]* → formato "exec"

Docker
Dockerfile

- **CMD:** define lo que será ejecutado cuando se inicie el contenedor. Se corresponde al comando/parámetros que se le pasan al ejecutable del ENTRYPOINT. Como mínimo, se debe especificar o bien un CMD o bien un ENTRYPOINT en el Dockerfile:
  - *CMD ping 8.8.8.8* → formato "shell" (sin ENTRYPOINT).
  - *CMD ["ping", "8.8.8.8"]* → formato "exec" (sin ENTRYPOINT).
  - CMD ["8.8.8.8"] → como parámetro al ENTRYPOINT.

- **USER**: establece el usuario y el grupo con el que llevar a cabo la ejecución de comandos. Se pueden establecer diferentes USER a lo largo del Dockerfile.
  - *USER user:group* → el grupo es opcional.

- **WORKDIR**: establece el directorio de trabajo para las instrucciones RUN, CMD, ENTRYPOINT, COPY y ADD. Se pueden usar diferentes WORKDIR a lo largo del Dockerfile.
  - *WORKDIR /ruta*

- **VOLUME**: define volúmenes a utilizar por el contenedor una vez esté en ejecución.
  - *VOLUME /volumen*

Docker

Dockerfile (PoC)

- Vamos a construir, ejecutar y probar un servidor echo con Docker y Dockerfile.
- Código en C del servidor echo: https://github.com/mafintosh/echo-servers.c/blob/master/tcp-echo-server.c
- Utilizaremos una imagen multifase (multi-stage builds).

```
 1 FROM debian:latest as compilacion
 2 RUN apt-get update && apt-get -y install gcc
 3 WORKDIR /app
 4 COPY ./servidor_echo.c .
 5 RUN gcc -static servidor_echo.c -o servidor_echo.o
 6
 7 FROM alpine:latest
 8 WORKDIR /app
 9 COPY --from=compilacion /app/servidor_echo.o /app
10 EXPOSE 8080
11 ENTRYPOINT ["./servidor_echo.o"]
12 CMD ["8080"]
```

Docker

Dockerfile (PoC)

- **Construir una imagen a partir de un Dockerfile:**
  - *docker build -t servidor_echo .* → importante el punto final.
- **Ejecutamos un contenedor usando la nueva imagen:**
  - docker run -it --name servidor_echo *servidor_echo* → ¿?
  - docker run -it -p 8080:8080 --name servidor_echo *servidor_echo*
- **Probamos el funcionamiento del servidor echo:**
  - nc localhost 8080

Docker

Docker Compose

- Herramienta escrita en Go para ejecutar múltiples contenedores de manera ordenada y sincronizada.
- Permite definir de manera declarativa en un archivo YAML los servicios que se van a levantar así como las dependencias entre ellos.
- Es una capa de abstracción para levantar contenedores de forma automática, pero no añade nada nuevo a Docker (herramienta aparte).
- Se utiliza en distintos entornos como son el desarrollo, testing y puesta en producción de aplicaciones.
- **Instalación** → https://docs.docker.com/compose/install

Docker

Docker Compose

```yaml
services:

    mysqlserver:
        image: mysql
        restart: always
        hostname: mysqlserver
        command: --default-authentication-plugin=mysql_native_password
        environment:
            MYSQL_ROOT_PASSWORD: root
            MYSQL_USER: esiiabctf
            MYSQL_PASSWORD: licores
            MYSQL_DATABASE: login
        volumes:
            - "./mysql_init/:/docker-entrypoint-initdb.d/"
    web-server:
        build: .
        image: webserver
        hostname: webserver
        restart: always
        environment:
            USER: "esiiabctf"
            PASSWORD: "licores"
            SERVER: "mysql-server"
        ports:
            - 7777:80
```

Docker
Buenas prácticas

- Fortificación del “host”, limitar y controlar usuarios que pueden acceder a Docker, particiones diferentes para contenedores, etc.
- Utilizar usuarios no privilegiados para ejecutar la aplicación dentro del contenedor.
- Uso de imágenes con la funcionalidad mínima requerida para ejecutar la aplicación. Uso de imágenes multifase cuando se requiera de utilidades, información, etc. que sólo se necesitan para construir la imagen final.
- No montar directorios sensibles del “host” en un contenedor.
- Ejecución de contenedores con las capacidades mínimas de Linux.
  - Flags *--cap-drop*, *--cap-add*

Docker
Buenas prácticas

- Uso de la instrucción COPY en vez de ADD.
- Eliminar los permisos de setuid y setgid de los binarios de la imagen.
- No correr ningún contenedor en modo privilegiado.
- Análisis de las imágenes periódicamente para detectar posibles vulnerabilidades o problemas → clair, dagda, anchore, snyk.
- Configurar el motor de Docker para que solo permita ejecutar imágenes firmadas.

UCLM UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI Departamento de Sistemas Informáticos

XUNTA DE GALICIA | CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 3. Implantación de sistemas seguros de despliegue de software

- Orquestación de contenedores. Kubernetes

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Orquestación de contenedores

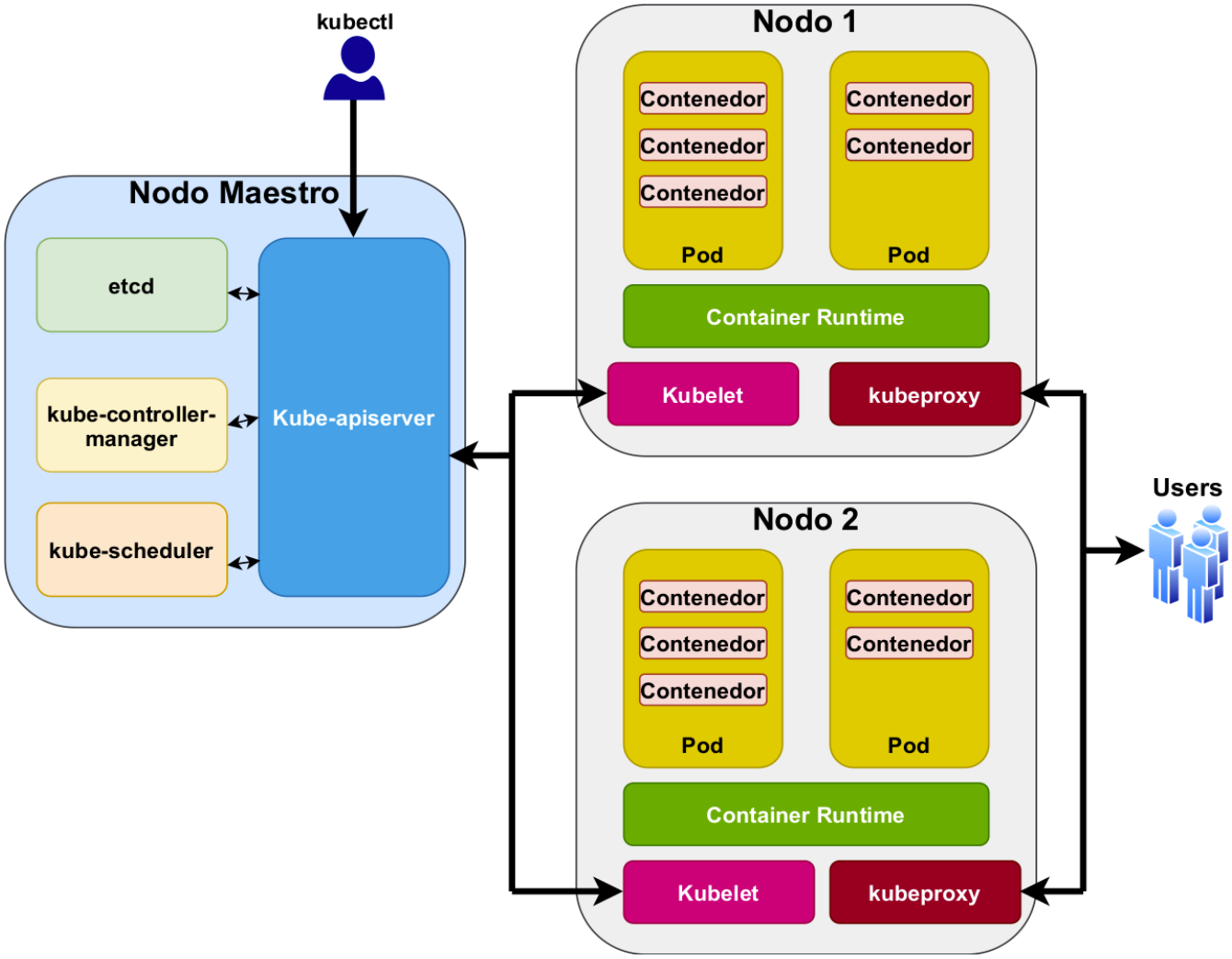
- Plataforma para gestionar y organizar servicios/aplicaciones (contenedores) de forma automatizada y proporcionando escalabilidad.
- Basan su funcionamiento en un clúster de nodos interconectados donde uno (o varios) de ellos actúa como nodo gestor.
- Proveen de redundancia a los contenedores de manera que es posible configurar varias réplicas de un mismo servicio/aplicación.
- Monitorizan el estado de los contenedores de manera que si falla alguno, vuelva a ejecutarse de forma transparente al administrador (tolerancia a fallos).
- Son capaces de distribuir automáticamente la carga de los contenedores entre los diferentes nodos del clúster (balanceo de carga).
- Ejemplos → Docker Swarm, Kubernetes, Apache Mesos, Marathon…

Kubernetes

- Kubernetes (k8s) es una plataforma de código abierto diseñada originalmente por Google y liberada en 2014 para toda la comunidad.
- Arquitectura basada en microservicios que permite desacoplar la infraestructura de los propios servicios/aplicaciones.
- Soporta diferentes entornos de ejecución de contenedores → Docker, containerd, CRI-O, rktlet…
- Existen plataformas de orquestación en la nube que abstraen al administrador la configuración del clúster y de sus nodos → Amazon EKS, Google Cloud Kubernetes, Azure Kubernetes, etc.
- Extensible: se le pueden agregar otros proyectos para darle más capacidades (plugins, extensiones).

Kubernetes

Arquitectura



Kubernetes

Componentes nodo master

- ***kubectl***: herramienta/interfaz de línea de comandos que permite administrar el clúster. Interactúa con kube-apiserver lanzando peticiones a su API mediante HTTP.

- ***kube-apiserver:*** administración central que recibe todas las peticiones HTTP de los demás componentes (kubectl, kubelet, controllers…). Es el único componente que interacciona con etcd.

- ***etcd***: base de datos clave-valor distribuida que almacena toda la información de control del clúster de Kubernetes (Pods, namespaces, servicios, etc).

Kubernetes

Componentes nodo master

- ***kube-controller-manager***: conjunto de procesos de control (daemons) ejecutados en segundo plano que regulan el estado actual del clúster. Por ejemplo:
  - Proceso que controla qué nodos worker están activos y cuáles no.
  - Proceso que mantiene el número de Pods correcto en ejecución.

- ***Kube-scheduler***: componente que planifica Pods, contenedores y servicios por todos los nodos del clúster. Necesita conocer los recursos totales de cada nodo y la asignación de estos a cada carga de trabajo. Se ejecuta cada vez que sea necesario planificar Pods y se nutre de un gran conjunto de políticas:
  - Requisitos QoS, hardware, software, de disponibilidad, de rendimiento, de capacidad, etc.

Kubernetes

Componentes nodo worker

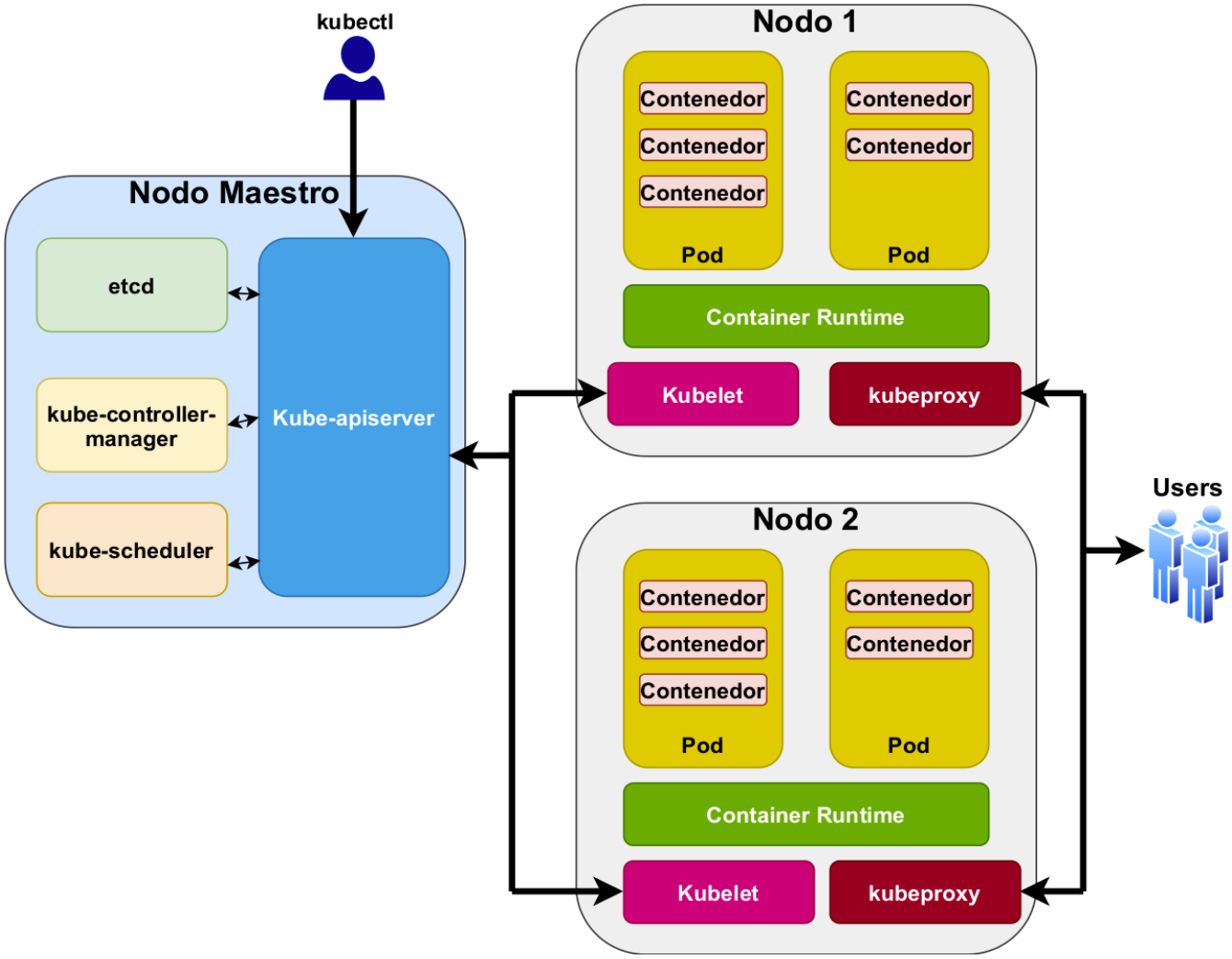
- ***kubelet***: daemon principal que monitoriza regularmente los Pods en ejecución y reporta el estado al nodo master. También se encarga de consultar las especificaciones del kube-apiserver y se asegura de que son cumplidas.
  - Nota: no gestiona ni monitoriza contenedores que no hayan sido creados por Kubernetes.

- ***kube-proxy:*** proxy de red que expone los servicios/aplicaciones hacia el mundo externo y reenvía las peticiones entrantes hacia los Pods/contenedores correctos.

- ***Container Runtime***: gestor de imágenes, contenedores, interfaces de red y almacenamiento para ejecutar contenedores dentro de Pods.

Kubernetes

Arquitectura y componentes



Kubernetes
Instalación

- **Docker Desktop (Mac/Windows):** https://docs.docker.com/get-docker
- **Distribuciones Linux**: documento Aula Virtual.

- Verificar la instalación:
  - *kubectl*
  - *kubectl cluster-info*
  - *eval $(minikube -p minikube docker-env)* → ¿?

Kubernetes
Objetos

- Son entidades que existen dentro de Kubernetes. Representan el estado actual del clúster. Pueden ser de distintos tipos:
  - Pods, services, volumes, namespaces, deployments, secrets, etc.

- Se pueden crear de manera imperativa (mediante kubectl) o declarativa (archivos de configuración YAML o JSON, aunque también ejecutados por kubectl).

- Archivos YAML. Todos los objetos comparten los siguientes elementos:
  - ***apiversion*:** indica la versión de la API de Kubernetes en uso.
  - ***kind*:** indica el tipo de objeto que se está creando.
  - ***metadata*:** ciertos metadatos del objeto como el nombre, namespace al que pertenece, etc.
  - ***spec*:** define el estado deseado del objeto. Este campo tendrá el formato específico del objeto que se quiere crear.

Kubernetes
Objetos (PoC)

- **Comando run:** crea y ejecuta una imagen en un Pod.
  - *kubectl run mi-servidor --image=nginx*
- **Comando get:** obtención de cualquier objeto.
  - *kubectl get nodes*
  - *kubectl get pods -o json*
  - *kubectl get pod mi-servidor -o yaml*
- **Comando describe:** muestra información detallada sobre cualquier objeto.
  - *kubectl describe pods*
  - *kubectl describe node minikube*

Kubernetes
Objetos (PoC)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mi-pod
spec:
  containers:
  - name: mi-pod
    image: php:7.0-apache
    ports:
      - containerPort: 80
```

- **Comando delete:** elimina objetos de Kubernetes.
  - *kubectl delete pod miservidor*

- **Comando create:** crea un objeto de Kubernetes. Es posible especificar un archivo YAML o JSON.
  - *kubectl create namespace mi-namespace*
  - *kubectl create -f prueba-pod.yml*

- **Comando apply:** sobrescribe la configuración de un objeto. Si no existiera, se crea el objeto directamente (solo es posible modificar cierta información).
  - *kubectl apply -f prueba-pod.yml*

Kubernetes

Namespaces

- Es una abstracción para crear clúster lógicos dentro del clúster físico de Kubernetes.

- Por defecto, Kubernetes cuenta con tres namespaces ya definidos:
  - **default:** todos los objetos que se creen y no especifiquen un espacio de nombres utilizarán este.
  - **kube-system:** se utiliza para los objetos que son creados por el propio sistema de Kubernetes.
  - **kube-public:** este espacio de nombres puede ser leído por cualquier usuario del clúster.

- Ciclo de vida de un namespace: "**Active**" y "**Terminating**".

Kubernetes

Namespaces

- **Comandos relacionados**:
  - *kubectl create namespace mi-namespace*
  - *kubectl get namespaces*
  - *kubectl get pods -A*
  - *kubectl get pods --namespace mi-namespace*
  - *kubectl run mi-pod --image=nginx -n mi-namespace*

Kubernetes
Pods

- Es la unidad mínima de trabajo en Kubernetes y donde se ejecutan los contenedores.

- Dentro de un Pod se pueden ejecutar múltiples contenedores aunque por lo general, la asignación será de uno a uno.

- Estructura que encapsula recursos de almacenamiento (volúmenes) y redes:
  - Un Pod actúa de host lógico, por lo que tiene su propia dirección IP y puertos.

- Ciclo de vida de un Pod en cuanto a los contenedores que contiene: "**Pending**", "**Running**", "**Succeeded**", "**Failed**", "**Unknown**".
  - También podemos observar el pre-estado "**ContainerCreating**".

Kubernetes
Pods

- **Forma imperativa:**
  - *kubectl run mi-pod --image=php:7.0-apache --port=80*
  - *kubectl cluster-info* → ¿Ping al Pod?

- **Forma declarativa:**
  - *kubectl apply -f fichero.yml*

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mi-pod
spec:
  containers:
  - name: mi-pod
    image: php:7.0-apache
    ports:
      - containerPort: 80
```

Kubernetes

Controladores. ReplicaSet

- Objeto que se utiliza para gestionar las réplicas de los Pods y el ciclo de vida de estos.
- Se encarga de mantener el número de instancias del Pod que se le configuran (autocuración, autoescalado).
- Se recomienda el uso del objeto “Deployment” en vez de “ReplicaSet” ya que gestionan también las réplicas además de otras características.
- **Comandos relacionados:**
  - *kubectl get replicaset -A*
  - *kubectl create -f prueba-rs.yml*
  - *kubectl scale replicaset web --replicas=5*

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
```

Kubernetes

Controladores. Deployment

- Proporcionan una manera declarativa (YAML o JSON) de actualizar Pods y ReplicaSets.

- Describen el estado deseado en el clúster → el controlador es capaz de crear o eliminar réplicas de Pods para preservar dicho estado.

- Se puede definir la estrategia que seguirá un Deployment para actualizar los Pods:
  - **Recreate:** destruye los Pods existentes y después se crean los nuevos.
  - **RollingUpdate:** se actualizan los Pods actuales de manera ordenada.

- **Comandos relacionados:**
  - *kubectl get deployments -A*
  - *kubectl create -f prueba-deploy.yml*
  - *kubectl scale deployment nginx-deployment --replicas=10*
  - *kubectl delete deployment nginx-deployment*

Kubernetes

Controladores. Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

```yaml
spec:
  replicas: 3
  strategy:
    type: Recreate
```

```yaml
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2
      maxUnavailable: 0
```

Kubernetes
Servicios

- Forma abstracta de exponer aplicaciones que se ejecutan dentro de Pods como servicios de red.

- Por ejemplo, un servicio permite que:
  - Los Pods se encuentren entre sí sin tener que conocer sus direcciones IP.
  - Los Pods puedan ser accesibles desde el exterior.

- Normalmente se utilizan **selectores** (directiva en los YAML) para establecer los Pods a los que se les aplica el servicio.

Kubernetes
Servicios

- **ClusterIP:** expone un servicio en una IP privada e interna del clúster, por lo que permite el acceso directo de otras aplicaciones (Pods). Es el tipo de servicio que se utiliza por defecto.

- **NodePort:** expone un servicio en cada nodo del clúster (utilizando el mismo puerto) y es accesible desde fuera del clúster. El tráfico entrante por cada nodo se redirige al servicio y, finalmente, a los Pods.

- **LoadBalancer:** expone el servicio externamente utilizando un balanceador de carga. Solo se le asigna una IP encargada de reenviar todo el tráfico.

Kubernetes
Servicios

- **Comandos relacionados:**
  - *kubectl create -f prueba-service-lb.yml*
  - *kubectl cluster-info* ←→ *minikube ip*
  - *kubectl get services* ←→ *minikube service nginx-service*

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - port: 8080
      targetPort: 80
      nodePort: 30010
```

Kubernetes
Servicios (PoC)

- Vamos a comprobar el funcionamiento de los servicios tipo **LoadBalancer**.
  - Generamos un Dockerfile basado en una imagen NodeJS que va a ejecutar un pequeño servidor que devuelve el hostname del contenedor/Pod donde se ejecuta.

```
FROM node
COPY server.js .
EXPOSE 8080
CMD node server.js
```

```
var http = require('http');
var host;

var handleRequest = function(request, response) {
        response.writeHead(200);
        response.write("Respondiendo desde: ");
        response.write(host);
        response.end("\n");
};

var www = http.createServer(handleRequest);
www.listen(8080, function() {
        host = process.env.HOSTNAME;
});
```

Kubernetes
Servicios (PoC)

- Generamos declarativamente un Deployment basándonos en la imagen que acabamos de crear:
  - *kubectl create -f deployment.yml*
- Creamos y asignamos el servicio al Deployment de forma imperativa:
  - *kubectl expose deployment node-deployment --type=LoadBalancer*

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node-deployment
  labels:
    app: node-server
spec:
  replicas: 5
  selector:
    matchLabels:
      app: node-server
  template:
    metadata:
      labels:
        app: node-server
    spec:
      containers:
      - name: node-server
        image: node-server
        ports:
        - containerPort: 8080
        imagePullPolicy: Never
```

Kubernetes

Ingress

- Se utiliza para exponer rutas de tráfico HTTP/HTTPS desde el exterior a los servicios de dentro del clúster.
- El acceso a los servicios se declara mediante reglas que definen qué peticiones llegan a qué servicios.
- No es un tipo de servicio como tal, directamente es un objeto en Kubernetes. El resultado sería el mismo que exponer una aplicación utilizando NodePort o LoadBalancer.
- Permiten agrupar reglas de enrutamiento en un solo recurso de Kubernetes (separando por hosts, protocolo y paths).

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
    name: prueba-ingress
spec:
  rules:
    - host: mi-prueba-ingress.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: web-server
                port:
                  number: 7777
```

Kubernetes
Volúmenes

- Son objetos de almacenamiento que se encuentran ligados al tiempo de vida de un Pod.

- Son accesibles por cualquier contenedor dentro del Pod, es decir, son compartidos.

- Un Pod puede utilizar diferentes tipos de volúmenes:
  - **AwsElasticBlockStore**
  - **AzureDisk**
  - **HostPath**
  - **PersistentVolumeClaim**
  - **ConfigMap**
  - **Secret**
  - …

Kubernetes

Volúmenes. HostPath

- Monta un archivo o directorio del sistema de ficheros del host en el Pod.
- Pods con **misma configuración** pero que en distintos nodos podrían comportarse de **manera distinta**.

```
    env:
      - name: MYSQL_DATABASE
        value: login
      - name: MYSQL_PASSWORD
        value: licores
      - name: MYSQL_ROOT_PASSWORD
        value: root
      - name: MYSQL_USER
        value: esiiabctf
    image: mysql
    imagePullPolicy: Never
    name: mysqlserver
    volumeMounts:
      - mountPath: /docker-entrypoint-initdb.d/
        name: mysql-storage
    ports:
     - containerPort: 3306
       name: mysqlserver

hostname: mysqlserver
restartPolicy: Always
volumes:
  - name: mysql-storage
    hostPath:
      path: /mysql-init
```

Kubernetes

Volúmenes. ConfigMap y Secret

- Ambos son objetos de Kubernetes destinados a almacenar información de configuración del entorno.

- **ConfigMap:** almacenan datos en formato clave-valor. Pueden ser utilizados por los Pods como variables de entorno, como archivos de configuración o como argumentos para comandos.

- **Secret:** permiten almacenar información sensible en Kubernetes como contraseñas, claves SSH, tokens, etc. Por defecto, no usan cifrado sino que van codificados en Base64.
  - Aumenta el control de la información confidencial y reduce el riesgo de exposición accidental.

Kubernetes

Volúmenes. ConfigMap y Secret

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-mysql
data:
  MYSQL_DATABASE: login
  MYSQL_PASSWORD: licores
  MYSQL_ROOT_PASSWORD: root
  MYSQL_USER: esiiabctf
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mi-secreto
type: Opaque
data:
  mivar: bWlkYXRh
```

UCLM UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA | CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Tema 3. Implantación de sistemas seguros de despliegue de software

- DevOps

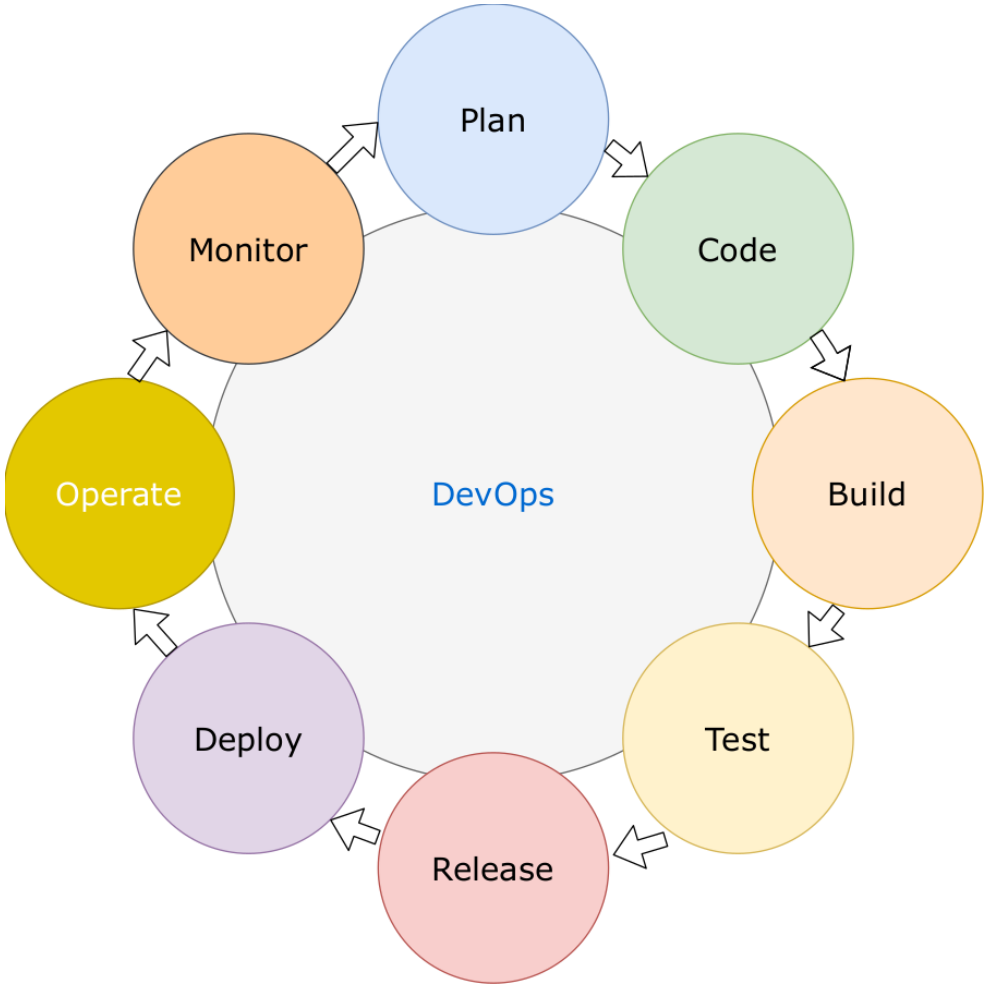
UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA | MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

DevOps

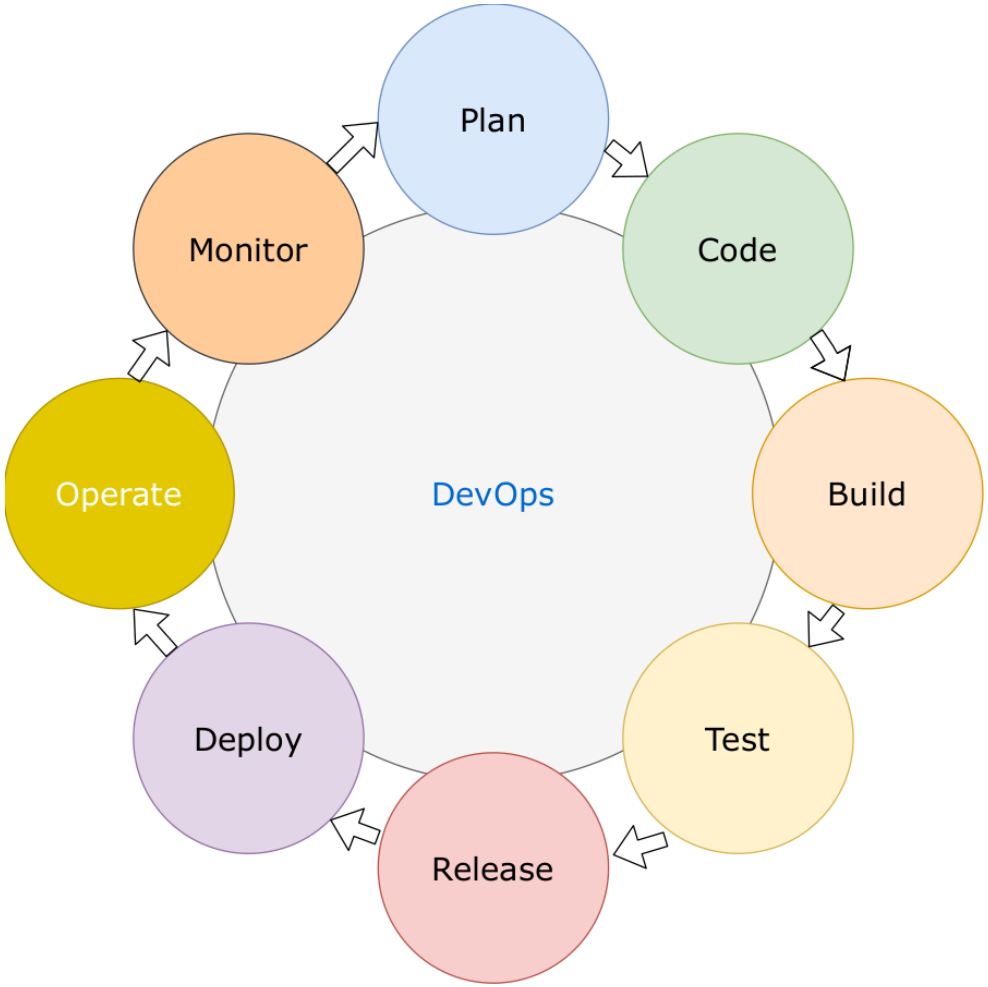
- Es un cambio de filosofía en la forma de trabajar entre los equipos de desarrollo y operaciones.
- La idea subyacente es la comunicación, colaboración e integración entre equipos de desarrollo y equipos de operaciones TI.
- El objetivo principal es reducir costes y tiempo mejorando la calidad del software desarrollado (agilizando, automatizando y monitorizando).
- Proceso asociado a metodologías de **integración continua** o **CI** (compilación y pruebas), **entrega continua** o **CD** (liberación de versiones) y **despliegue continuo** (en entornos reales/producción).
  - Procesos automáticos, repetibles y llevados a cabo lo más a menudo posible.

DevOps
Ciclo de vida (Dev)



- **Plan**: definición de requisitos del producto, así como la definición de roles y asignación de tareas a los integrantes del equipo.
- **Code**: fase de desarrollo e implementación del código del producto.
- **Build**: construcción de software (compilación) de forma automatizada y favoreciendo la creación de entornos replicables.
- **Test**: realización de pruebas de software automatizadas con el objetivo de encontrar y corregir errores.

DevOps
Ciclo de vida (Ops)



- **Release**: en esta fase se libera una nueva versión del producto (p.e. con una funcionalidad nueva o con errores solventados).
- **Deploy**: proceso automatizado en el cual se despliega la nueva versión en un entorno de producción. Este proceso no debe afectar al funcionamiento de la versión anterior ya desplegada.
- **Operate**: fase de configuración y corrección de errores que se encuentran en el entorno de producción al actualizar.
- **Monitor**: adopción de las medidas y métricas necesarias para controlar la salud, el comportamiento y el rendimiento del producto.

DevOps
SecDevOps

- Dentro del ciclo de vida de DevOps, el equipo de seguridad empieza a trabajar en las últimas fases.
  - Deploy, Operate y Monitor.

- Siguiendo la filosofía **SecDevOps**, la seguridad se aplica desde el inicio del desarrollo y es un proceso constante de manera que permite encontrar y subsanar problemas de seguridad desde el inicio y en todas las fases.

- Comunicación, colaboración e integración del equipo de seguridad, de desarrolladores y de TI desde el inicio del proyecto.

DevOps
Integración continua (CI)

- La **integración continua o CI** tiene como objetivo automatizar la integración de código de los desarrolladores.
  - Build, Test (local) y parte de Release.

- Facilita a los desarrolladores la unión (merge) de su código → permite analizar el producto global y encontrar posibles problemas de manera más sencilla.
  - Los desarrolladores envían sus cambios de forma periódica a un repositorio central que ejecuta un sistema de control de versiones (Git).

- En CI se **automatizan** las diferentes fases para validar que el código añadido no daña ningún componente principal del producto o aplicación.

DevOps
Integración continua (CI)

- Herramientas CI:
  - **Plan**: Jira, Trello, GitKraken Boards, etc.
  - **Code**: IDEs, Git, GitHub, Bitbucket, etc.
  - **Build**: Make, Maven, Gradle, Docker, etc.
  - **Test**: UnitTest (Python), Selenium (web), PHPUnit, etc.

  - **CI**: Jenkins, GitHub Actions, Travis, etc.
  - **DevSecOps**: OWASP ASVS, MobSF, git-secret, etc.

DevOps

Entrega continua (CD)

- Amplía la integración continua generando un paquete después de haber pasado las pruebas software funcionales.
- Su principal objetivo es tener un artefacto software preparado que pueda ser desplegado en un entorno de producción en cualquier momento.
- En CD también se **automatizan** todos los pasos necesarios para generar un paquete que pueda ser desplegado.
- Herramientas CD:
  - **Release**: Docker Hub, Nexus, Archiva, etc.

DevOps
Despliegue continuo

- Metodología/estrategia encargada de llevar a producción el resultado de las fases anteriores.

- **Automatización** de todos los pasos necesarios para poner a punto la aplicación en un entorno de producción.
  - La idea es que cualquier cambio de los desarrolladores pueda estar en producción de manera automática.

- Herramientas:
  - **Deploy**: Ansible, Chef, Puppet, etc.
  - **SecDevOps**: Nikto, NMAP, sqlmap, OWASP ZAP, etc.
  - **Simulación de fallos**: LoadRunner, JMeter, Blazemeter (apps móviles), etc.
  - **Monitorización**: Splunk, Nagios, Kibana, etc.

Control de versiones
Git

- **Git** es un sistema de control de versiones (CVS) open source.

- Su objetivo principal es el almacenamiento y administración de las diferentes versiones de los archivos que componen un producto software.

- Características:
  - Rápida recuperación de cualquiera de las versiones generadas con anterioridad → Actúan como copias de seguridad.
  - Comparación visual (diff) entre las diferentes versiones de un archivo → ¿Qué ha cambiado en la última versión? ¿Por qué?
  - Posibilidad de mantener diferentes versiones a la vez (branch) o dividir un proyecto en varias partes → Backend y frontend.

- Al igual que las imágenes de Docker en los registros, los proyectos en Git se almacenan en repositorios.

Control de versiones
GitHub

- Por el contrario, **GitHub** es una plataforma web online donde los usuarios pueden cargar y gestionar sus repositorios Git de forma remota.

- Facilita la compartición de código y la colaboración entre los integrantes de un equipo de desarrollo software.
  - Provee de repositorios públicos y privados.

- Fomenta la participación de los usuarios en proyectos open source.

GitHub

Control de versiones

Git y GitHub (PoC)

- **Instalar Git:** http://git-scm.com/downloads

- **Comprobar la instalación y obtener la versión instalada:**
  - *git version*

- **Crear un nuevo repositorio local:**
  - *git init*

- **Ver el estado del repositorio:**
  - *git status*

- **Ver la rama actual donde nos encontramos y renombrar la rama:**
  - *git branch*
  - *git branch -M main*

Control de versiones

Git y GitHub (PoC)

- **Agregar los archivos sin seguimiento al “commit”:**
  - *git add .*
- **Registrar los cambios (commit) en el repositorio:**
  - *git commit -m “1.0.0: Taller Kubernetes PPS”*
- **Nos registramos en GitHub y creamos un nuevo repositorio.**
- **Asignar el repositorio remoto de GitHub al repositorio Git local:**
  - *git remote add origin URL_REPO.git*
- **Subir commit/s con los últimos cambios a nuestro repositorio en GitHub:**
  - *git push -u origin main*

Control de versiones

Git y GitHub (PoC)

- **Crear y cambiar de rama localmente:**
  - *git branch mi-rama*
  - *git checkout mi-rama*
- **Subir la rama nueva con los cambios:**
  - *git push --set-upstream origin mi-rama*
- **Descargar un repositorio remoto:**
  - *git clone URL_REPO.git*
- **Obtener los últimos cambios de un repositorio desde GitHub:**
  - *git pull*

Automatización con Jenkins

PoC

- **Jenkins** es un software de automatización open source que implementa procesos de integración continua CI.
- Admite la integración de herramientas CVS como Git y de construcción de software como Gradle o Ant.
- Cuenta con imagen oficial en Docker Hub.

Jenkins

UCLM
UNIVERSIDAD DE CASTILLA-LA MANCHA

Escuela Superior Ingeniería Informática

DSI
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

XUNTA DE GALICIA
CONSELLERÍA DE CULTURA, EDUCACIÓN E UNIVERSIDADE

CFR FERROL

Puesta en producción segura

- Tema 3. Implantación de sistemas seguros de despliegue de software

UNIÓN EUROPEA
Fondo Social Europeo
EL FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL