Docker

Módulo 1

Ing. Andrés S. Ruiz
asruiz.sis@gmail.com

EDUCACIÓN IT

Roadmap del curso

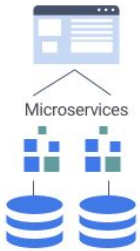
Introducción a Docker
Imágenes y contenedores

Docker
volúmenes y redes

Docker
docker-compose

Introducción a Kubernetes
Conceptos y arquitectura

Kubernetes en la nube

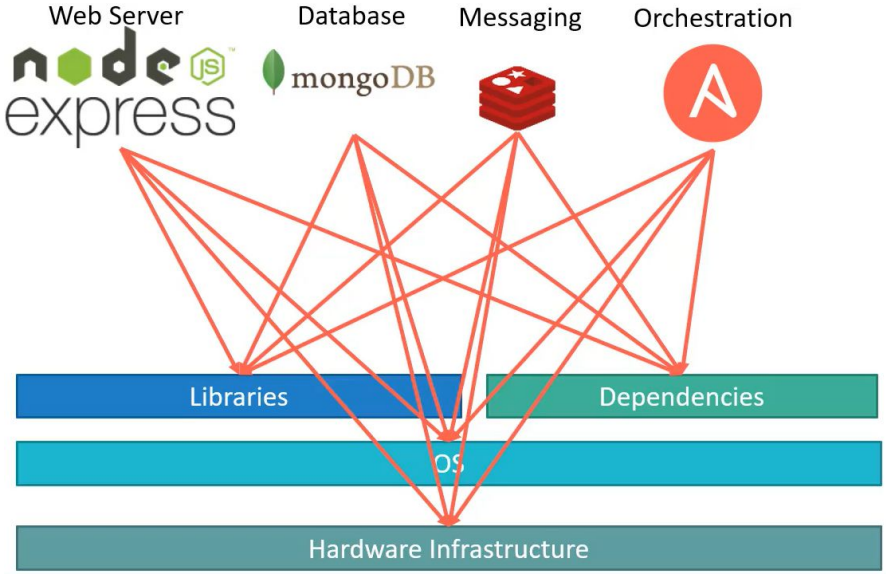


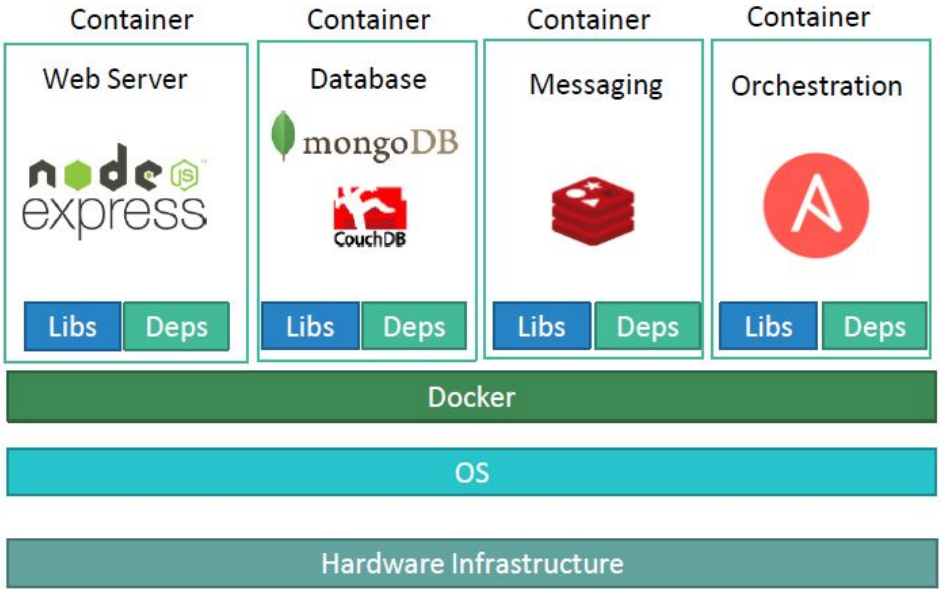
Despliegue de aplicación de ejemplo
completa con arquitectura de
microservicios en EKS

Módulo 1

- Introducción a tecnología de contenedores.
- Introducción a Docker.
- Imágenes y contenedores.
- Docker registry.

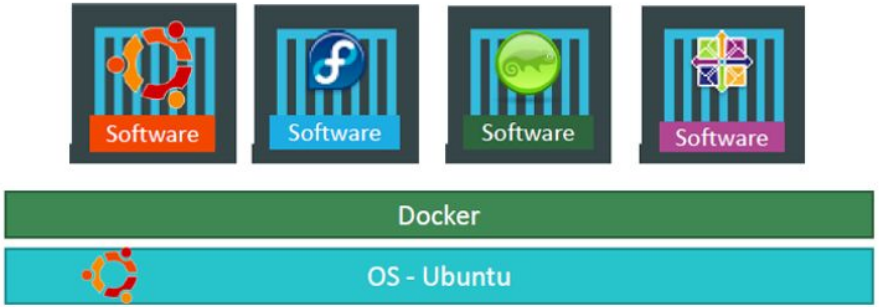
¿Para qué sirven los contenedores?



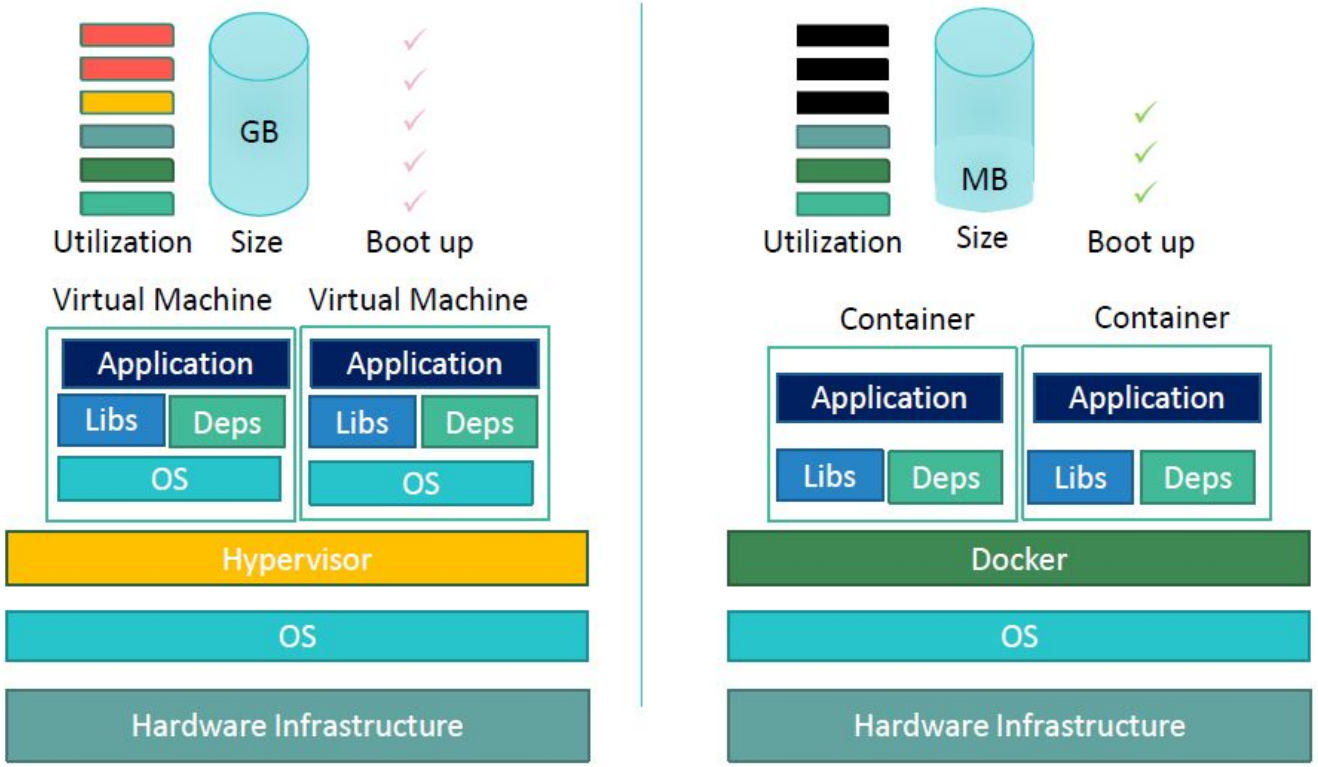


¿Qué es un contenedor?

Los contenedores son entornos completamente aislados, ya que pueden tener sus propios procesos o servicios, sus propias interfaces de red y otros recursos, al igual que las máquinas virtuales, excepto que todos los contenedores comparten el mismo núcleo del sistema operativo.



Containers vs Virtual Machines



¿Qué es docker?

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

Docker utiliza características de aislamiento de recursos del kernel Linux, tales como **cgroups** y espacios de nombres (**namespaces**) para permitir que "contenedores" independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

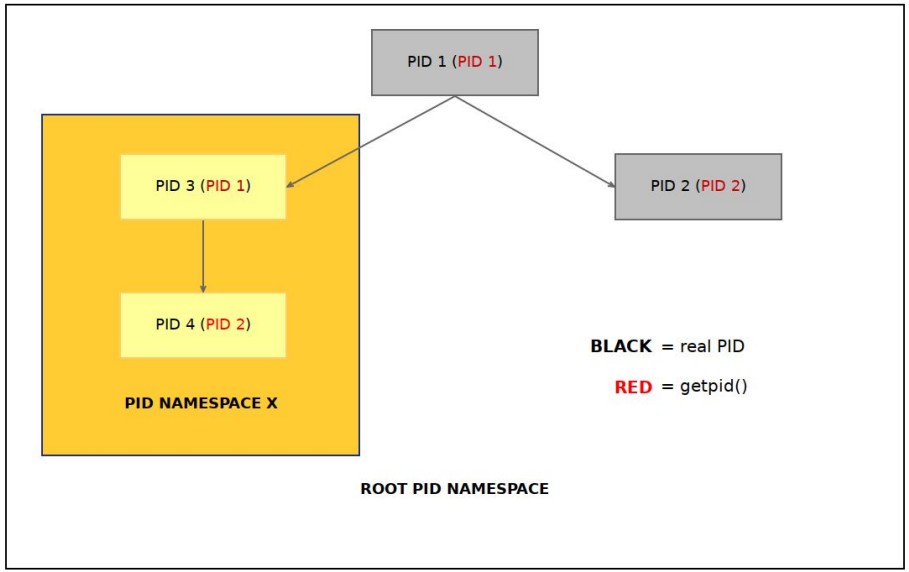
Docker es la tecnología de contenedores más popular.

Docker y el Kernel Linux

Los containers creados por Docker a partir de una imagen de Docker son aislados uno del otro y del host por diferentes características del kernel de Linux, ellas son:

- **Namespaces** → *Visibilidad.*
- **Control groups (cgroups)** → *Limitación de recursos.*
- **SElinux** → *Control de acceso.*

Namespaces



- **Comunicación entre procesos (IPC).**
- **Identificación de sistemas (UTS).**
- **Recursos de red (NET).**
- **Puntos de montaje de los sistemas de fichero (MNT)**

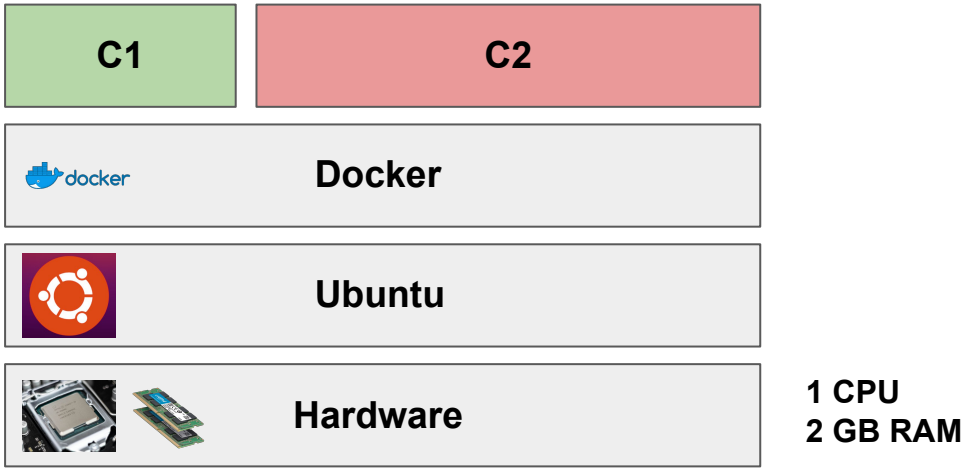
Docker y el Kernel Linux

Los containers creados por Docker a partir de una imagen de Docker son aislados uno del otro y del host por diferentes características del kernel de Linux, ellas son:

- **Namespaces** → *Visibilidad.*
- **Control groups (cgroups)** → *Limitación de recursos.*
- **SElinux** → *Control de acceso.*

CGroups

Los Cgroups limitan el acceso de procesos a la memoria, la CPU y los recursos I/O de manera que evitan que las necesidades en cuanto a recursos de un proceso concreto afecten a otros procesos en ejecución.



Docker y el Kernel Linux

Los containers creados por Docker a partir de una imagen de Docker son aislados uno del otro y del host por diferentes características del kernel de Linux, ellas son:

- **Namespaces** → *Visibilidad.*
- **Control groups (cgroups)** → *Limitación de recursos.*
- **SElinux** → *Control de acceso.*

```
[root@Rhel82 ~]# systemctl restart docker
[mcalizo@Rhel82 root]$ docker info | grep Security -A3
 Security Options:
  seccomp
   Profile: default
  selinux
[mcalizo@Rhel82 root]$
```

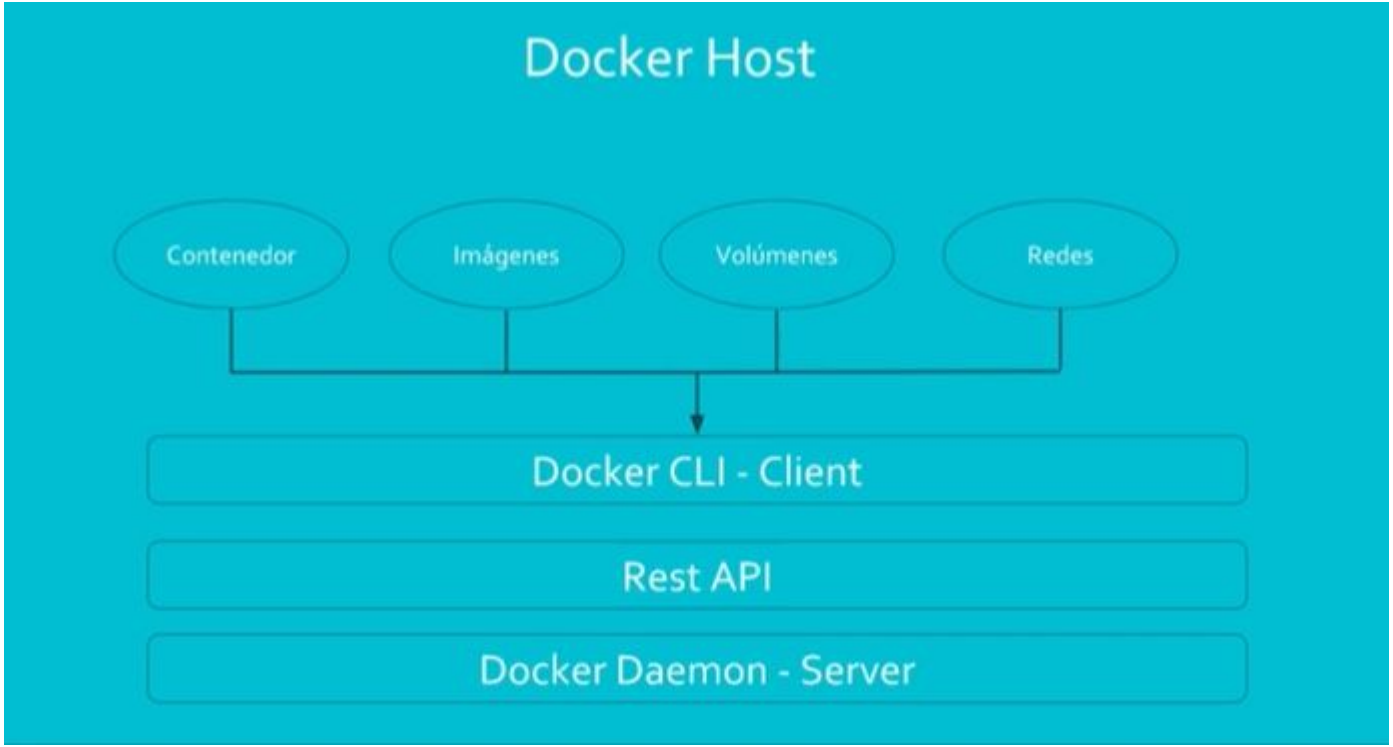
Laboratorio 1
Instalar Docker

Docker

Ubuntu

Hardware

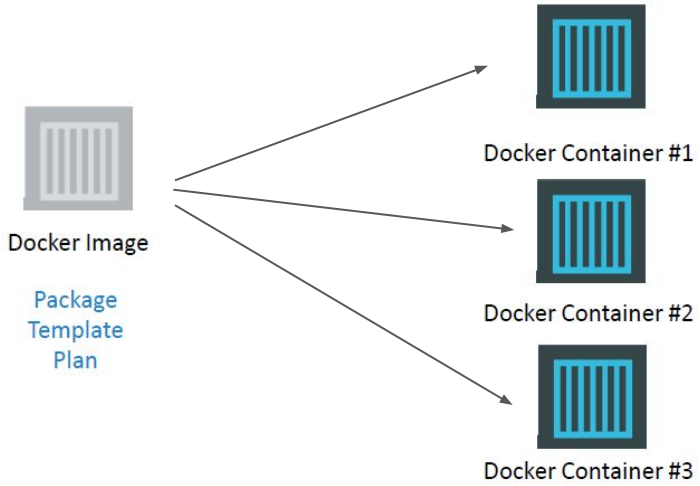
Arquitectura de Docker



Imágenes y contenedores

Una **imagen** de docker es un archivo que contiene todas las dependencias y configuraciones necesarias requeridas para correr una aplicación.

Un **contenedor** es una instancia corriendo de una imagen.



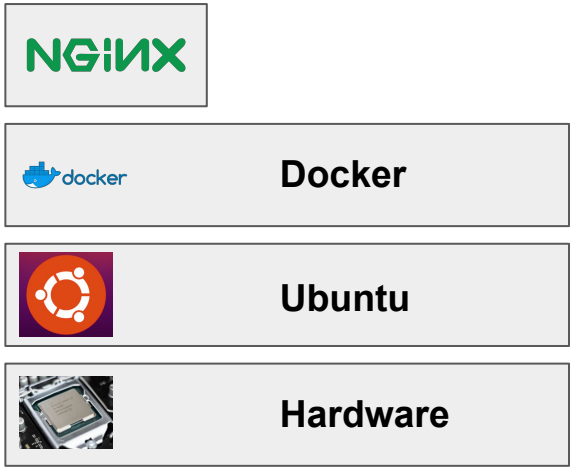
Laboratorio 2
Vemos todo junto funcionando!

Docker hub: https://hub.docker.com/

Comandos:

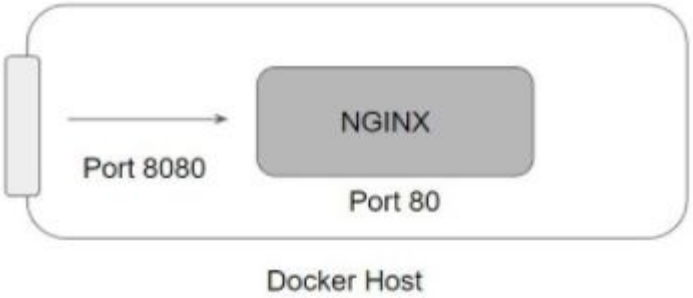
```
$docker images pull nginx
$docker images
$docker container run -dt -p 8080:80 nginx
$docker container ps
$docker container stop “ID”
$docker container ps -a
$docker container start “ID”
$netstat -nlpt
$curl -I localhost
$docker container rm -f “ID”
$docker rmi
```

Docker docs: https://docs.docker.com/



Entendiendo el flag “p”

`-p 8080:80` →



Dockerfile

- Todo container está creado a partir de una imagen.
- Las imágenes que usamos hasta ahora fueron creadas por otros y descargadas de Docker Hub.
- Docker puede construir imágenes leyendo instrucciones de un fichero llamado Dockerfile.

Laboratorio 3
Hacemos nuestra propia imagen

```
$nano Dockerfile
    FROM ubuntu #Base image
    RUN apt-get update
    RUN apt-get install -y nginx
    CMD ["nginx","-g","daemon off;"]
$docker build .
$docker run -d -p 8080:80 MyImage
```

Comandos para crear una imagen

- FROM
- RUN
- CMD
- LABEL
- EXPOSE
- ENV
- ADD
- COPY
- ENTRYPOINT
- VOLUME
- USER

Docker registry

Un registro es una aplicación del lado del servidor sin estado y altamente escalable que almacena y permite distribuir imágenes de Docker.

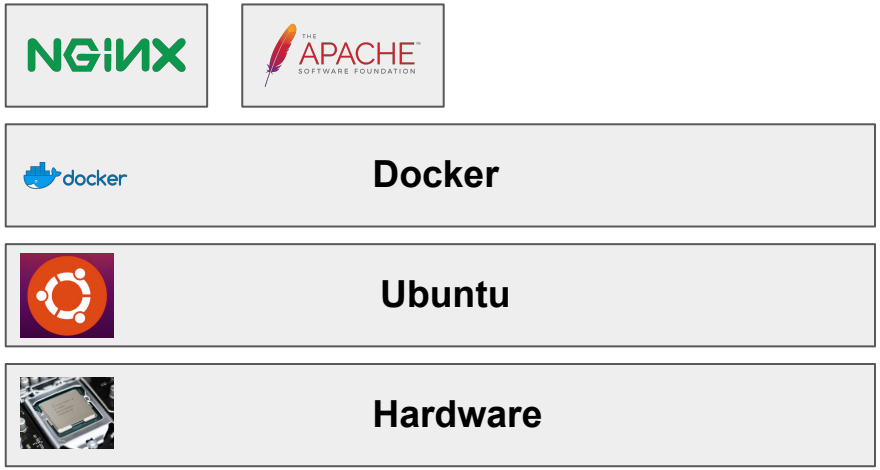
Docker Hub es el ejemplo más simple.

Hay varios tipos de registro disponibles, que incluyen:

- Docker Registry.
- Docker Trusted Registry.
- Repositorio privado (AWS ECR).
- Docker Hub.

Docker container exec

El comando ***docker container exec*** permite ejecutar un comando en un container que está en ejecución.



Docker commit

Cuando hacemos cambios dentro del container puede ser útil guardar todos estos cambios en una imagen.

Por defecto, el container sobre el que hacemos el commit y sus procesos va ser pausado mientras se crea la imagen.

Mover imágenes de un host a otro

Ejemplo: James creó una imagen de Docker y la tiene en su equipo local, lo quiere enviar por correo a Matthew.

El comando `docker save` permite guardar 1 o más imágenes en un fichero .tar.

El comando `docker load` permite cargar una imagen desde un fichero .tar.

Default container command

Siempre que ejecutamos un contenedor, se ejecuta un comando predeterminado que normalmente se ejecuta como PID 1.

Este comando se puede definir mientras definimos la imagen del contenedor.

```
[root@docker-demo ~]# docker run --name mynginx -dt -p 800:80 nginx
9fba1f62038d96159630bd436532ce56105396a6465c1335e16d4d09336cd969
[root@docker-demo ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
      PORTS                   NAMES
9fba1f62038d        nginx               "nginx -g 'daemon ..."   5 seconds ago       Up 5 seconds
      0.0.0.0:800->80/tcp   mynginx
```

Políticas de reinicio

Por defecto, los contenedores no se reinician cuando el daemon de docker se reinicia.

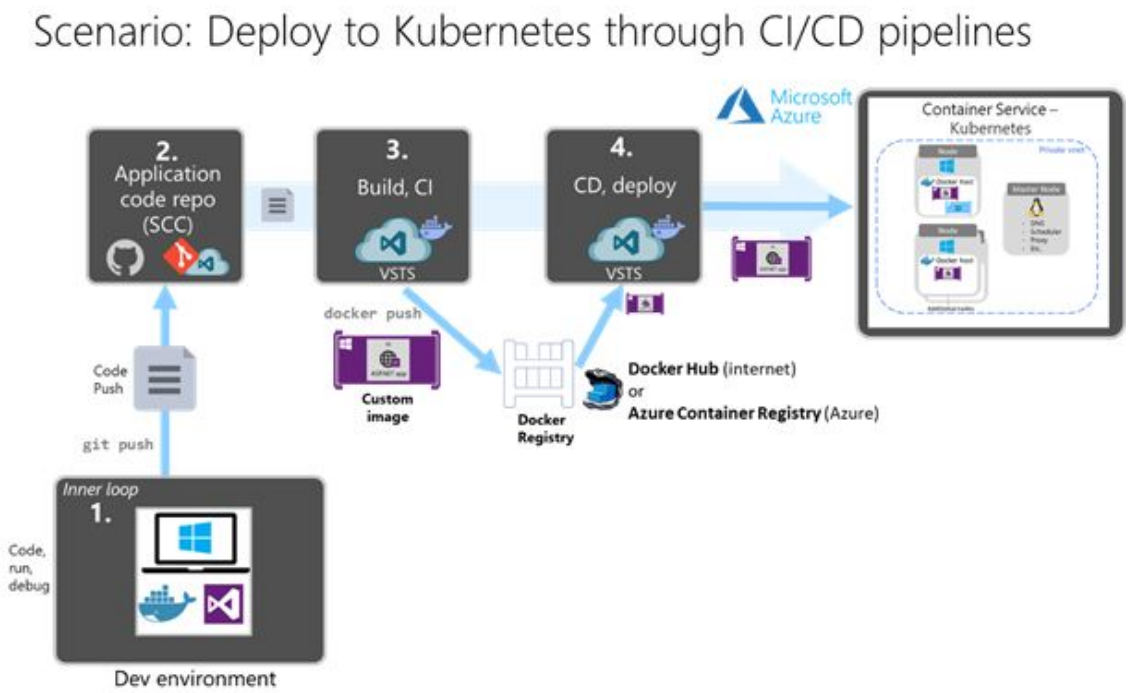
Docker provee políticas de reinicio para controlar si los contenedores se reinician automáticamente o no.

Podemos especificar la política de reinicio usando `--restart` con el comando `docker run`.

Flag	Description
`no`	Do not automatically restart the container. (the default)
`on-failure`	Restart the container if it exits due to an error, which manifests as a non-zero exit code.
`always`	Always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted. (See the second bullet listed in restart policy details)
`unless-stopped`	Similar to `always`, except that when the container is stopped (manually or otherwise), it is not restarted even after Docker daemon restarts.

Ventaja de los contenedores en el despliegue

¿DevOps?



https://docs.microsoft.com/es-es/dotnet/architecture/containerized-lifecycle/docker-devops-workflow/create-ci-cd-pipelines-azure-devops-services-aspnetcore-kubernetes

Tareas sugeridas

1. Repasar el contenido de la clase.
2. Instalar docker (Preferentemente en Linux).
3. Crearse una cuenta en Docker Hub.
4. Crear contenedores de nginx y apache.
5. Probar los comandos que vimos en la clase.

Docker

Módulo 1