

solidaritylabs.io/

/intro

Santi Abastante

- Ex Oficial de Policía
- Cloud Security Engineer
- Incident Responder

@Solidaritylabs





Introducción a Docker Contenedores

Para Ingenieros en Seguridad

/Capítulo 1

Fundamentos de Contenedores

- Cual es el problema?
- VMs y Contenedores
- Por que nos importa?
- Como se ve un contenedor?
- Que es Docker?

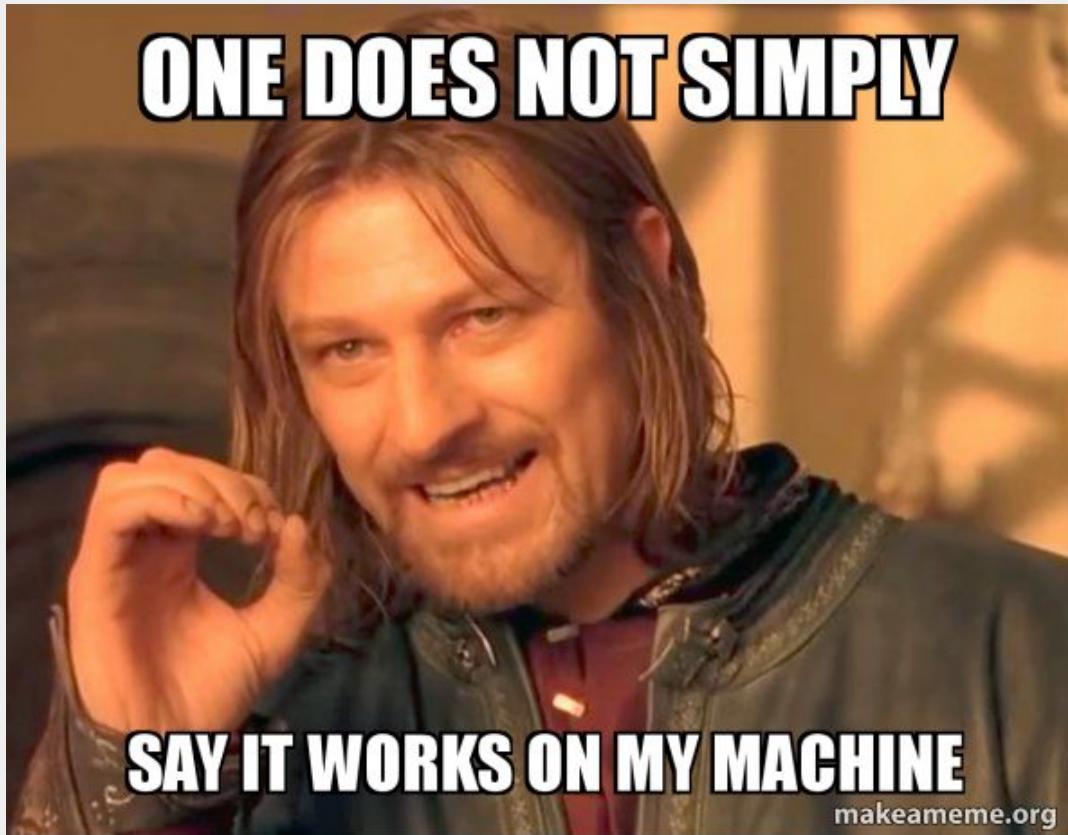
@Solidaritylabs



Cual es el Problema? (1)

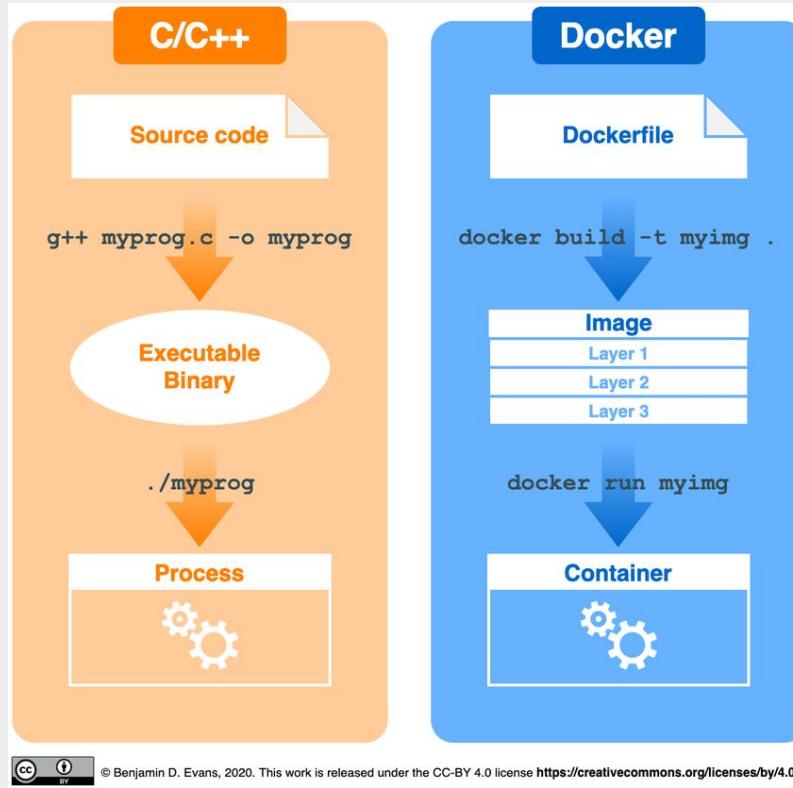


ONE DOES NOT SIMPLY



SAY IT WORKS ON MY MACHINE

Cual es el Problema? (2)



Cual es el problema? (3)



Los desarrolladores tenían contextos distintos:

- Ciclos de release mas lentos.
- Inconsistencias entre las PCs de los devs.
- Producción divergía de los ambientes de desarrollo.

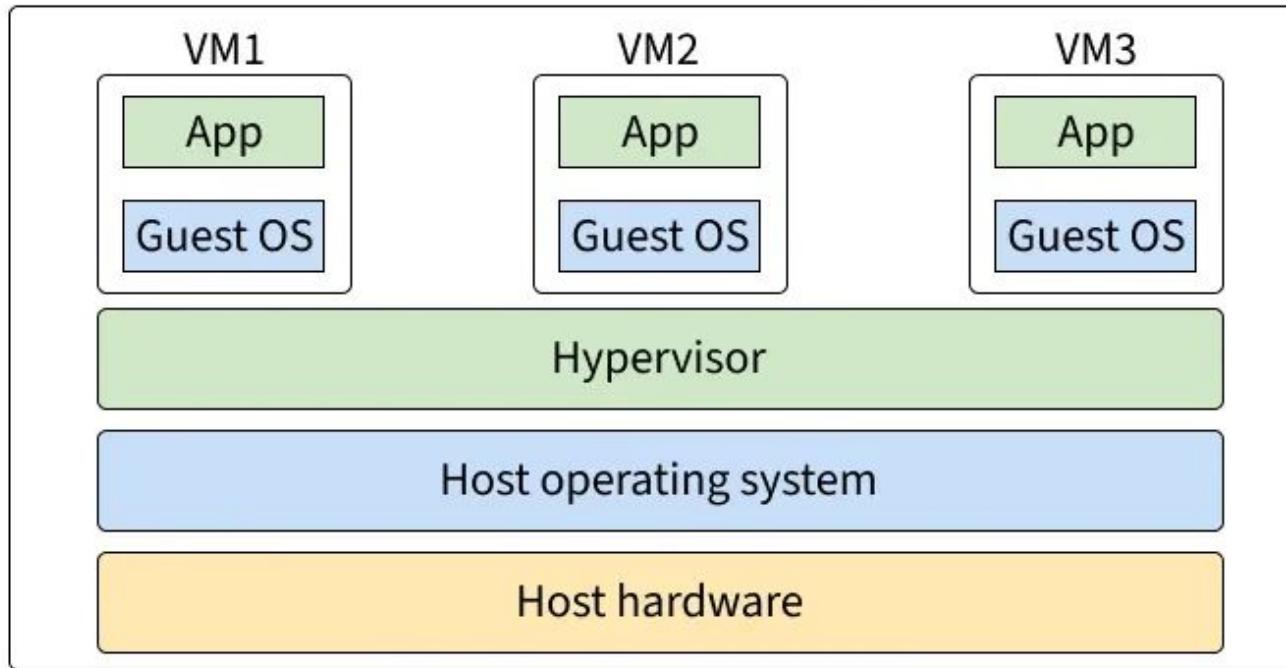
Las soluciones existentes no eran suficientes:

- Máquinas Virtuales
- Configuration Management (GIT)
- Golden Images

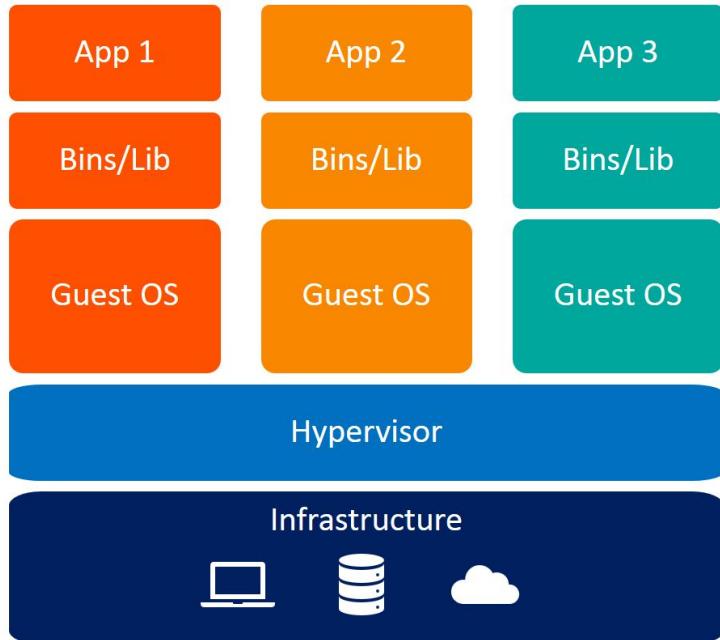
VMs y Contenedores (1)



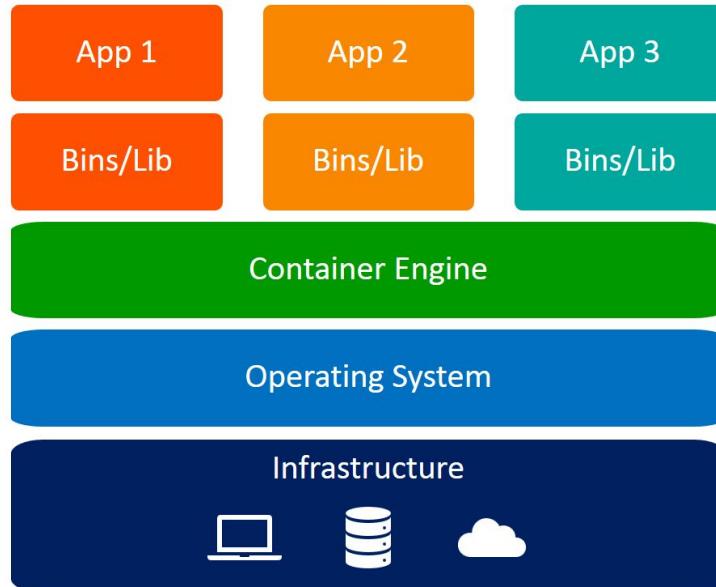
Virtual machines



VMs y Contenedores (2)



Virtual Machines



Containers

VMs y Contenedores (3)



Virtual Machine	Container
OS Completo	Comparten el Kernel del Host
Pesadas (GBs de Disco / Startup Lento)	Ligeras (MB, Startup en ms)
Cada VM tiene librerías duplicadas	Containers reutilizan llamadas de sistema compartidas
Aislamiento Fuerte	Aislamiento de procesos (No es un control de seguridad)

Por que nos importa? (1)



Devs:

- Desarrollan más rápido, hablan todos el mismo idioma
- Imágenes más livianas

Ops:

- Procesos de release consistentes.
- Software Development Workflows mas fáciles de automatizar.

Sec:

- Mayor centralización y reutilización de componentes.
- Mayor capacidad de ejecutar escaneos de Seguridad

Por que nos importa? (2)



1. **Consistencia**

- Mismo ambiente, en todos lados.
- Deployments predecibles.

2. **Velocidad**

- Los containers inician en milisegundos

3. **Portabilidad**

- Los containers corren de la misma manera en todos lados

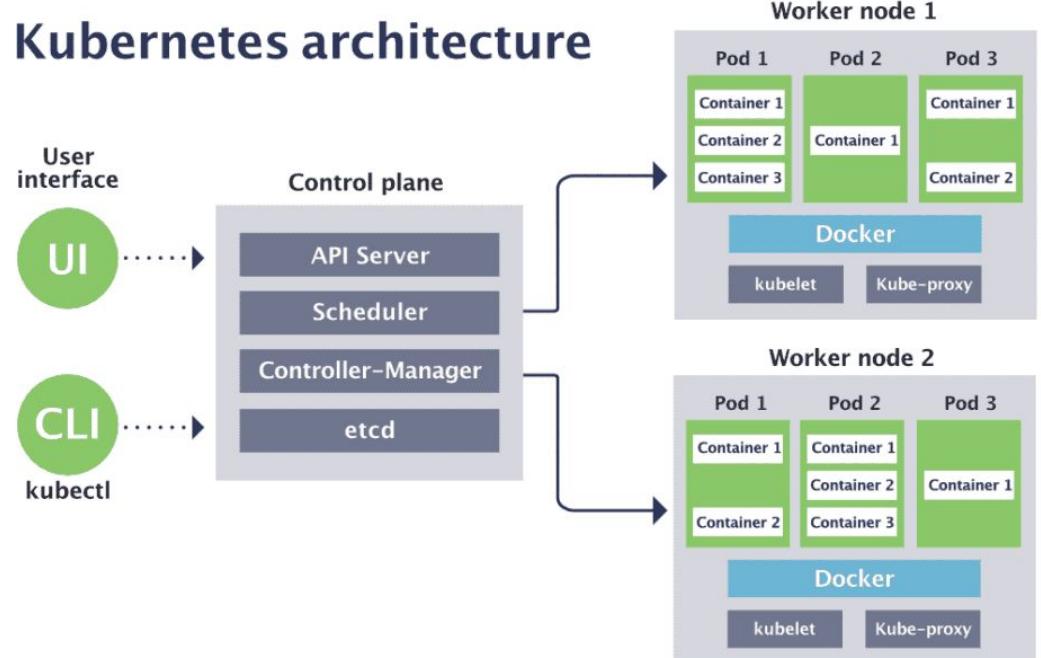
4. **Escalabilidad y arquitecturas modernas**

- Los containers son los fundamentos para Microservicios, kubernetes y serverless

Como se ve el futuro?



Kubernetes architecture



Entonces Docker?



*“Docker nos permiten **empaquetar código** y **todas sus dependencias** en una unidad llamada **IMAGEN** y correr esa imagen como un proceso aislado llamado container, garantizando **consistencia en cualquier workstation**”*

Instalar Docker

[AI](#)[Products](#)[Developers](#)[Pricing](#)[Support](#)[Blog](#)[Company](#)[Sign In](#)[Get Started](#)

Get Started with Docker

Build applications faster and more securely with Docker for developers

[Learn how to install Docker](#)[Download Docker Desktop](#)

Download for Mac – Apple
Silicon



Download for Mac – Intel
Chip



Download for Windows –
AMD64



Download for Windows –
ARM64



Download for Linux

Docker Hub

Comandos Básicos



Pullar una imagen:

- `docker pull nginx`

Correr un container

- `docker run nginx`

Listar Containers:

- `docker ps`
- `docker ps -a`

Listar Imágenes:

- `docker images`
- `docker images -a`

Parar e Iniciar

- `docker stop web`
- `docker start web`

Eliminar un container

- `docker rm nginx`
- `docker rm $(docker ps -aq)`

Eliminar Imágenes:

- `docker rmi nginx`

Purgar

- `Docker system prune`

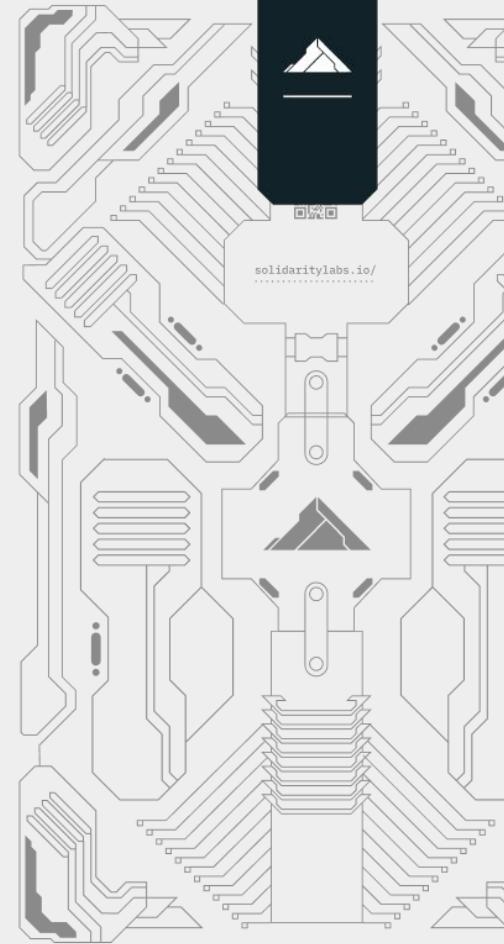
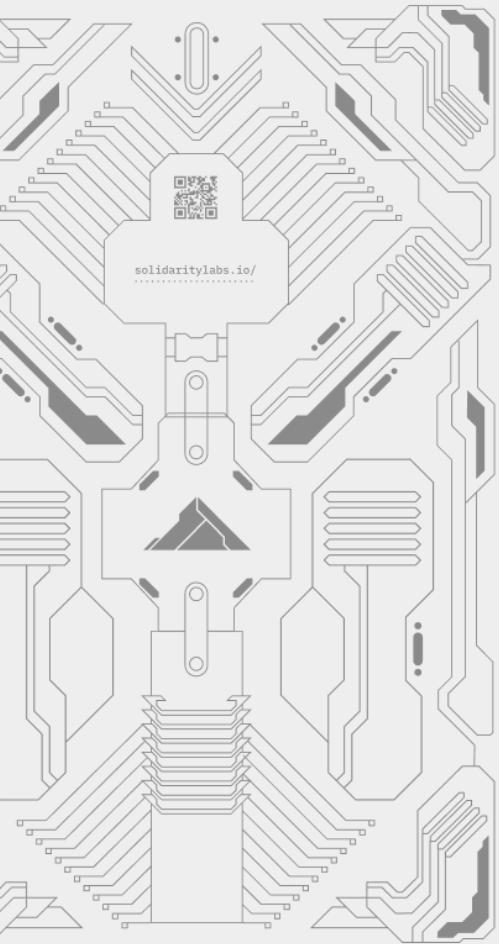
Flags (Docker RUN)



- d:** DETACHED en background | docker run -d nginx
- it:** Terminal interactiva | docker run -it alpine sh
- name:** asignar un nombre | docker run --name web nginx
- p HOST:CONTAINER:** Publicar puerto | docker run -p 8080:80 nginx
- e KEY:VALUE:** Asignar env vars | docker run -e PASS:123 nginx
- env-file FILE:** Cargar variables de entorno de archivo
- v HOST:CONTAINER:** Montar un directorio (Bind)
- rm:** remover el container automáticamente cuando termina
- entrypoint:** Pisar el entrypoint del container

Demo

Usando una APP



Ejercicio



- 1. Instalar Docker**
- 2. Bajar la imagen de NGINX**
- 3. Correr la imagen de NGINX**
- 4. Crear un index.html para el nuevo NGINX**
- 5. Correr nuevamente el NGINX con la página nueva**
- 6. Acceder a la página web creada**

/Capítulo 2

Conceptos Core

- Imagenes vs Containers
- Dockerfile
- Docker Hub / Registries
- Networking

@Solidaritylabs

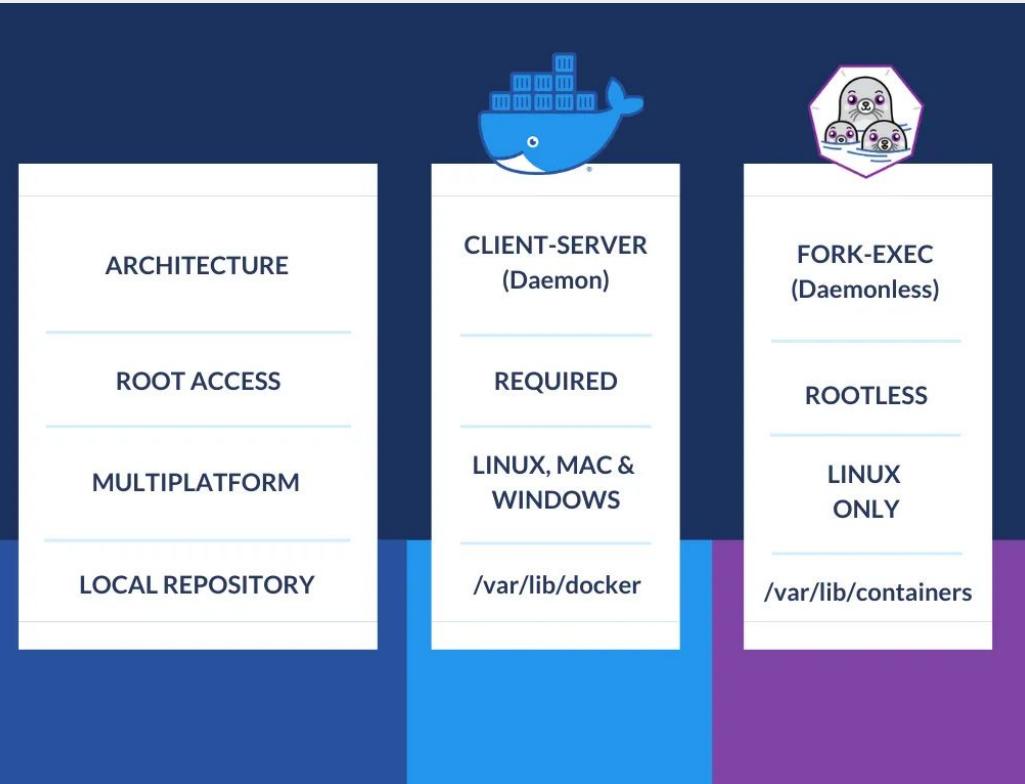


Docker vs Containers (1)



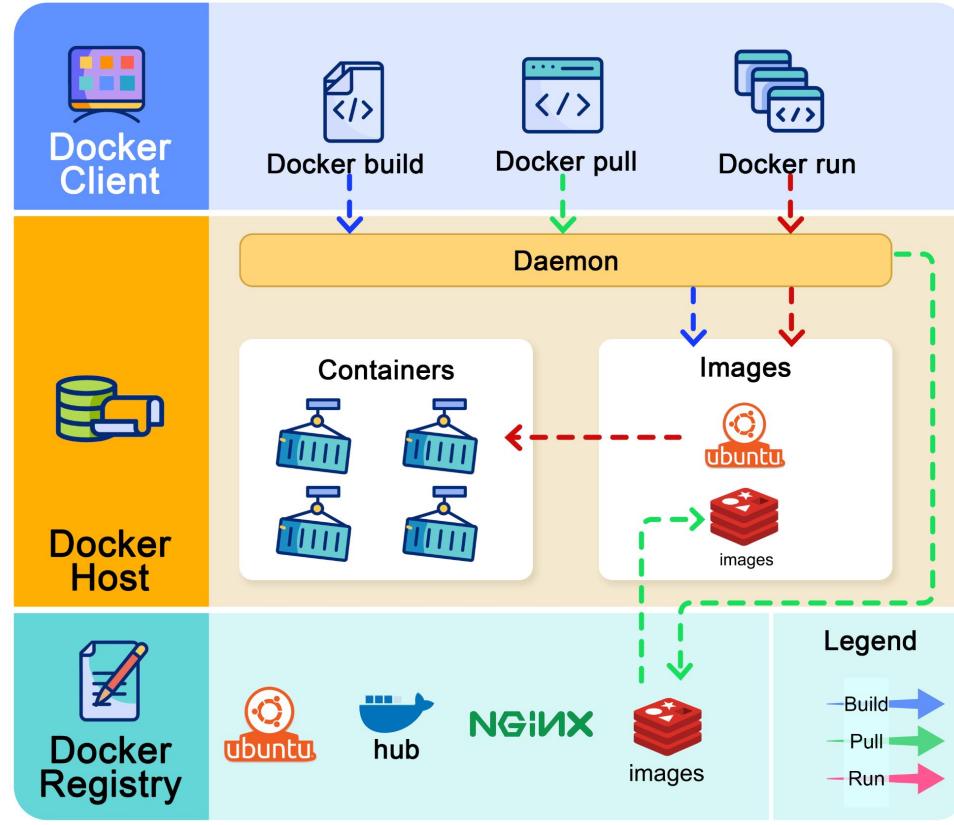
Feature	Docker	Podman	Containerd	CRI-O
Architecture	Client-server (daemon-based)	Daemonless	Low-level runtime (core)	Kubernetes-native
Security	Runs as root	Rootless, daemonless	Secure, low-level	Secure, lightweight
Kubernetes Support	Requires additional integration	Directly compatible	Native integration	Built for Kubernetes
Performance	Higher overhead due to daemon	Lightweight, better security	High performance	Lightweight, Kubernetes-optimized
Use Case	General containerization	Development, secure environments	Low-level Kubernetes and Docker integration	Kubernetes production environments
Ecosystem	Docker Hub, extensive tooling	Compatible with Docker CLI	Extensible with plugins	Optimized for Kubernetes

Docker vs Containers (2)

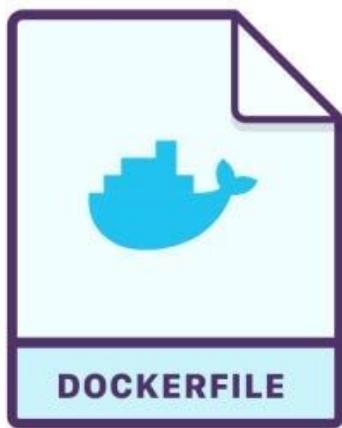


How does Docker Work ?

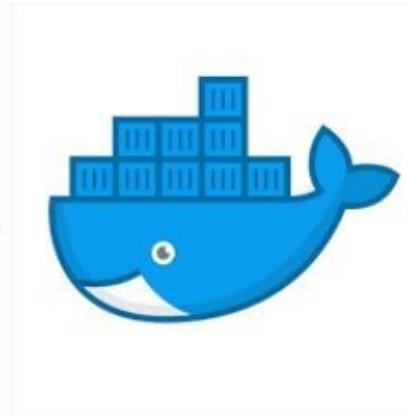
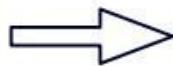
 blog.bytebytego.com



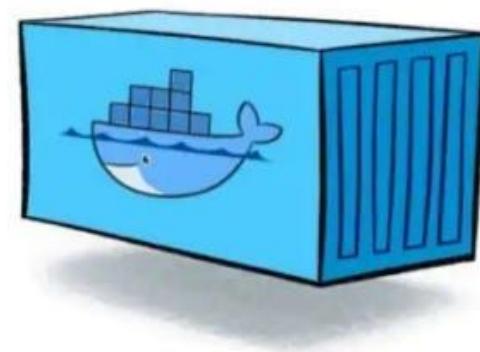
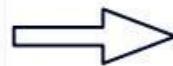
Cómo funciona? (1)



Docker file



Docker Image



Docker Container

Dockerfile - Ejemplo Mínimo



A screenshot of a code editor showing a Dockerfile. The editor has tabs for 'commands.txt' and 'Dockerfile'. The 'Dockerfile' tab is active, showing the following code:

```
conferences > docker > Dockerfile > ...
1 # Use official NGINX image as base
2 FROM nginx:latest
3
4 # Copy your website content into the container
5 COPY ./html /usr/share/nginx/html
```

Dockerfile Sintaxis



Base Image

- Necesario, cual va a ser nuestra base
`FROM ubuntu:22.04`

WORKDIR

- Donde vamos a trabajar?
`WORKDIR /app`

COPY

- Copiar Archivos
`COPY . /app`

RUN

- Correr comandos en el build
`RUN apt-get update`

ENV

- Variables de Entorno
`ENV PORT=8080`

EXPOSE

- Exponer Puertos
`EXPOSE 8080`

CMD

- Comando a ejecutar al inicio
`CMD ["python", "app.py"]`

ENTRYPOINT

- Define el ejecutable
`ENTRYPOINT ["python"]`

USER

- Define el usuario a utilizar
`USER appuser`

Dockerfile - Minimo (Prod Ready)



A screenshot of a code editor interface showing a Dockerfile. The editor has a dark theme with syntax highlighting. The Dockerfile contains six numbered commands:

```
1 FROM python:3.11-slim
2 WORKDIR /app
3 COPY requirements.txt .
4 RUN pip install --no-cache-dir -r requirements.txt
5 COPY . .
6 CMD ["python", "main.py"]
```

The file path shown in the sidebar is conferences > docker > Dockerfile > ...

Optimización



1. Cada linea genera una **LAYER**
2. El orden importa, el layering va de **ARRIBA hacia ABAJO**
3. Siempre poner **las dependencias primero**, o lo que es menos probable que cambie
4. Nunca correr el container como **ROOT**
5. Usar imágenes **mínimas** como ALPINE
6. Usar imágenes **DISTROLESS**, sin shell
7. **No instalar herramientas** que no necesitas
8. No guardar **SECRETOS** en el Dockerfile
9. Usar un filesystem de solo lectura (si es posible)
10. Pinear versiones de las imágenes

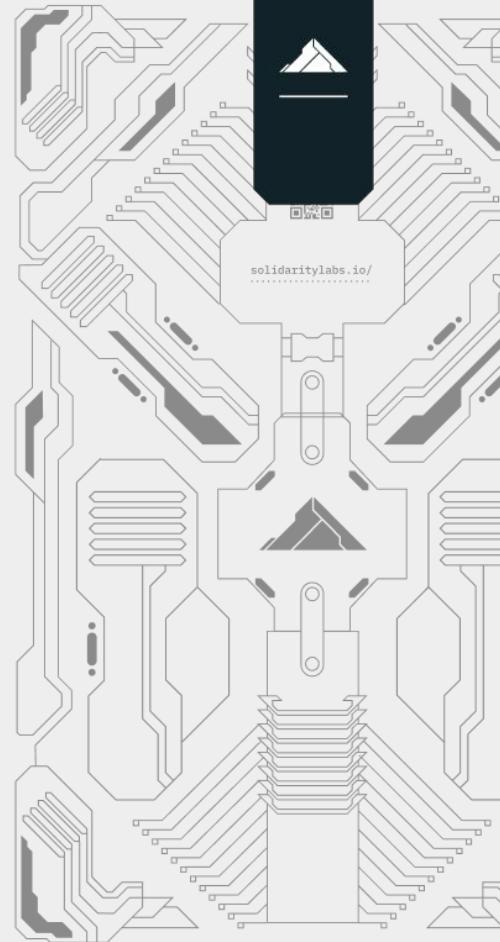
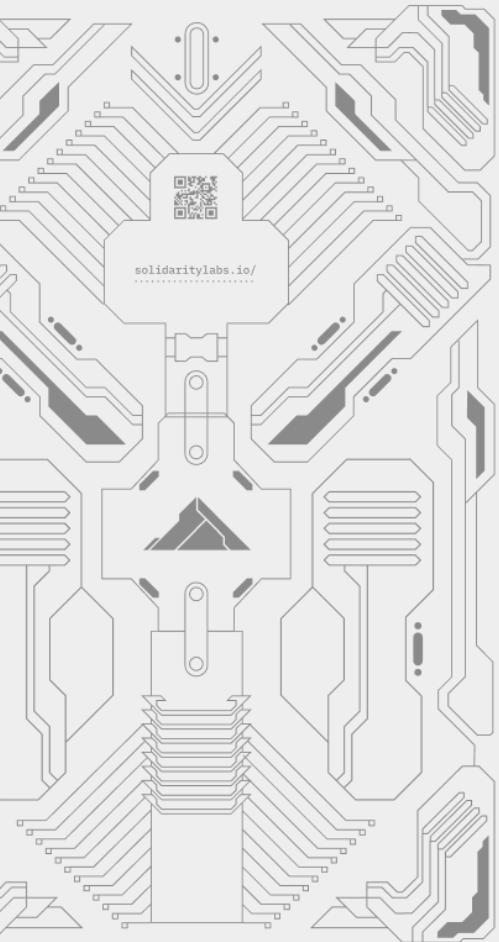
Dockerfile - Minimo y Seguro



```
conferences > docker > 📜 Dockerfile > ...
1  FROM python:3.11-slim
2
3  # Install deps
4  WORKDIR /app
5  COPY requirements.txt .
6  RUN pip install --no-cache-dir -r requirements.txt
7
8  # Add non-root user
9  RUN adduser --system --group appuser
10 USER appuser
11
12 COPY . .
13
14 # App runs with no root privileges
15 CMD ["python", "main.py"]
```

Demo

Construyendo nuestra docker
image

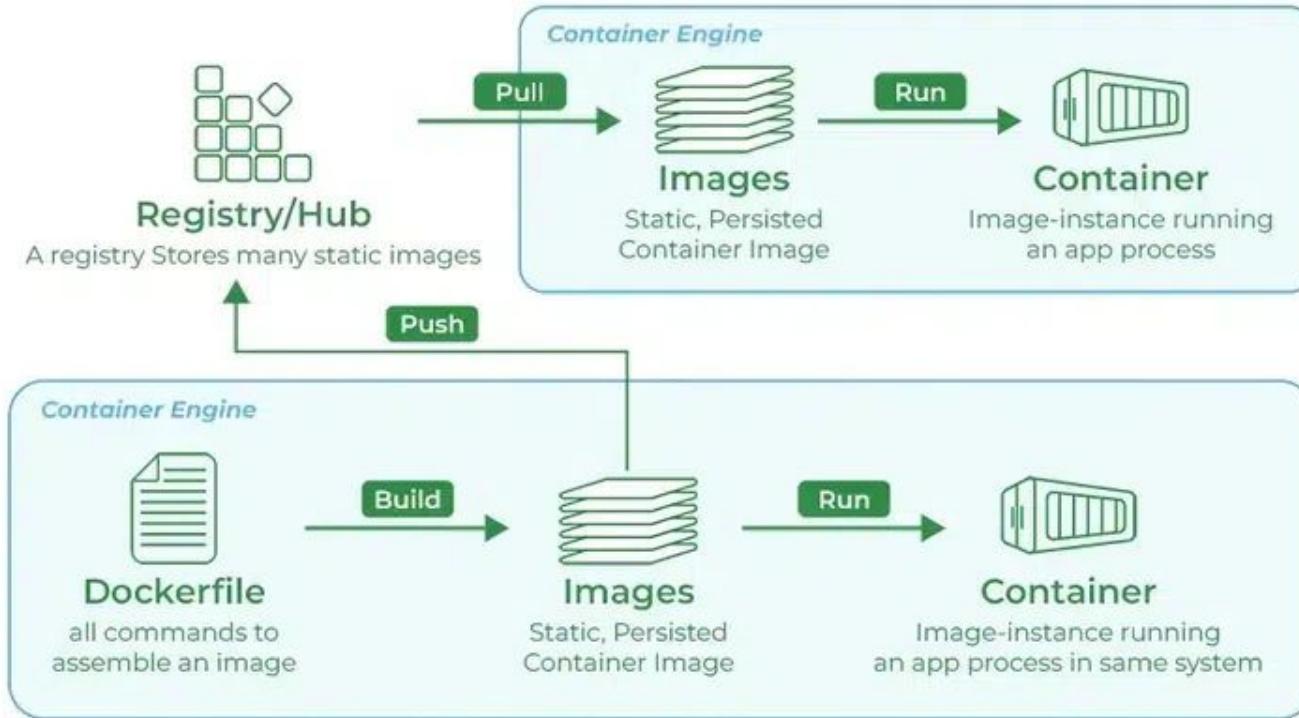


Ejercicio 2



1. Crear una API Mínima en el lenguaje que vos quieras
2. Crear un Dockerfile Seguro
3. Buildear la imagen
4. Correr la imagen creada, exponiendo la API en nuestro local.
5. Hacer un curl a la imagen para ver el resultado
6. EXTRA: Probar con tools como POSTMAN

Docker Registries (1)



Docker Registries (2)



hub.docker.com/search?q=nginx

hub

CtrlK

Filter by

1 - 30 of 272,174 results for nginx.

Products

- Images
- Extensions
- Plugins
- Compose
- AI Models

Trusted content

- Docker Official Image
- Verified Publisher
- Sponsored OSS

Categories

- Networking
- Security
- Languages & frameworks

IMAGE + 1 MORE

 nginx Docker Official Images

Official build of Nginx.

Pulls Stars Last Updated

1B+ 21061 5 days

IMAGE

 nginx/nginx-ingress NGINX Inc.

NGINX and NGINX Plus Ingress Controllers for Kubernetes

Pulls Stars Last Updated

1B+ 110 23 minutes

IMAGE

 nginx/nginx-prometheus-exporter NGINX Inc.

NGINX Prometheus Exporter for Nginx

Pulls Stars Last Updated

IMAGE

 nginx/unit NGINX Inc.

This repository is retired, use the Docker official images: https://hub.docker.com/_/unit

Pulls Stars Last Updated

10M+ 66 over 2 years

IMAGE

 nginx/nginx-ingress-operator NGINX Inc.

NGINX Ingress Operator for NGINX and NGINX Plus Ingress Controllers. Based on the Helm chart for NGI

Pulls Stars Last Updated

1M+ 2 14 days

IMAGE

 nginx/docker-extension NGINX Inc.

Pulls Stars Last Updated

IMAGE

 nginx

IMAGE

 nginx

IMAGE

 nginx

Docker Registries (3)



- docker.io/library/nginx:1.25
- 123456789012.dkr.ecr.us-east-1.amazonaws.com/myapp:prod

Repositorio: `123456789012.dkr.ecr.us-east-1.amazonaws.com`

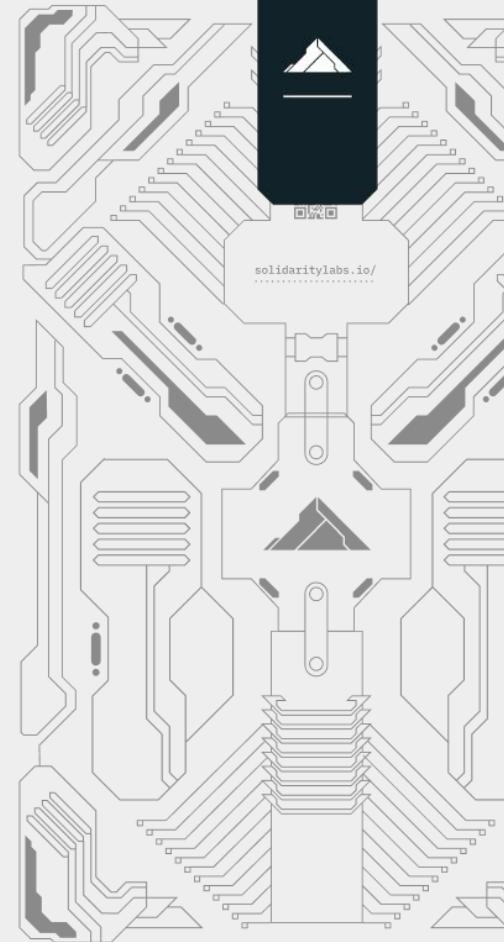
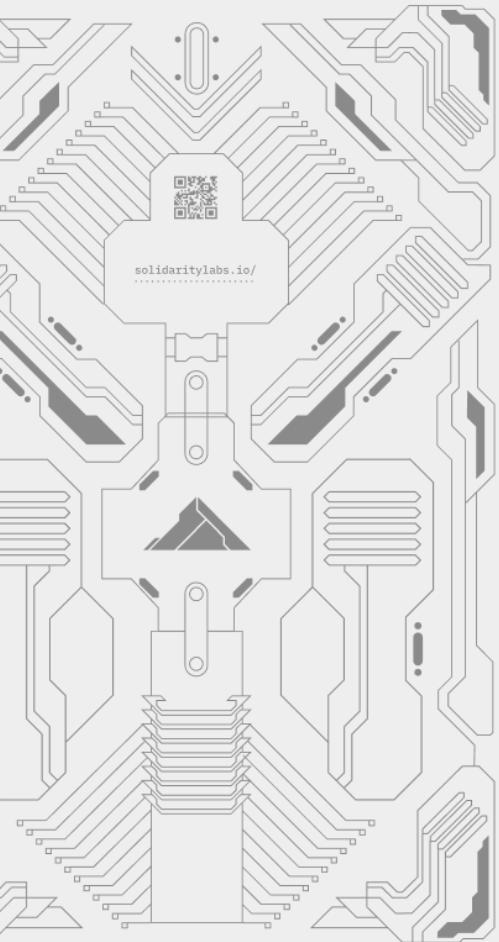
App: `myapp`

Tag: `prod`

Digest: Hash obtenido al subir la app

Demo

**Subiendo nuestra imagen a un
repositorio**



Ejercicio 3



- 1. Registrarse en DockerHub**
- 2. Logearse en la terminal a su registro**
- 3. Buildear la imagen local**
- 4. Tagear la imagen**
- 5. Subirla al DockerHub**
- 6. Compartirla con un compañer@**
- 7. Correr la imagen de un compañero**

Seguridad en Docker Registries (1)



- *En producción: Pin Digest*

```
docker run --rm youruser/ekohack-api@sha256:PUT_REAL_DIGEST_HERE
```

- *Escaneo de Imágenes*

```
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock aquasec/trivy:latest
image YOUR_DOCKERHUB_USER/ekohack-api:1.0.0
```

- *Firmar las imágenes*

```
cosign sign YOUR_DOCKERHUB_USER/ekohack-api:1.0.0
cosign verify YOUR_DOCKERHUB_USER/ekohack-api:1.0.0
```

Seguridad en Docker Registries (2)



Buenas Prácticas:

- **Usar tokens (PATs), no passwords;**
- Definir el Scope de los tokens a **read-only**;
- **Usar read-write** solo en el job de CI;
- **Nunca hardcodear tokens y passwords en imágenes**

Mantener las imágenes pequeñas y reproducibles

- Pinear las base images (no `:latest`);
- Usar slim/alpine cuando sea posible;

Troubleshooting

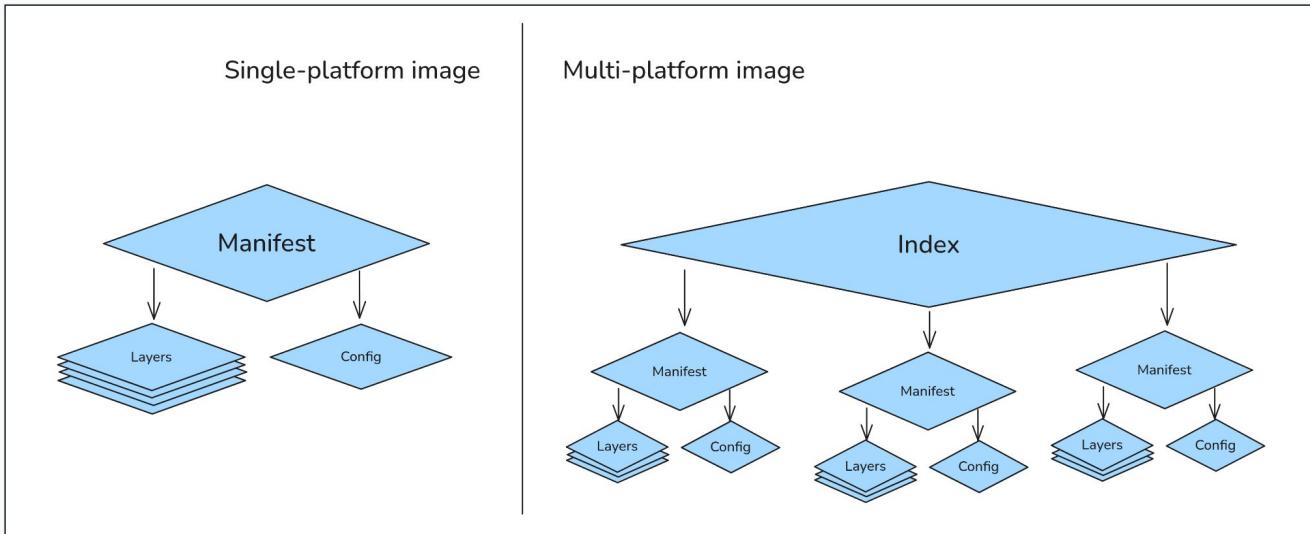


- `denied: requested access to the resource is denied`
→ Error en el nombre de repo / No estás logead@ .
- `name invalid: ...`
→ Nombre de repo invalido, quizás una mayúscula?.
- `unauthorized: authentication required`
→ Expiró o falta el Token, puede ser que no tengas suficientes permisos

Build de Múltiple Arquitectura



```
docker buildx create --use  
docker buildx build --platform linux/amd64,linux/arm64 \  
-t YOUR_DOCKERHUB_USER/ekohack-api:1.0.0 --push .
```



Ejercicio 4



1. Buscar una herramienta que uses en tu día a día
2. Ver si existe una versión “dockerizada”
3. Correr la versión dockerizada de la tool:
 - a. OpenVAS
 - b. OWASZAP
 - c. Suricata
 - d. Ollama
4. Correr un escaneo de Trivi, desde su imagen de docker a la tool que estamos usando.

Build de Arquitectura Múltiple



```
bash
```

Copy code

```
nginx:1.25
├─ linux/amd64  (normal PC/server)
├─ linux/arm64   (Apple Silicon, AWS Graviton, Pi 4)
└─ linux/arm/v7  (older Raspberry Pi)
```

```
bash
```

Copy code

```
# run an ARM64 Alpine on x86_64 host (or vice versa)
docker run --rm --platform=linux/arm64 alpine uname -m
# → aarch64

# your image example
docker run --rm --platform=linux/arm64 -p 8080:8080 docker.io/sabastante/ekohack-api:1.0.6
```

/the\ end/

Questions?

Happy to answer any questions!

Contact Info:

- LinkedIn: @sabastante
- Instagram: @santi.abastante

@Solidaritylabs

