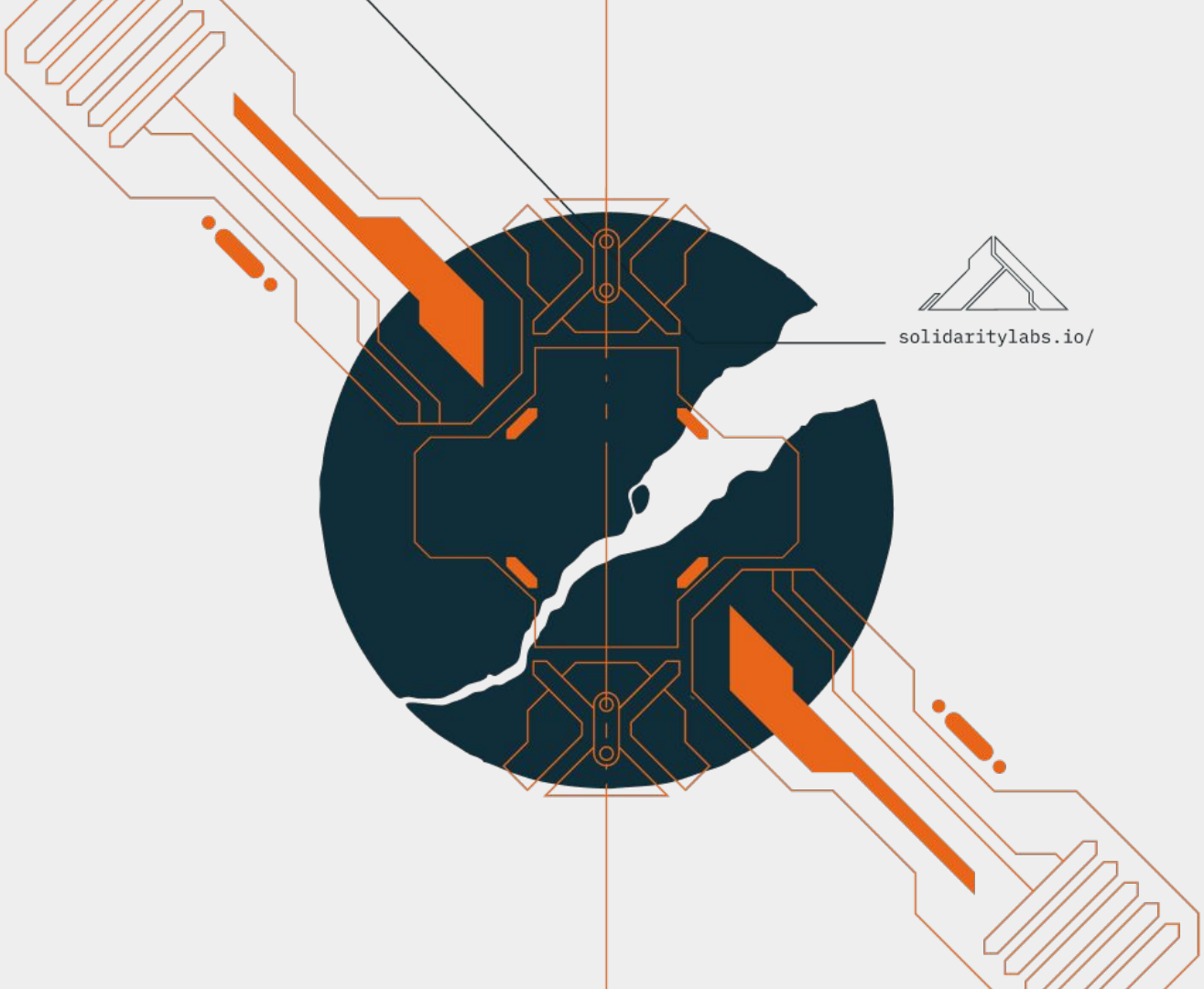
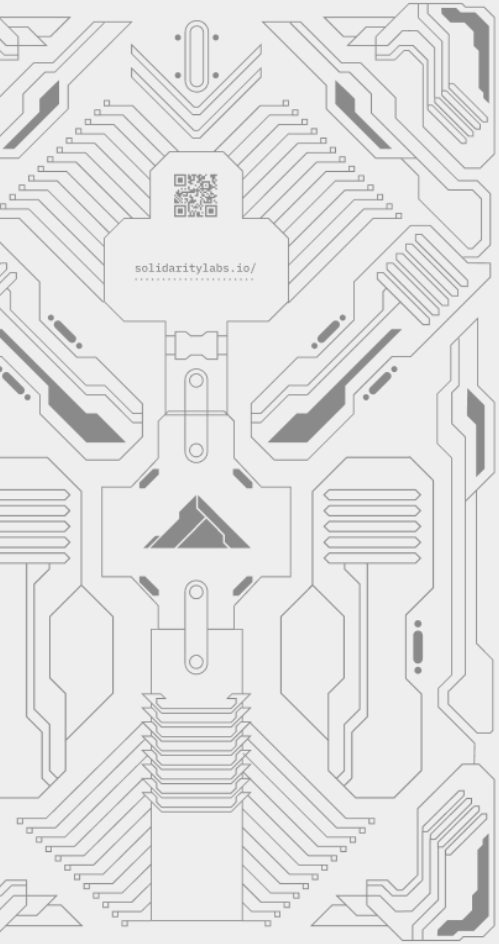
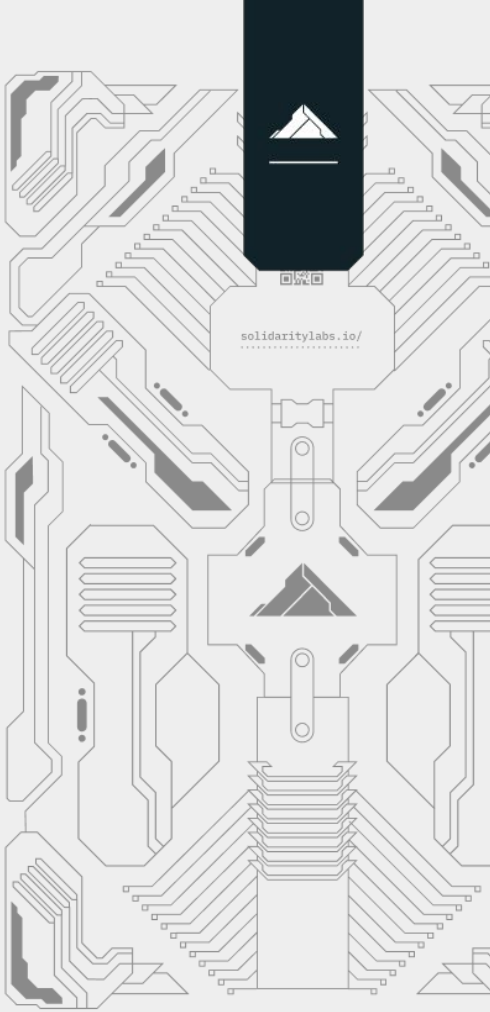

solidaritylabs.io/



Introducción a ~~Docker~~ Contenedores

Para Ingenieros en Seguridad



/intro

Santi Abastante

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

@Solidaritylabs

Objetivos del Curso

1. Entender **qué son los contenedores** y cómo se utilizan en los entornos de desarrollo modernos;
2. Aprender a **CORRER aplicaciones** dockerizadas;
3. Aprender a **CREAR aplicaciones** dockerizadas;
4. Aprender a **utilizar Registros** de Imágenes;
5. Entender cómo funcionan los **Orquestadores**;
6. Entender cómo funciona el **Networking** en entornos contenerizados;
7. Aprender a **atacar** entornos de containers;
8. Aprender a **defender** entornos de containers.

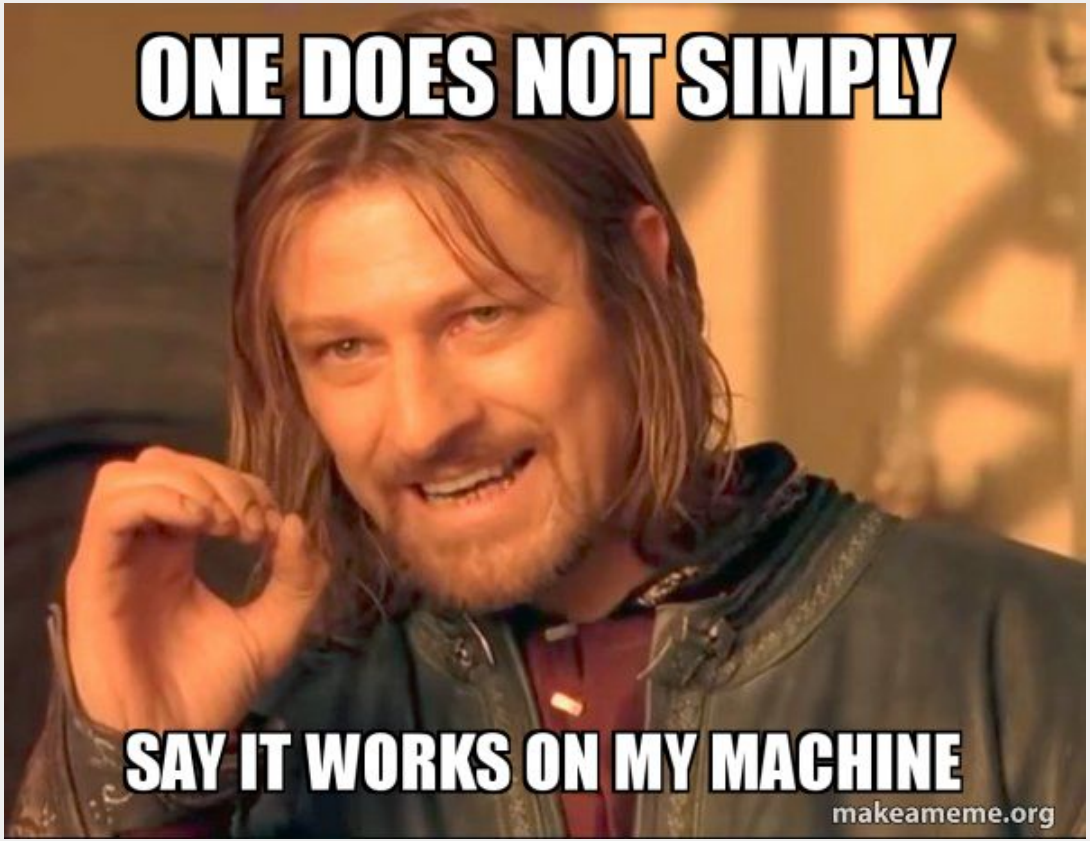
/Capítulo 1

Fundamentos de Contenedores

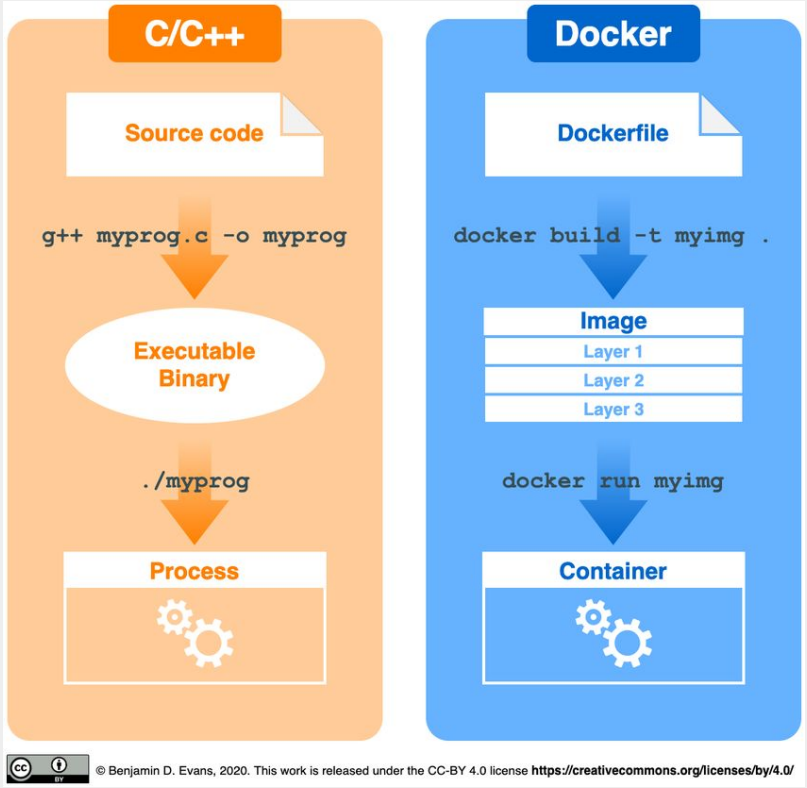
- Cual es el problema?
- VMs y Contenedores
- Por que nos importa?
- Como se ve un contenedor?
- Que es Docker?

@Solidaritylabs

Cual es el Problema? (1)



Cual es el Problema? (2)



Cual es el problema? (3)

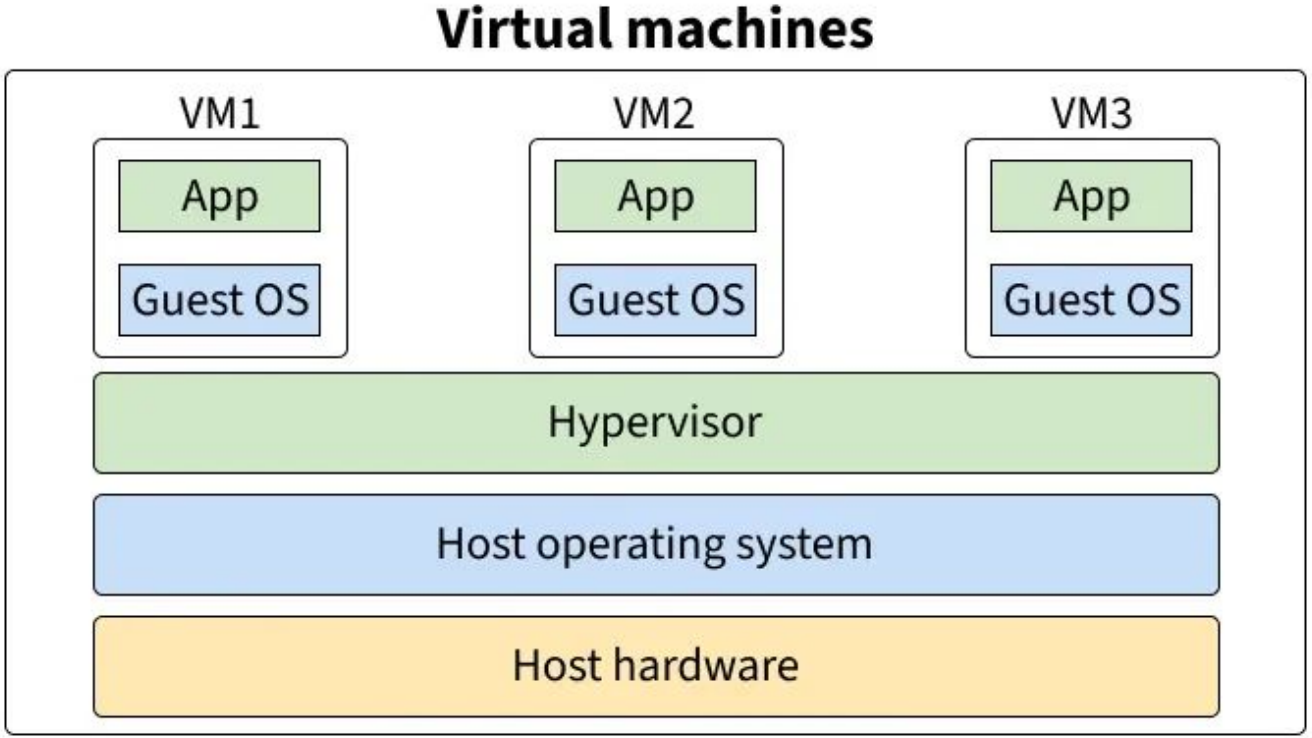
Los desarrolladores tenían contextos distintos:

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

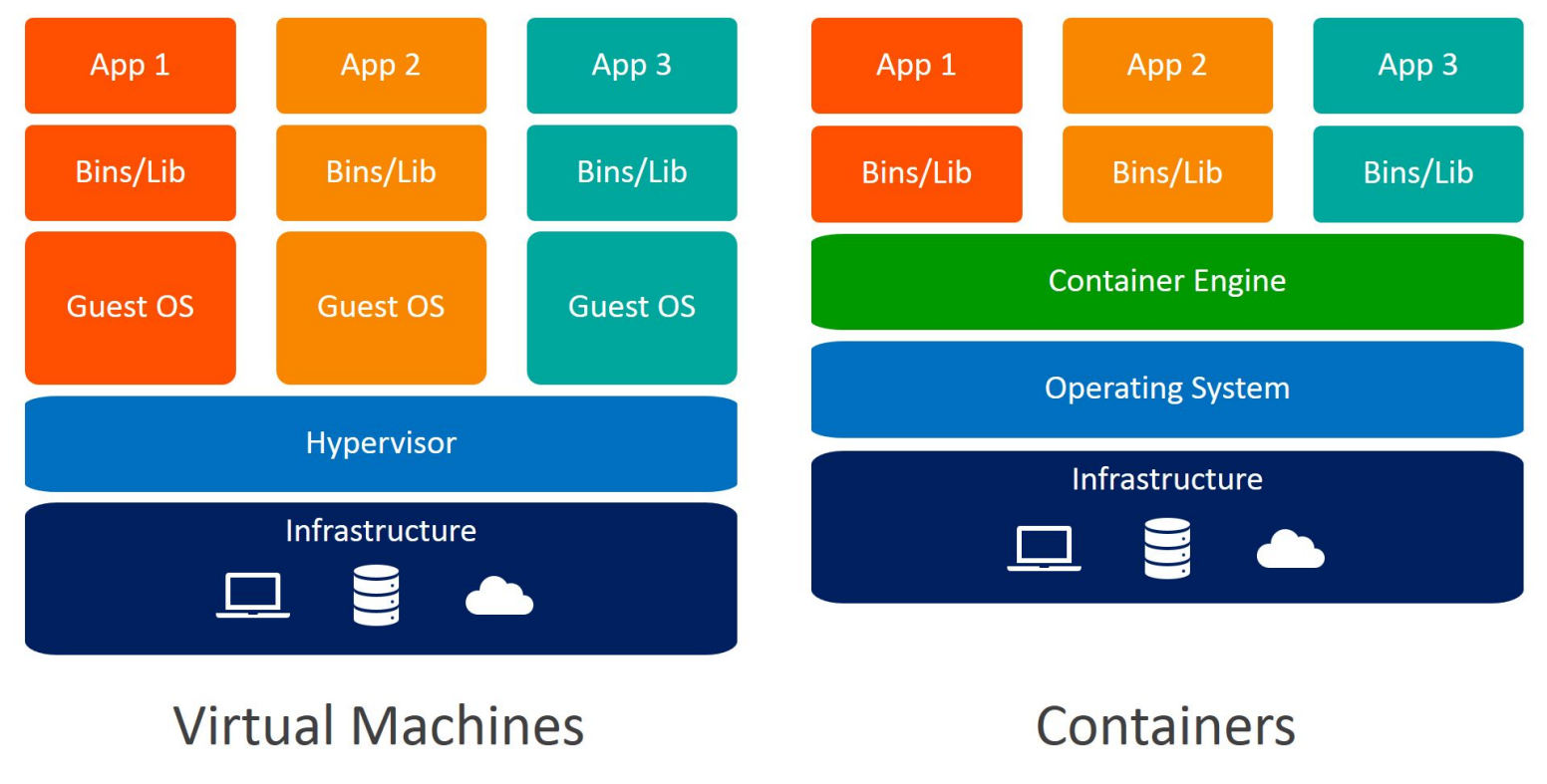
Las soluciones existentes no eran suficientes:

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

VMs y Contenedores (1)



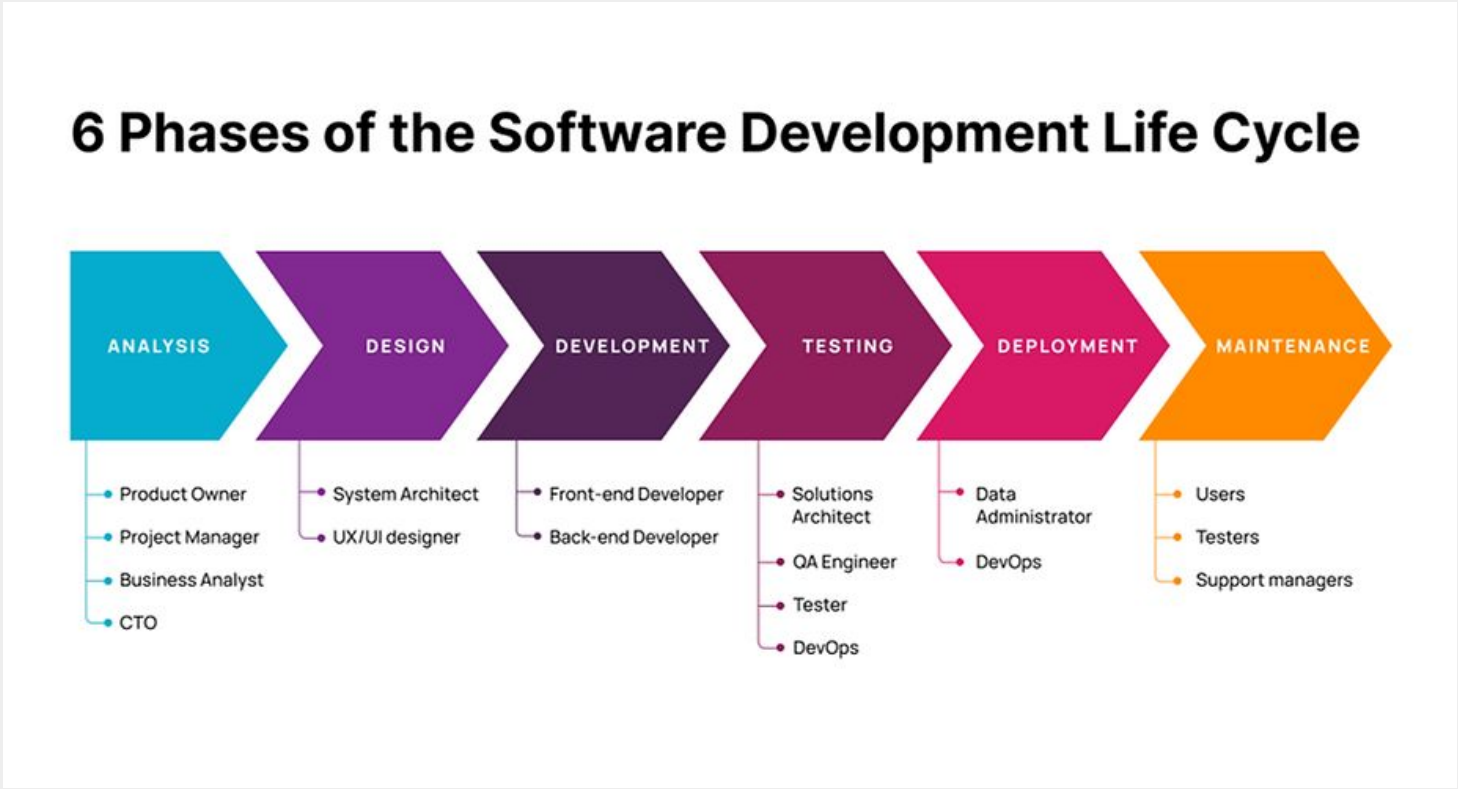
VMs y Contenedores (2)



VMs y Contenedores (3)

Virtual Machine	Container
OS Completo	Comparten el Kernel del Host
Pesadas (GBs de Disco / Startup Lento)	Ligerias (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)

Ciclo de vida de desarrollo de software



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)

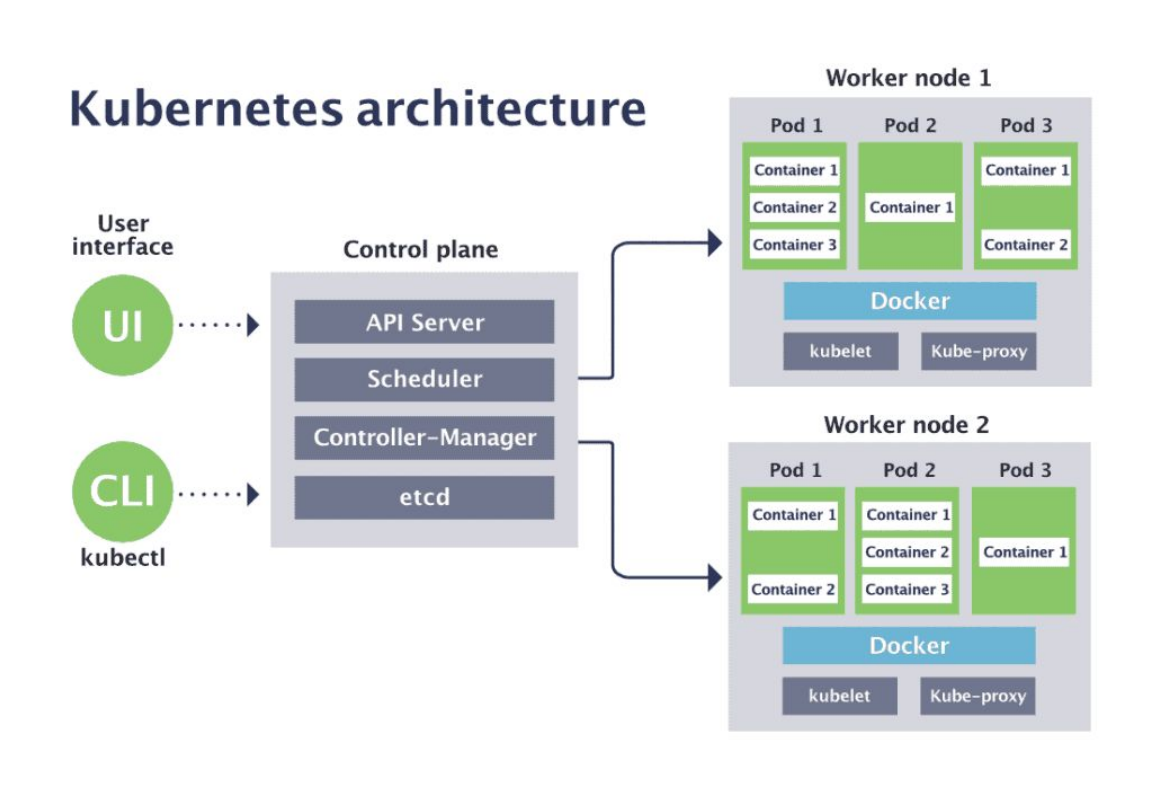
1. **Consistencia**
   - Mismo ambiente, en todos lados.
   - Deployments predecibles.

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

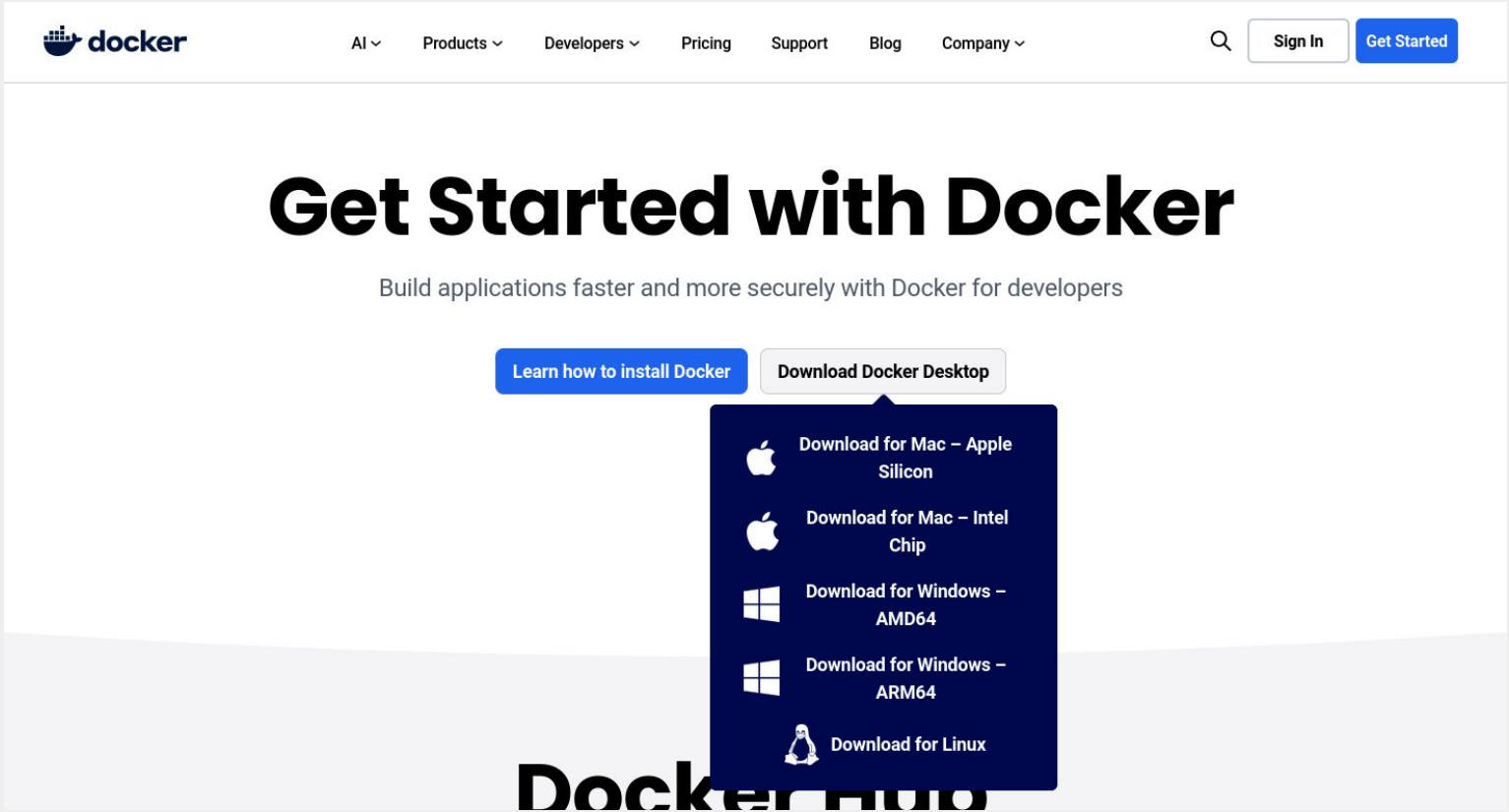
Como se ve el futuro?



Entonces Docker?

*"Docker nos permite **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



Comandos Básicos

Pulear una imagen:
- *docker pull nginx*

Correr un container
- *docker run nginx*

Listar Containers:
- *docker ps*
- *docker ps -a*

Listar Imagenes:
- *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

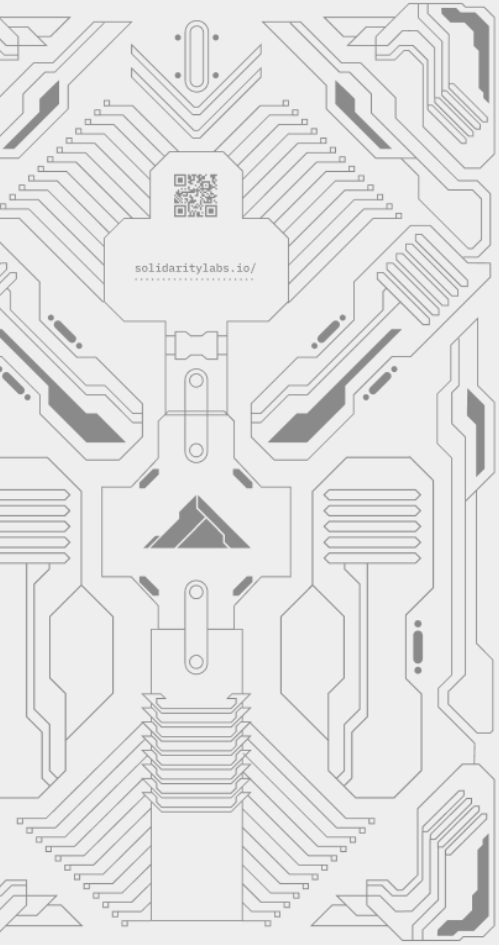
-e KEY:VALUE: Asignar env vars | docker run -e PASS:123 nginx

--env-file FILE: Cargar variables de entorno de archivo

-v HOST:CONTAINER: Montar un directorio (Bind)

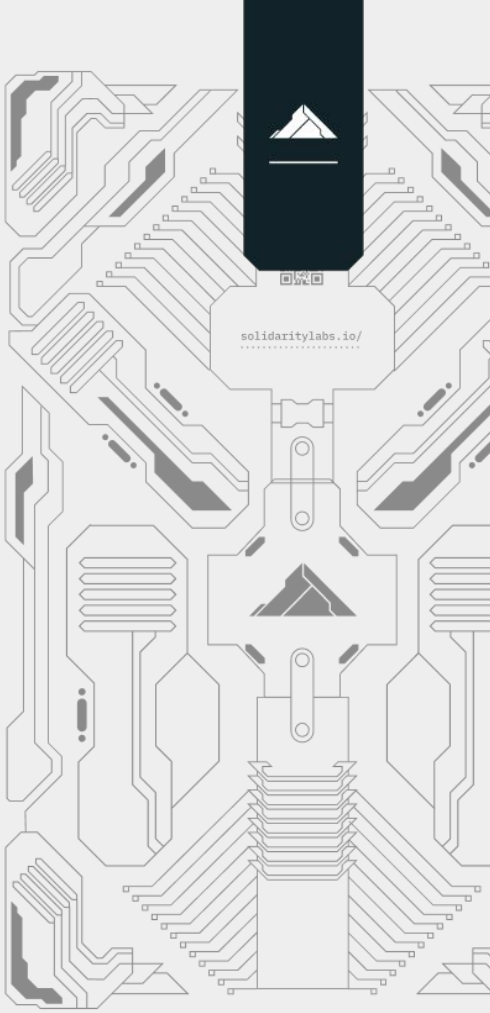
--rm: remover el container automáticamente cuando termina

--entrypoint: Pisar el entrypoint del container



Demo

Usando una APP



Ejercicio 1

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

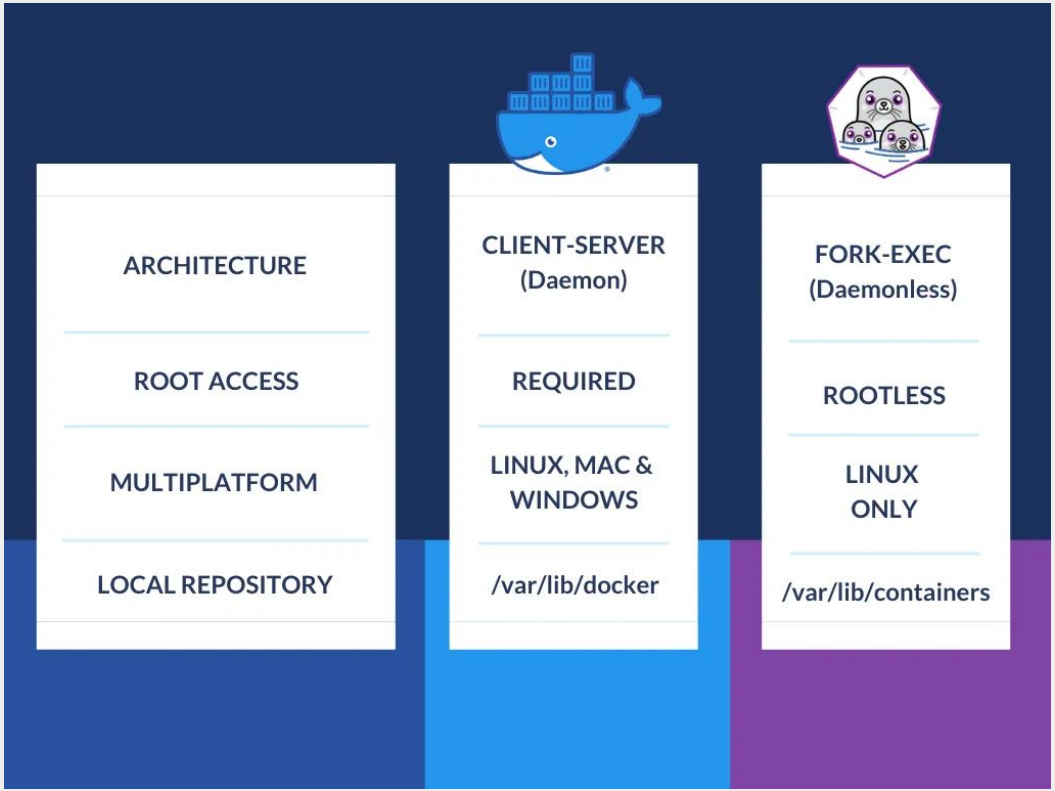
- Imagenes vs Containers
- Dockerfile
- Docker Hub / Registries
- Networking

@Solidaritylabs

Docker vs Containers (1)

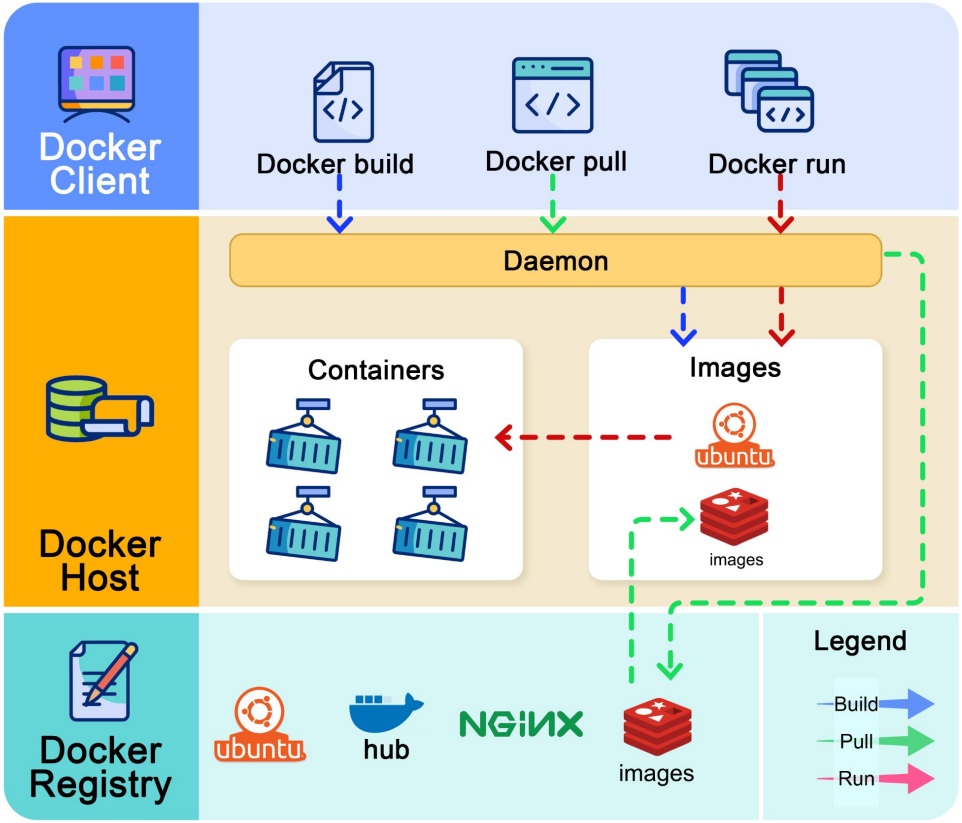
Feature	Docker	Podman	Containerd	CRI-O
Architecture	Client-server (daemon-based)	Daemonless	Low-level runtime (core)	Kubernetes-native
Security	Runs as root	Rootless, daemonless	Secure, low-level	Secure, lightweight
Kubernetes Support	Requires additional integration	Directly compatible	Native integration	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)

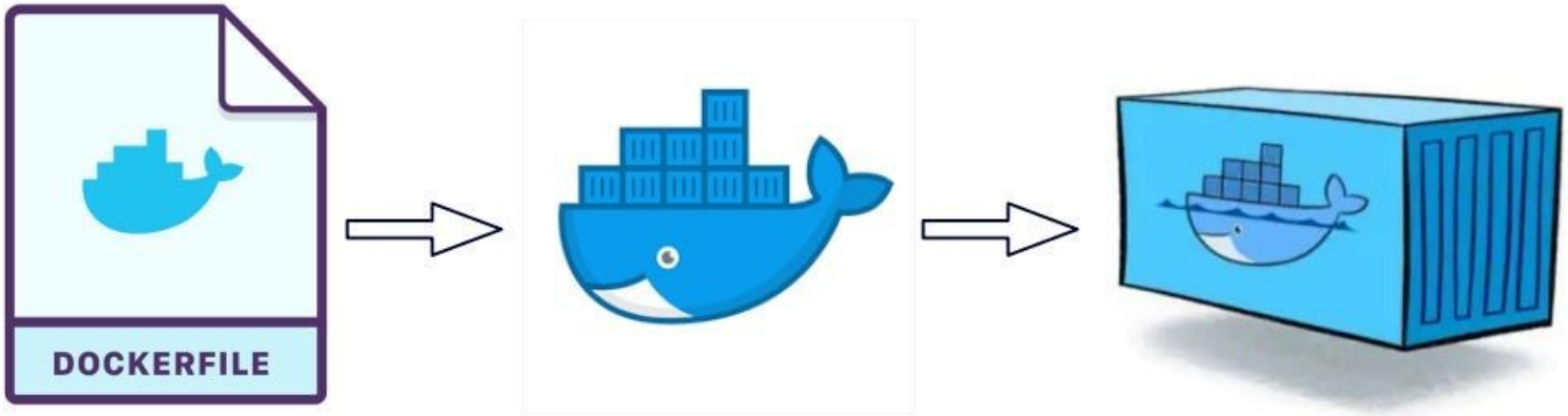


How does Docker Work ?

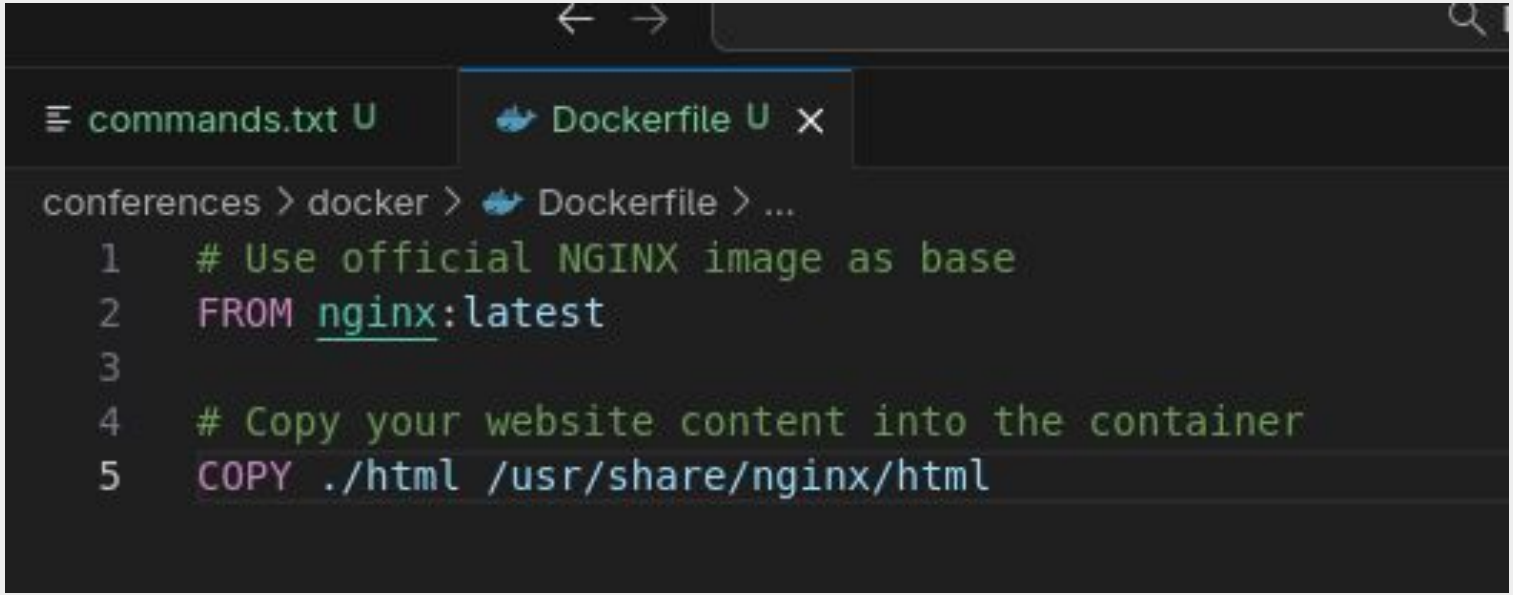
blog.bytebytego.com



Cómo funciona? (1)



Dockerfile - Ejemplo Mínimo



Dockerfile Syntax

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`

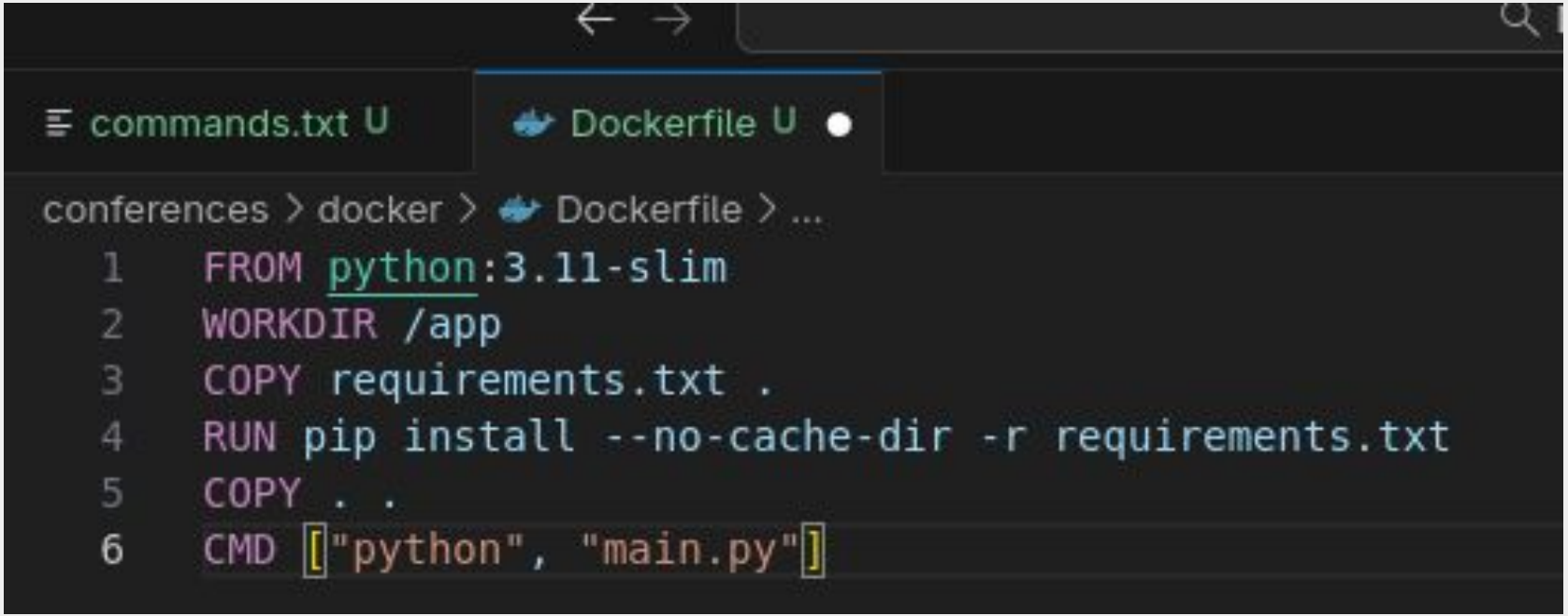
EXPOSE
- *Exponer Puertos*
  `EXPOSE 8080`

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)



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 imagenes **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 - Mínimo 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"]
```

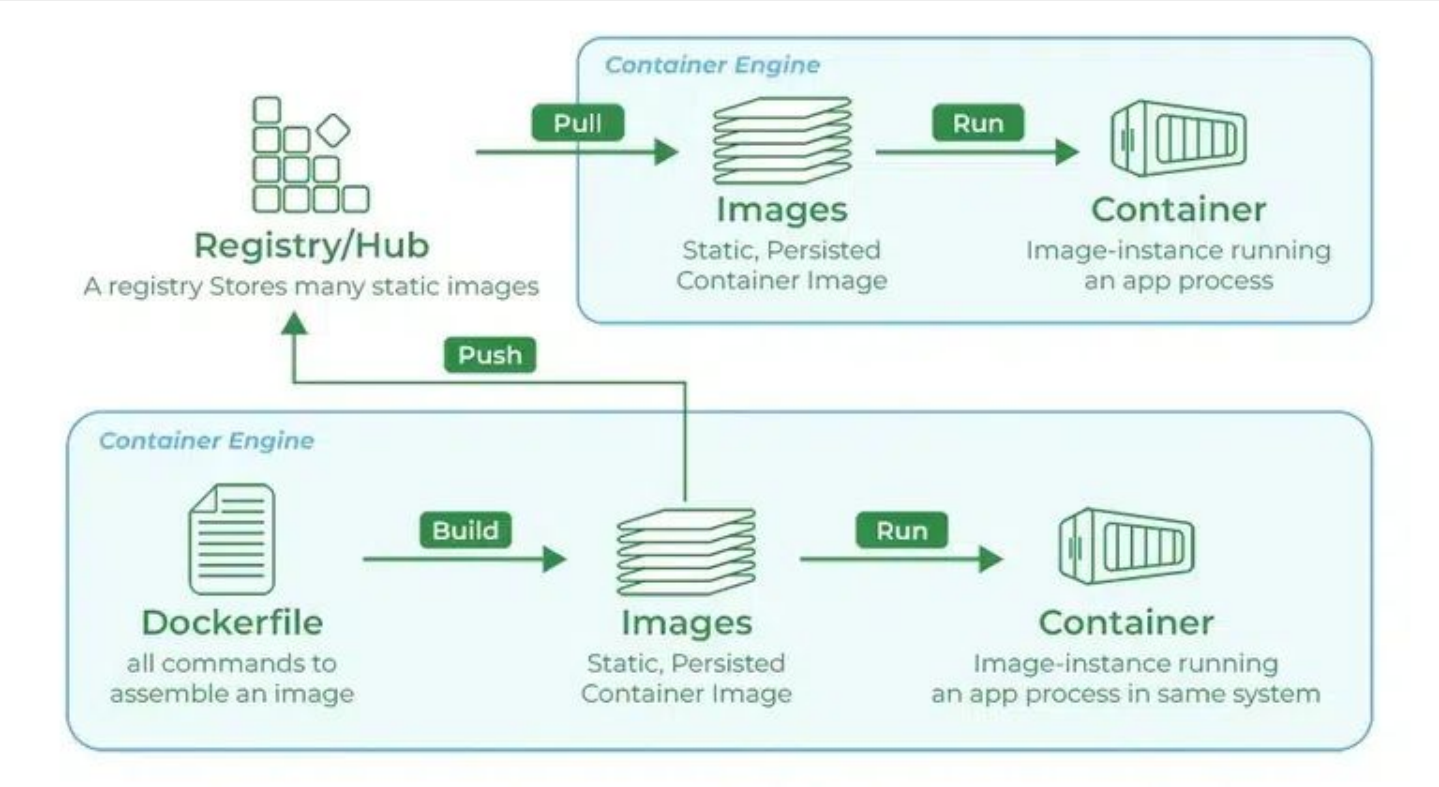
Demo

Construyendo nuestra docker image

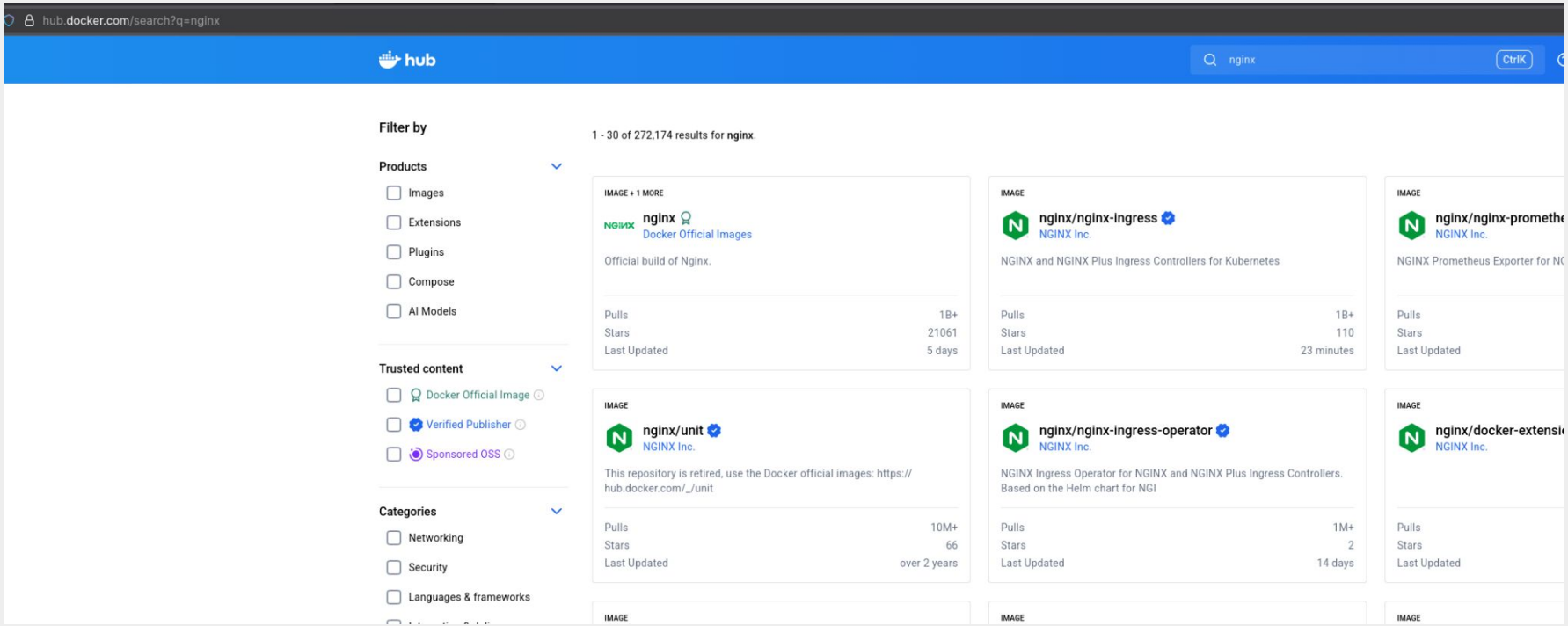
Ejercicio 2 - Docker Images

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)



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 - Docker Registries

1. Registrarse en Docker Hub
2. Loguearse en la terminal a su registro
3. Buildear la imagen local
4. Tagear la imagen
5. Subirla al Docker Hub
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 imagenes**

Mantener las imágenes pequeñas y reproducibles

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

Troubleshooting

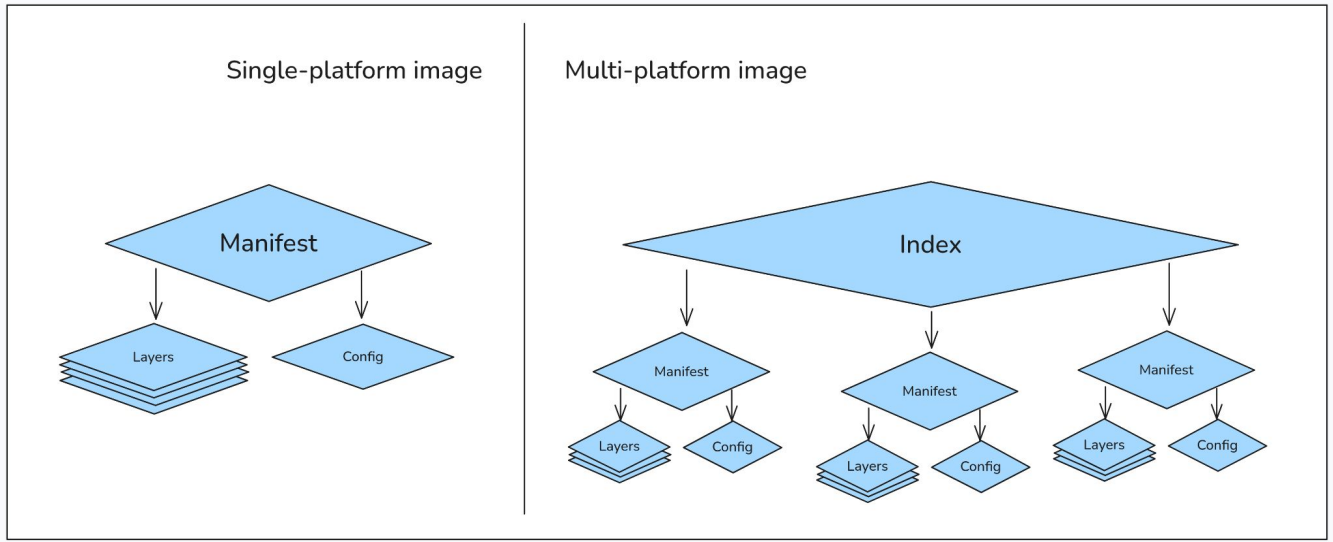
- `denied: requested access to the resource is denied`
  → Error en el nombre de repo / No estás logead@ .

- `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 .



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

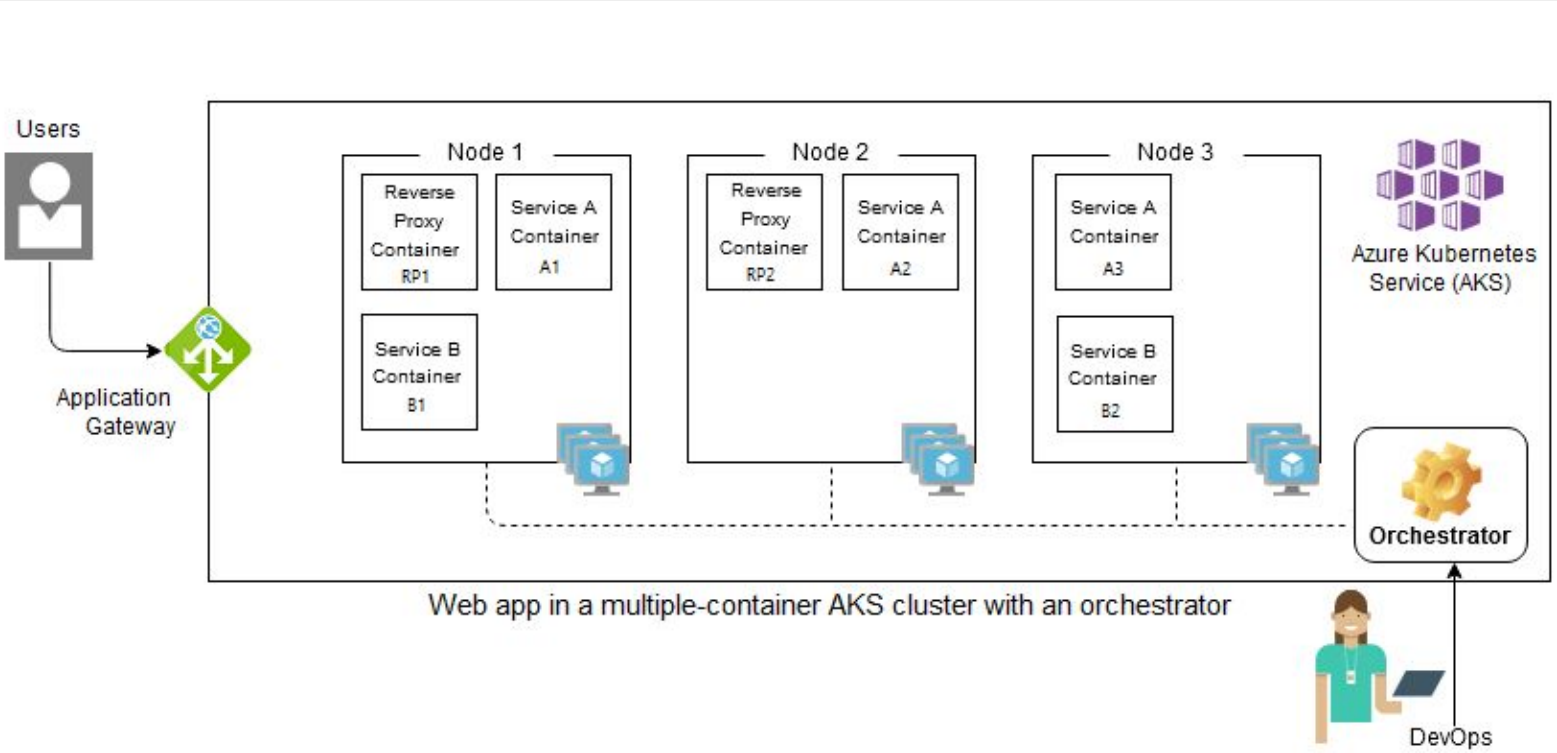
```
# run an ARM64 Alpine on x86_64 host (or vice versa)
docker run --rm --platform=linux/arm64 alpine uname -m
# → aarch64

# your image example
docker run --rm --platform=linux/arm64 -p 8080:8080 docker.io/sabastante/ekohack-api:1.0.0
```

/Capítulo \3

Orquestación con Compose

- Docker Compose
- Arquitectura
- Pros y Cons
- Usos vs K8s

Por qué existen los Orquestadores? (1)



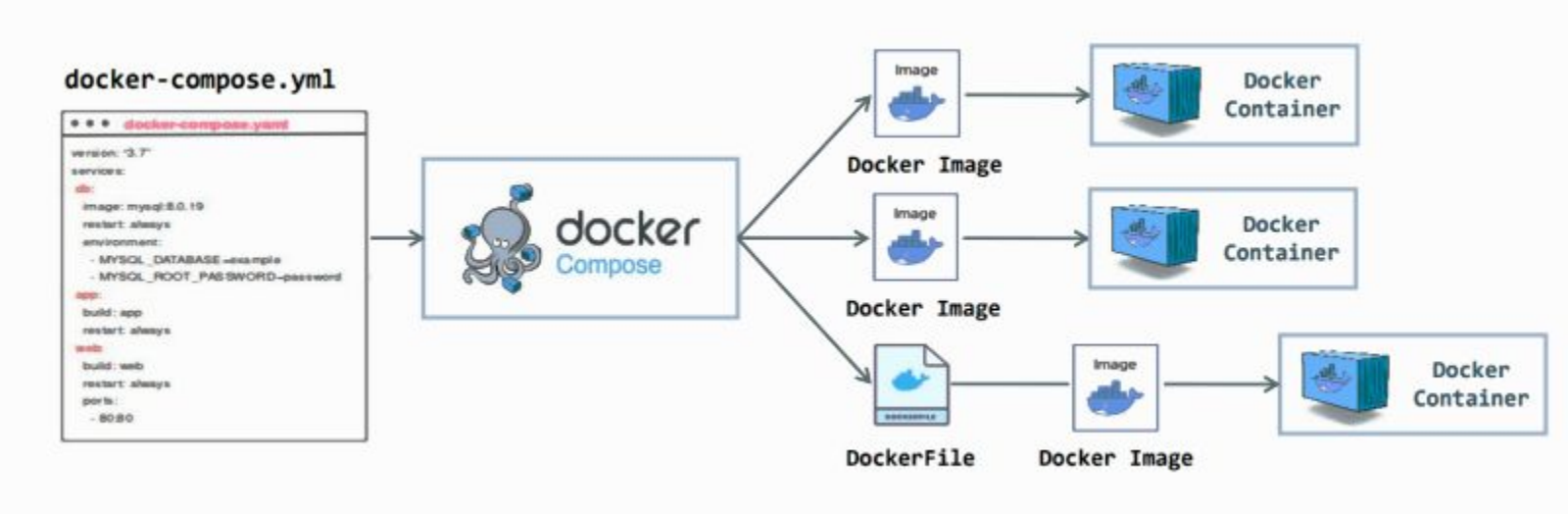
Web app in a multiple-container AKS cluster with an orchestrator

Por qué existen los orquestadores? (2)

Correr containers manualmente es dificil por:

- *Gestionar múltiples servicios (frontend, backend, db);*
- *Administrar restarts automáticos;*
- *Load balancing;*
- *Deploys con Zero-downtime;*
- *Infraestructura distribuida;*
- *Manejo de Secretos;*
- *Rolling updates;*
- *Rollbacks.*

Docker Compose (1)



Docker Compose (2)

compose.yaml U

conferences > docker > docker_compose > compose.yaml

```yaml
version: "3.9"
# ▷Run All Services
services:
  # ▷Run Service
  api:
    image: ghcr.io/example/api:1.0.0
    ports:
      - "8080:8080"
    environment:
      - DATABASE_URL=postgres://user:pass@db:5432/app
    depends_on:
      - db

  # ▷Run Service
  db:
    image: postgres:16
    environment:
      - POSTGRES_PASSWORD=pass
    volumes:
      - db_data:/var/lib/postgresql/data

volumes:
  db_data:
```

Comandos:

- *docker compose up*
- *docker compose up -d*
- *docker compose down*
- *docker compose ps*
- *docker compose logs -f*
- *docker compose exec api sh*
- *docker compose up -d --scale api=3*

Docker Compose (3)

conferences > docker > docker_compose > ! compose2.yaml

```yaml
version: "3.9"

services:
  nginx:
    image: nginx:1.25
    ports:
      - "8080:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - api

  api:
    build: ./api
    environment:
      - DB_HOST=db
      - DB_USER=app
      - DB_PASSWORD=pass
    ports:
      - "3000:3000"
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
      interval: 10s
      timeout: 3s
      retries: 3

  db:
    image: postgres:16
    environment:
      - POSTGRES_PASSWORD=pass
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

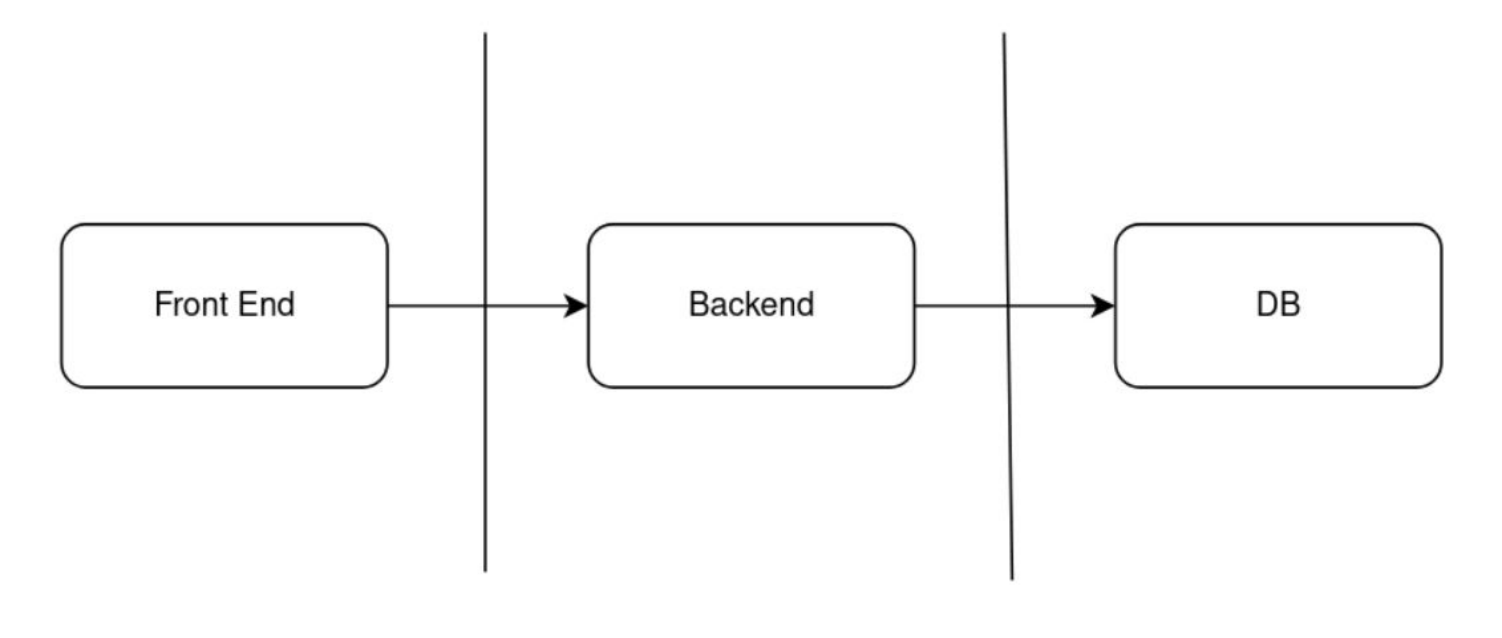
Comandos:

- *docker compose up*
- *docker compose up -d*
- *docker compose down*
- *docker compose ps*
- *docker compose logs -f*
- *docker compose exec api sh*

Ejercicio 4 - Compose

1. Crear una carpeta /api
2. Crear un nginx.conf
3. Crear un compose.yaml
4. Agregar nuestra api creada anteriormente
5. Correr docker compose up!
6. Escalar el deployment a 3 containers
7. Correr docker ps
8. Correr docker log -f y ejecutar request a la api

Architecture Review



/Capítulo \4

Kubernetes

- Kubectl / Kubeconfig
- Pod
- Deployment
- RBAC
- Namespaces
- Manifiestos
- Networking
- Volumes

Kubernetes (1)

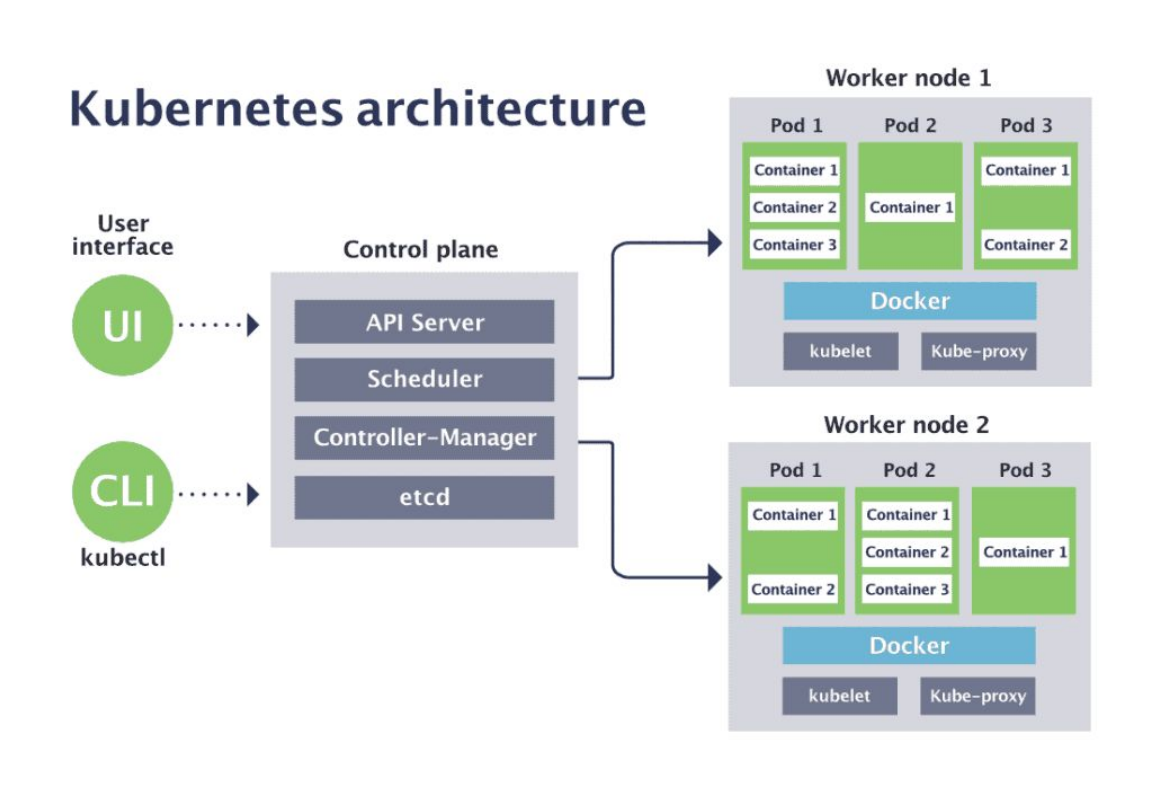
Por que Kubernetes?

1. Deployments Distribuidos entre nodos;
2. Scheduling;
3. Failover;
4. Autoscaling;
5. RBAC.

Regla de 3 simple:

1. Compose.yaml -> deployment.yaml
2. Volumes -> persistentVolumeClaim
3. Ports -> Service

Kubernetes (2)

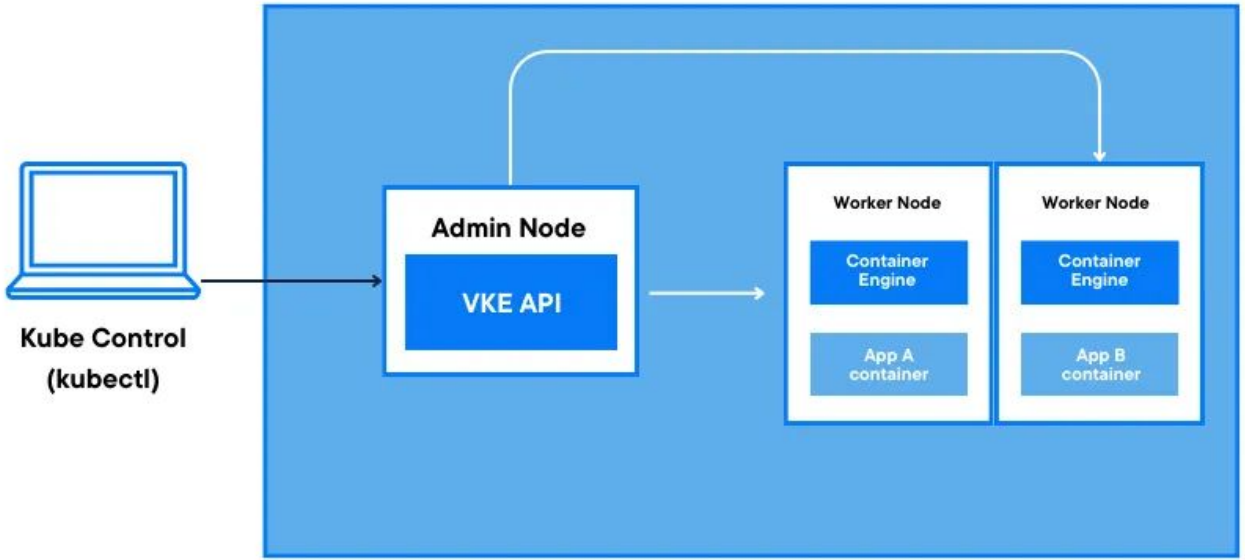


Kubernetes (3)

Compose → Kubernetes mental model

Concept	Docker Compose	Kubernetes
Container	service	pod
Multiple replicas	scale	replicaSet
Multi-container app	services block	deployment
Networking	networks	services / clusterIP
Persistence	volumes	PVC / PV
Env vars	environment	env
Healthchecks	healthcheck	readiness/liveness probes
Startup order	depends_on	initContainers
Secrets	environment file	k8s secrets

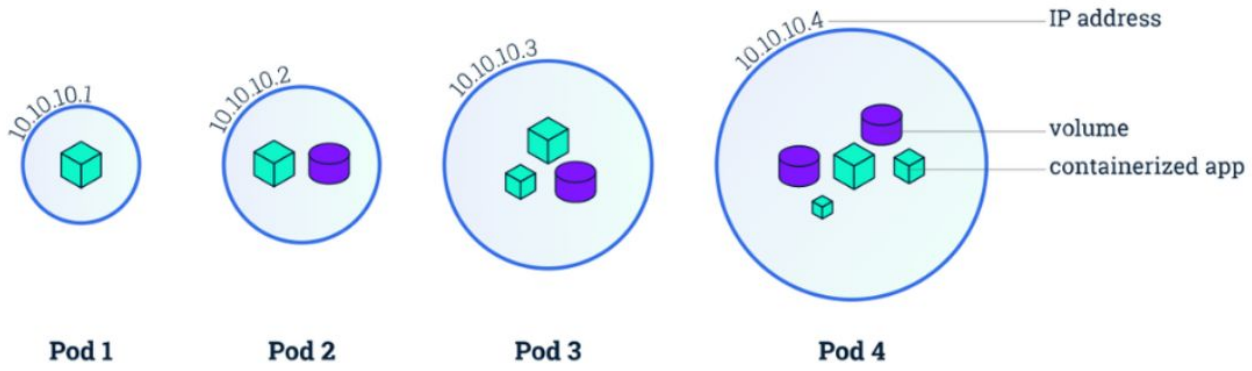
kubectl


Kube Control
(kubectl)
Admin Node
VKE API
Worker Node
Container Engine
App A container
Worker Node
Container Engine
App B container
VULTR

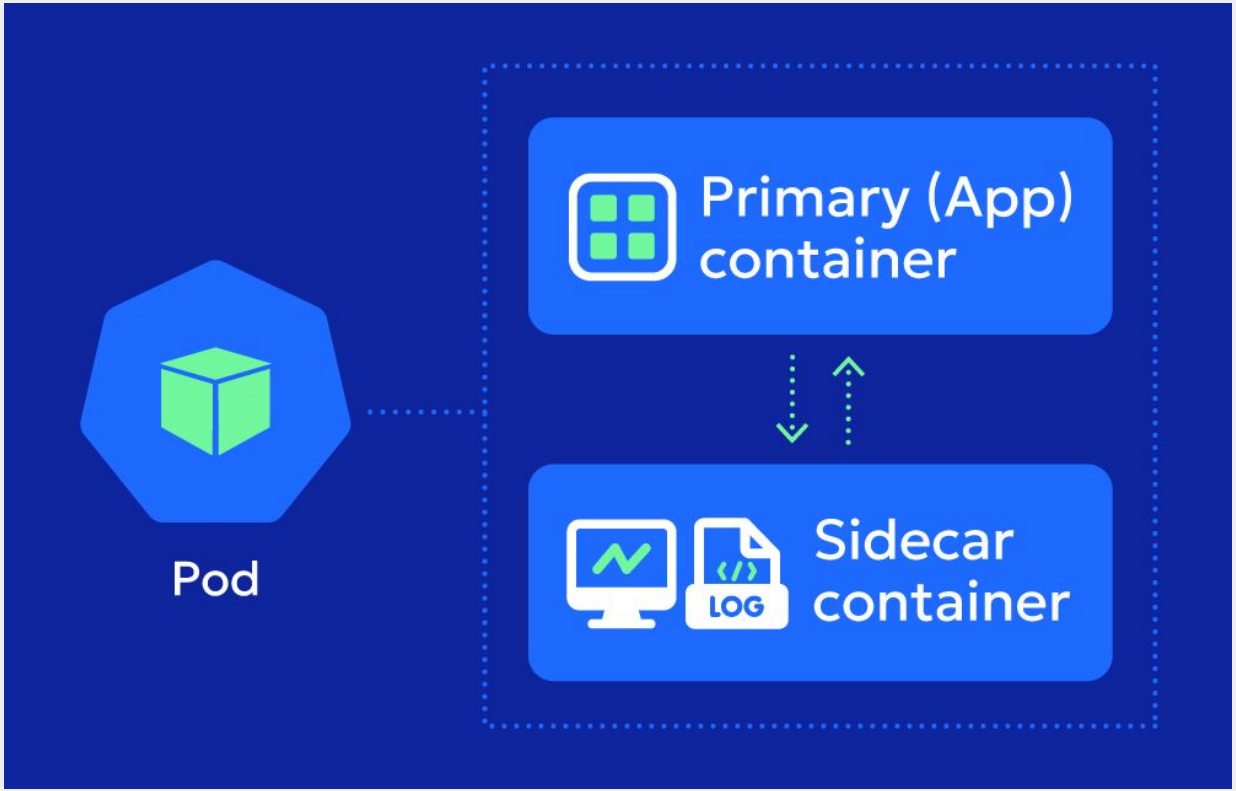
kubeconfig

```
santi@fedora:~$ cat .kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t
nc2T3U3NHN3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhNS2EzVmlaW
F4TXpGYUZ3MHpOVEV4TWpNeE5UQTJNekZhTUJVeApFekFSQmdOVkJBTVRDbXQxWW1WeW
RRUJBUVVBQTRJQkR3QXdnZ0VLCkFvSUJBUUN4SnhGNzF1aGJDc3N0OUtFU2RmVVZYY0S
QkNoT1kwT28KTjhuN2VCbjJvbXhTK3B2a3RTRVNyN2xsM2NIbDhJZGRKbk1KbWZIdG5T
WYweU0vbFhUMGNNa1NBT3h5WEZzYThLeTBacDV0RW0rcjcwbDNvTHRlUEkxM04xcGZDQ
dIQXBIbGRpSDJGMGptZm5ScndCb08vYXBiaGdtendtR1BPMFptMlozcGVZWUNXRUptWT
Qb2V6TzB2d3kyR0lDNW12SXBRcUkra1ZDTG90Y2hkRGVuZUEzZ0RYSQpZYmw3dzFWZWc
R2pXVEJYTUE0R0ExVWREd0VCL3dRRUF3SUNwREFQCkJnTlZIUk1CQWY4RUJUQURBUUgv
nNFdzhpWXRFa0RxSE5FRkVUQVYKQmdOVkhSUVEakFNZ2dwcmRXSmxjbTVsZEdWek1BM
EycCs0VGhPcQpUdFYzRzJ2ZW5PMG1CVFdFSTRZSGdFZFRKTVJQNFJBZCtMeG9GMjZ4a2
2cm5oU0ErdnM1K3FuVEVQUnJVM1NwMlI3N1NvMExXM09QZmVVWnBxYVhQTzZmUGxsYVk
SzROOE9hY0ZiNm16S2FPZHFuelpneFRnUE5VZUk0L2RnbVo4L2swMUJmUkdOUklQT1VR
TF1WkcyTTBWbEFYT0lRbTZGTFBZOHA1RVRFM25hdkFQdmVqeG9GWlVkCklaNG5RN1JNT
```

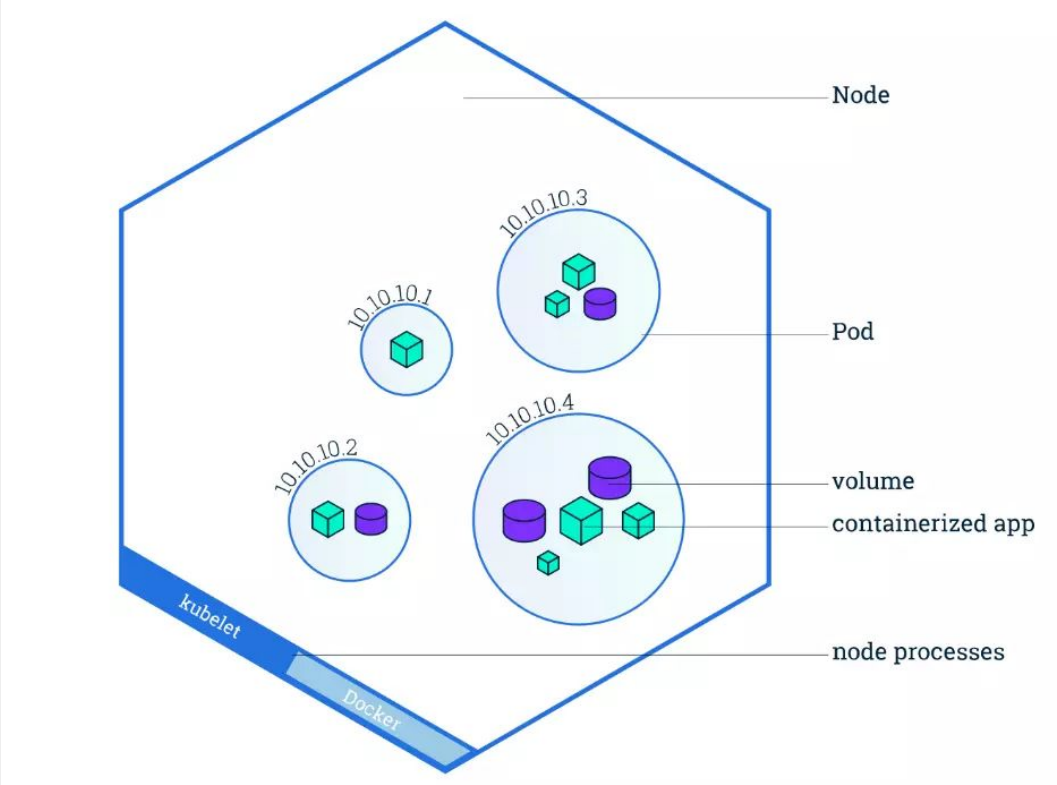
Pods (1)


10.10.10.1
10.10.10.2
10.10.10.3
10.10.10.4
IP address
volume
containerized app
Pod 1
Pod 2
Pod 3
Pod 4

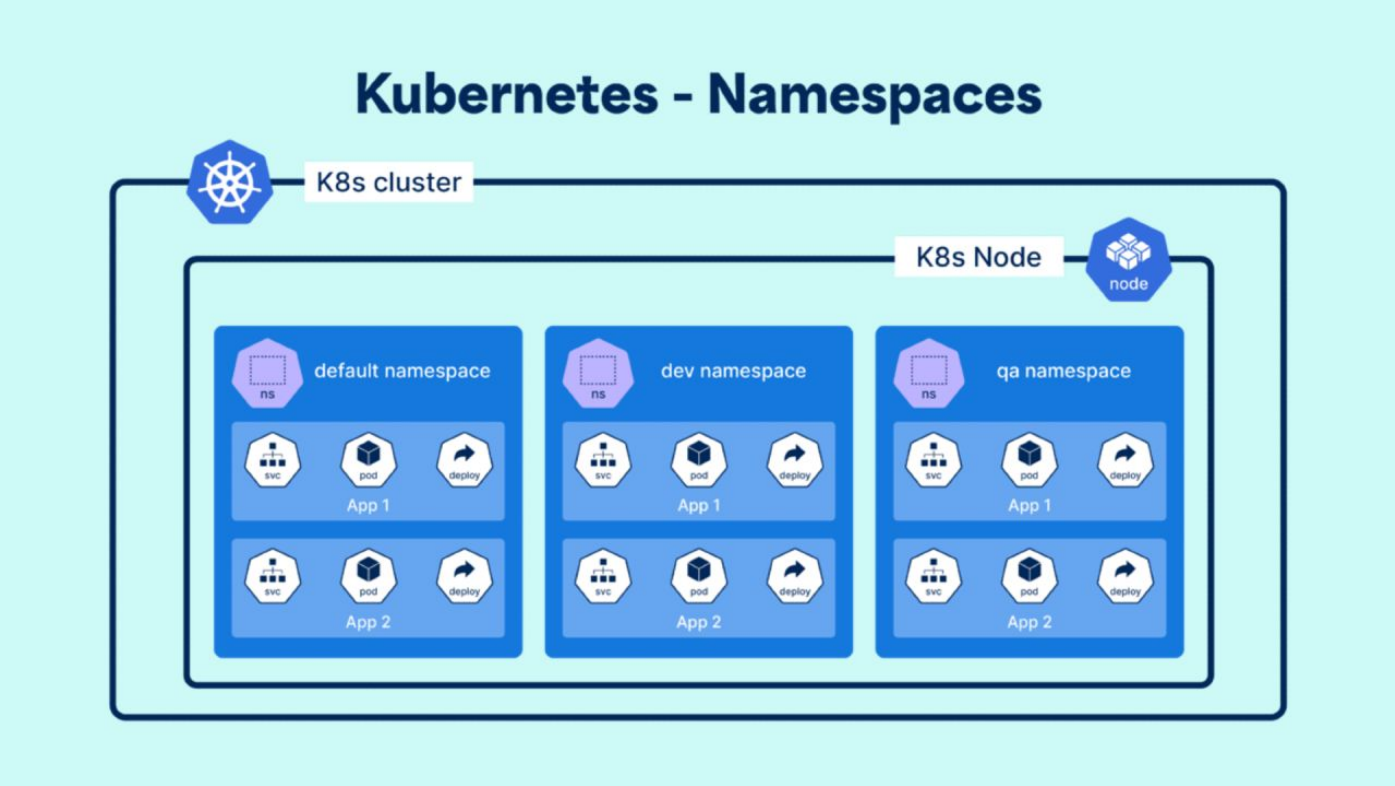
Pods (2)



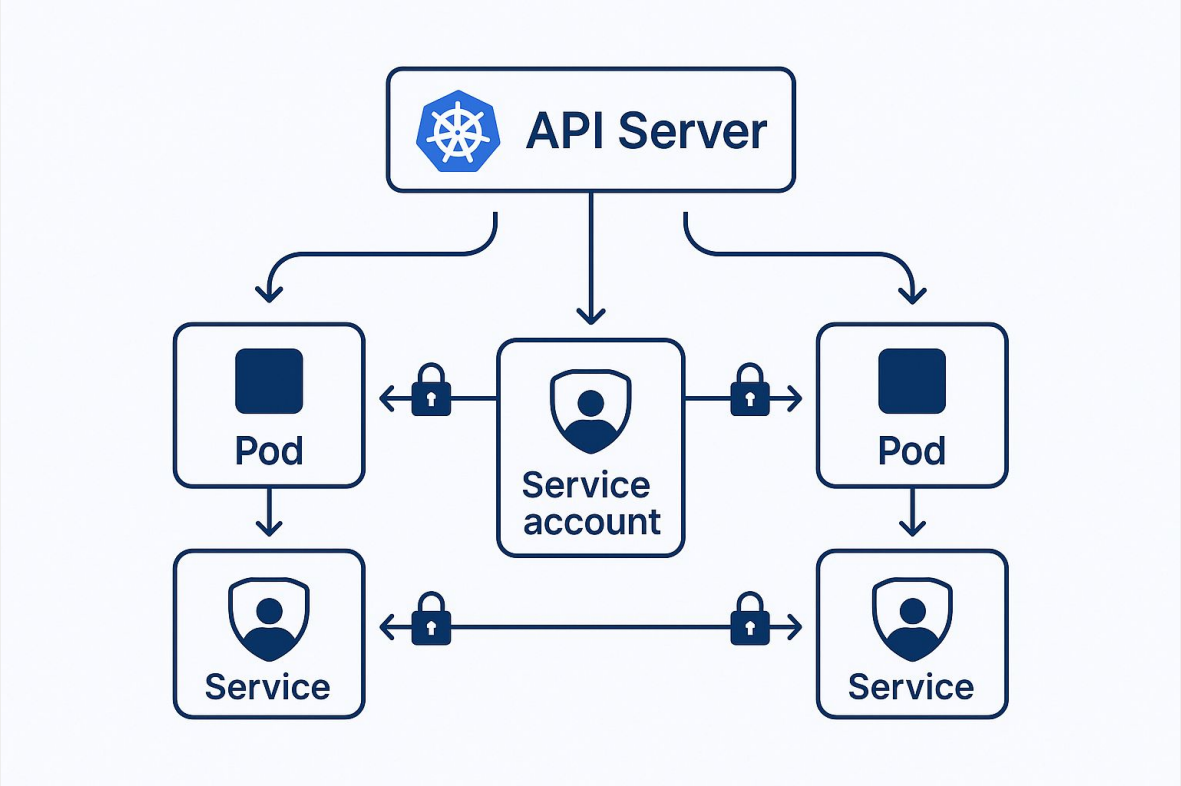
Nodes


Node
10.10.10.3
10.10.10.1
Pod
10.10.10.4
10.10.10.2
volume
containerized app
kubelet
Docker
node processes

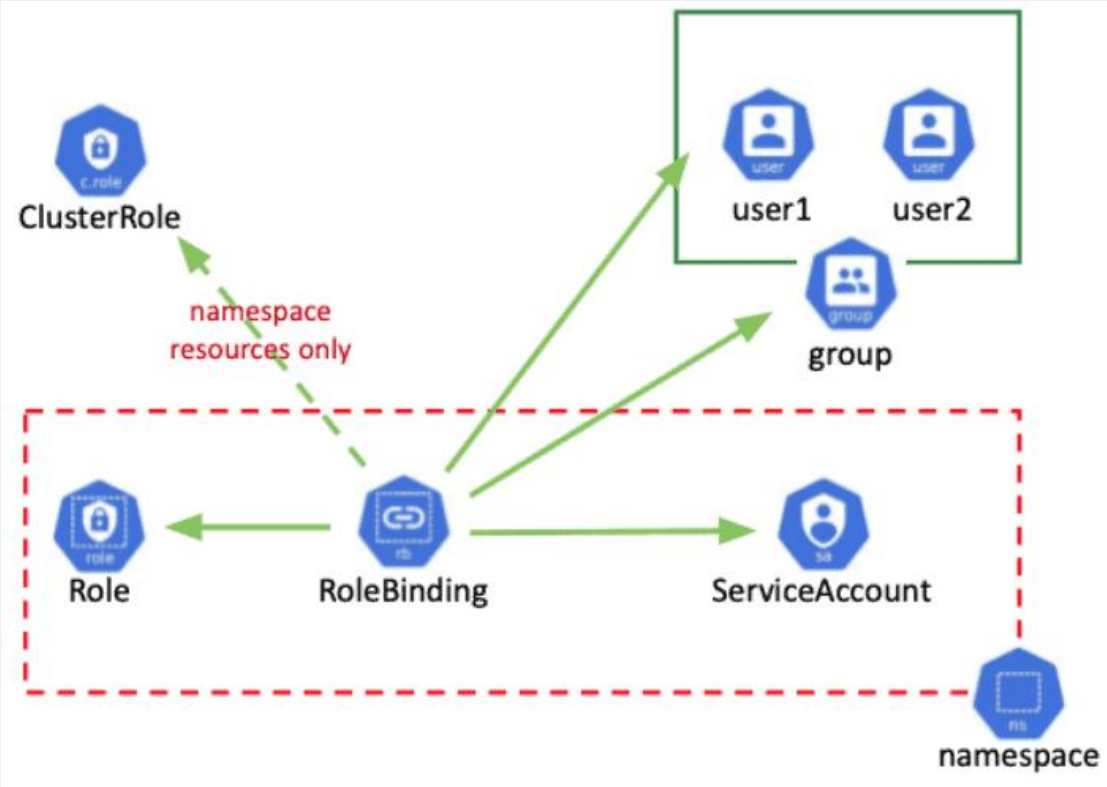
Namespaces


Kubernetes - Namespaces
K8s cluster
K8s Node
node
ns
default namespace
svc
pod
deploy
App 1
App 2
dev namespace
qa namespace

Identity and Access Management (1)



Identity and Access Management (2)



Identity and Access Management

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: team-a-readwrite
  namespace: team-a
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch", "create", "delete"]
```

Manifiestos!

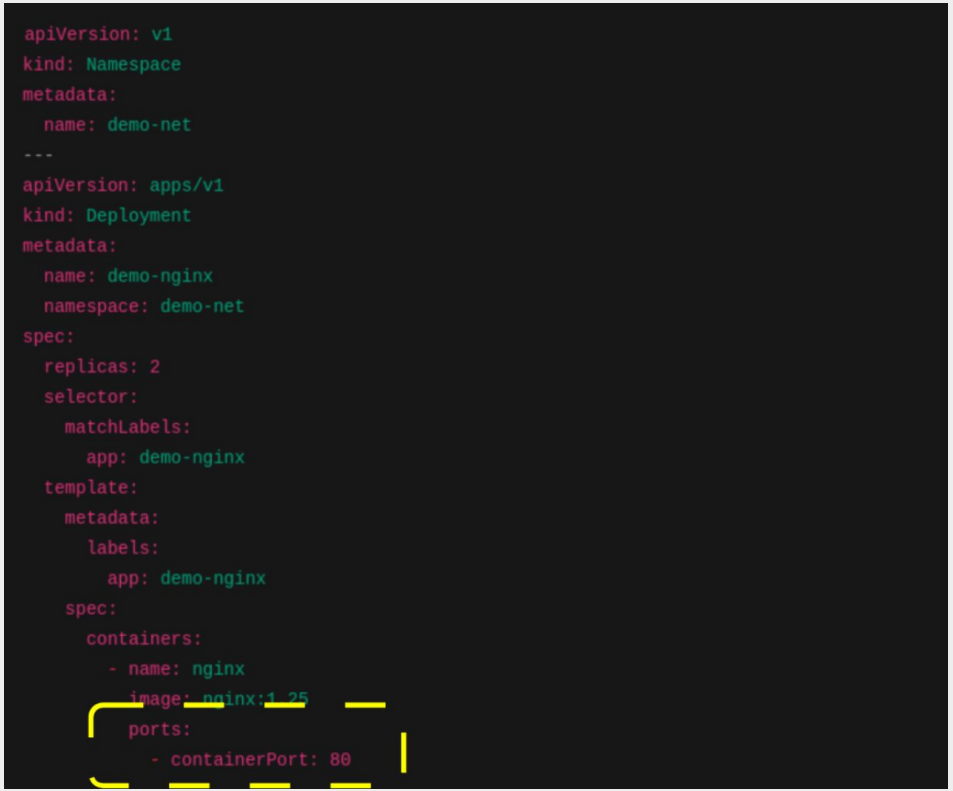
conferences > docker > docker-security > training_ground > ejercicio_6 > ! nginx.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:stable
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
```

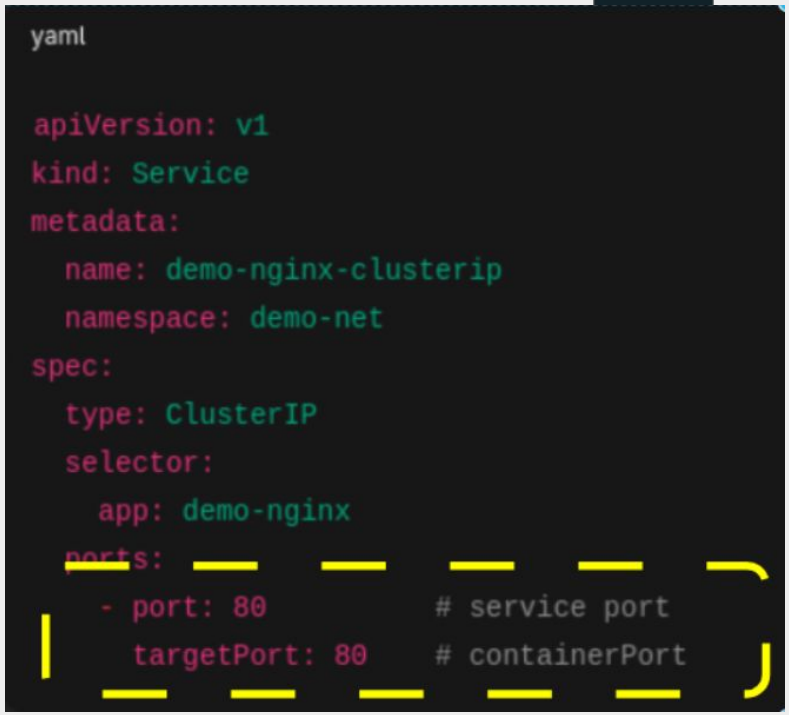
Networking (1)

- **Pod-to-pod-networking (ip variable)**
  - Cada pod tiene su ip
  - Cada pod puede ver a los demás
  - No NAT
- **Service Networking (Una ip fija)**
  - Asigna una IP Virtual
  - Balancea tráfico entre pods
  - Sigue vivo si el pod muere

Networking (1.1)



```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: demo-net
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-nginx
  namespace: demo-net
spec:
  replicas: 2
  selector:
    matchLabels:
      app: demo-nginx
  template:
    metadata:
      labels:
        app: demo-nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.25
          ports:
            - containerPort: 80
```



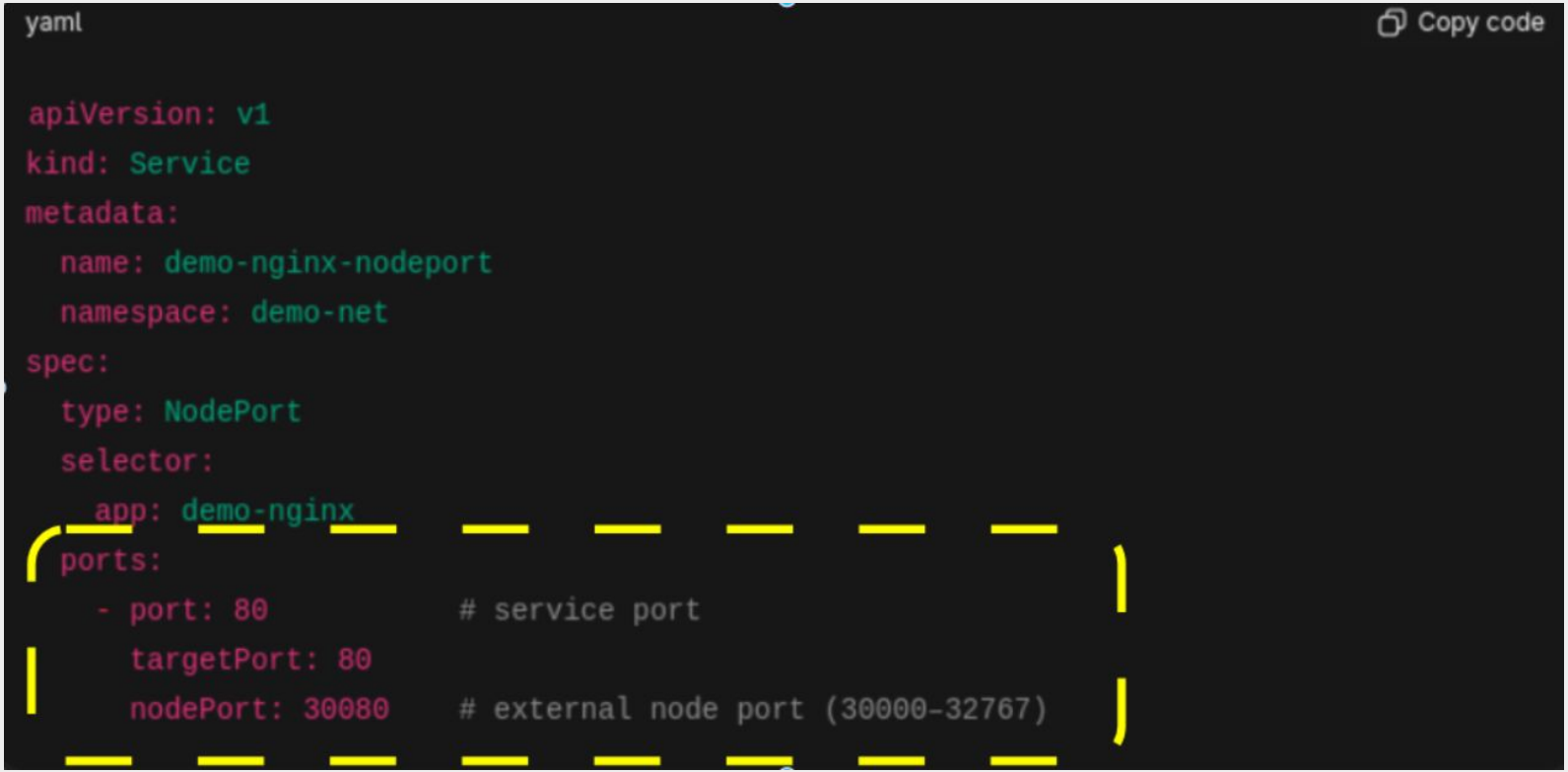
yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: demo-nginx-clusterip
  namespace: demo-net
spec:
  type: ClusterIP
  selector:
    app: demo-nginx
  ports:
    - port: 80          # service port
      targetPort: 80    # containerPort
```

Networking (2)

- **NodePort**
  - Expone tu servicio en todos los nodos
  - Puertos del 30000–32767
- **LoadBalancer**
  - Asigna una IP real
  - Usado por Cloud Providers (Ex, AWS usa el ALB)
- **Ingress**
  - L7 Routing (/health -> A | /version -> B)
  - Necesita un ingress controller

Networking (2.1)



Networking (2.2)

yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: demo-nginx-lb
  namespace: demo-net
spec:
  type: LoadBalancer
  selector:
    app: demo-nginx
  ports:
    - port: 80
      targetPort: 80
```

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: demo-nginx-ingress
  namespace: demo-net
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx   # depends on your controller
  rules:
    - host: demo.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: demo-nginx-clusterip
                port:
                  number: 80
```

Networking (3.1) - Policy

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-ingress
  namespace: demo-net
spec:
  podSelector: {}   # select ALL pods in the namespace
  policyTypes:
    - Ingress
  ingress: []       # deny everything
```

Networking (3.2) - Policy

yaml Copy code

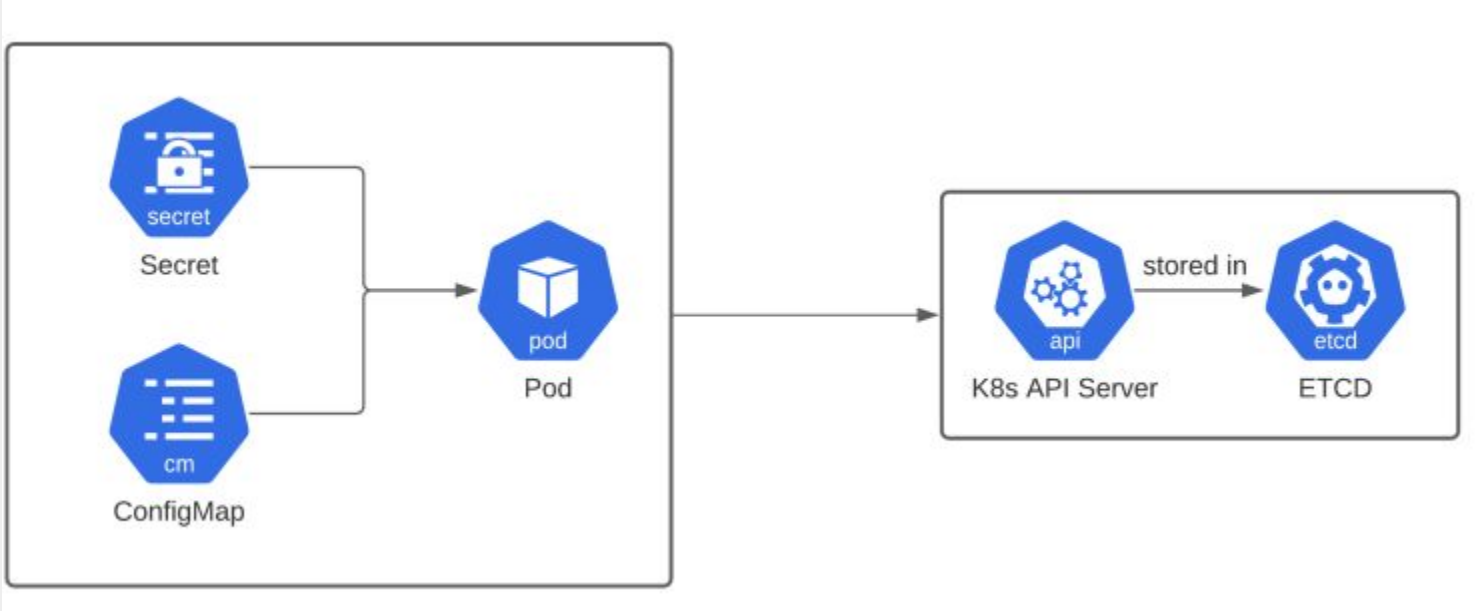
```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-nginx
  namespace: demo-net
spec:
  podSelector:
    matchLabels:
      app: demo-nginx           # protect nginx pods
  policyTypes:
    - Ingress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: frontend    # only pods with this label
      ports:
        - protocol: TCP
          port: 80
```

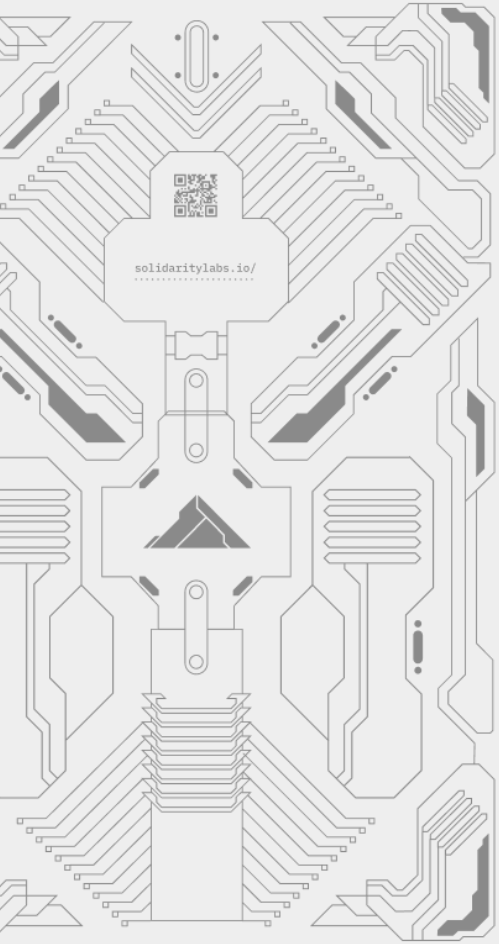
Volumes

- **Ephemeral Volumes** (Se pierde cuando el pod muere)
  - emptyDir
    - Compartido entre containers
  - Configmap/Secrets
    - Se usan para configuraciones
  - downwardAPI
    - Expone la metadata de la API
- **Persistent Volumes**
  - El storage del cluster
-

- **Modos de Acceso**
  - RWO (ReadWriteOnce)
    - Solo un nodo puede montarlo en Read/Write
  - ROX (ReadOnlyMany)
    - Múltiples nodos pueden leer
  - RWX (ReadWriteMany)
    - Múltiples nodos en Read/Write

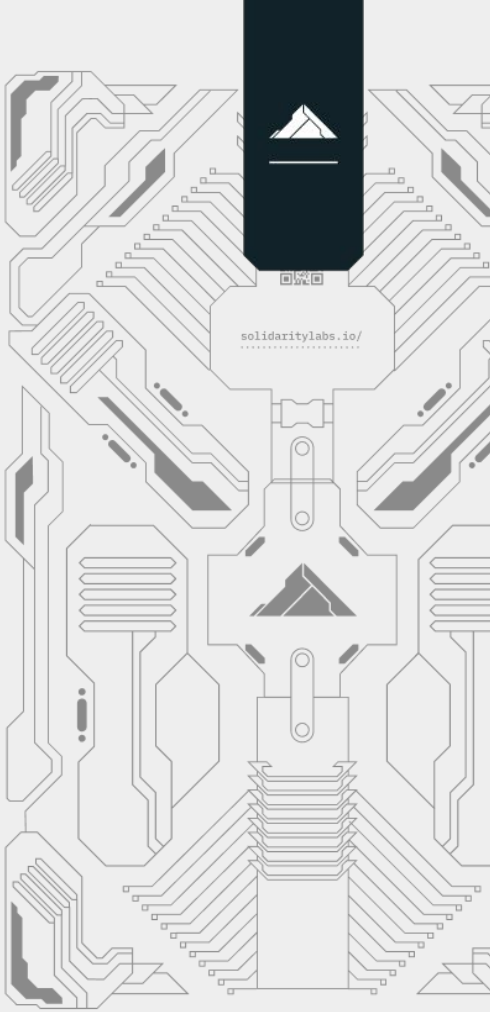
Configmaps / Secrets





Demo

Usando nuestro cluster local



Seguridad en k8s

1. Usar RBAC y Mínimo privilegio
2. Separar contextos en Namespaces
3. Regular el tráfico de networking
4. Minimizar permisos en contenedores
5. Usar registros privados
6. Controlar la cantidad de recursos
7. Securizar el API Server
8. Usar herramientas de seguridad en runtime - AKA Falco

Ejercicio 6.1 - Kubernetes RBAC

1. Levantar tu cluster de k8s con minikube
2. Crear 2 namespaces
3. Crear una service account en uno de los namespace
4. Obtener acceso a la service account
5. Crear un pod en uno de los namespaces
6. Comprobar que no se pueda ver el otro namespace

Ejercicio 6.2 - Kubernetes RBAC

1. Correr un pod con nginx en tu cluster en uno de los namespaces.

2. Correr la API que hicimos con compose en kubernetes.

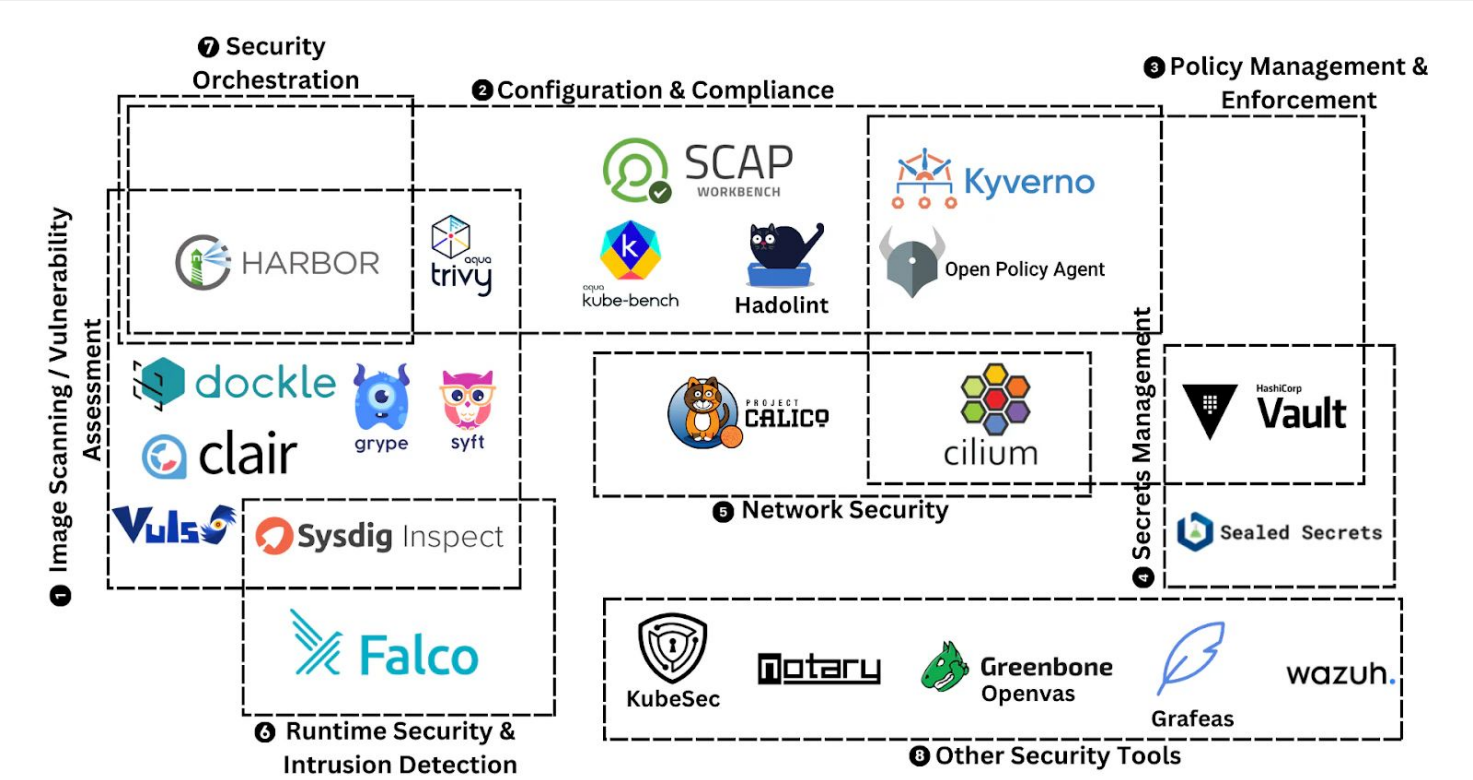
3. Agregar un /version a la api y hacer un un update, actualizando la versión del deployment.

/Capítulo \5

Defendiendo Containers

- Identificar
- Proteger
- Detectar
- Responder
- Recuperar

Container Security



Seguridad es seguridad, siempre

Identify

Inventory of workloads and assets
Assess risks and vulnerabilities
Understand the organizations risk landscape

Protect

Security policies and procedures
Access control and authentication
Protect sensitive data

Detect

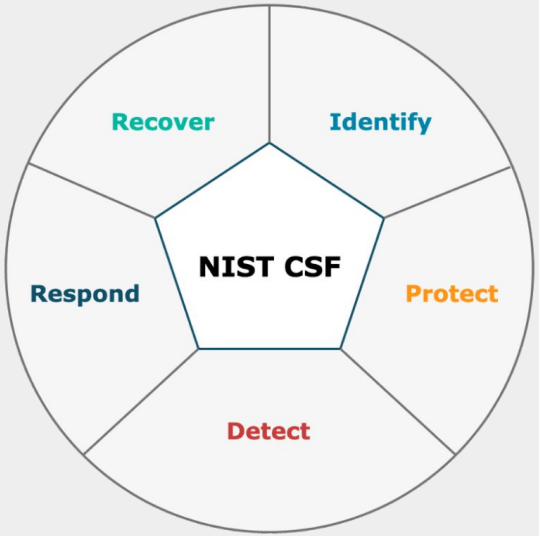
Alert on potential security events
Review and analyze logs for system anomalies
Intrusion detection systems

Respond

Incident response plan
Immediate action to contain the impact
Incident response drills

Recover

System restoration procedures
Post-incident review
Update security controls and incident plan



Identify

Identificar se trata casi siempre de procesos, pero pueden utilizarse herramientas:

- Inventario de Activos
- Identificar aplicaciones
- Conocer las vulnerabilidades
- **Identificar las amenazas**
- Análisis de Riesgo
- Establecer procesos de actualización y patch management
- **Tener un mapa de nuestra organización, un diagrama.**

Lo que no se puede medir, no se puede gestionar.

Trivy

```
trivy [main] ⚡ trivy k8s --report summary
204 / 204 [------------------------------------------------------------------------------------------------------------
```

Summary Report for kind-kind

Workload Assessment

Namespace	Resource	Vulnerabilities					Misconfigurations					Secrets				
		C	H	M	L	U	C	H	M	L	U	C	H	M	L	U
local-path-storage	Deployment/local-path-provisioner	4	32	10	2			1	3	9						

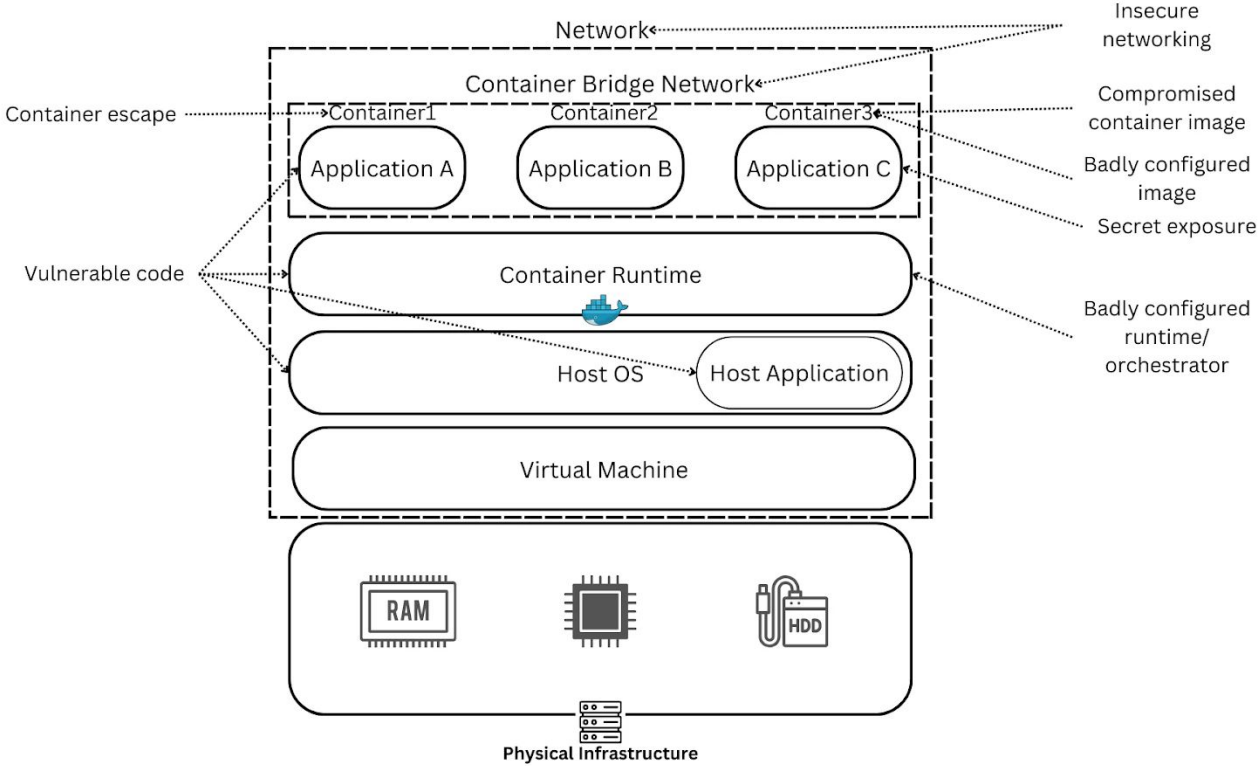
Severities: C=CRITICAL H=HIGH M=MEDIUM L=LOW U=UNKNOWN

Infra Assessment

Namespace	Resource	Vulnerabilities					Misconfigurations					Secrets				
		C	H	M	L	U	C	H	M	L	U	C	H	M	L	U
kube-system	Pod/etcd-kind-control-plane					6		2	4	7						
kube-system	Pod/kube-apiserver-kind-control-plane					5		2	5	17						
kube-system	Deployment/coredns		13	11	3			1	3	4						
kube-system	DaemonSet/kube-proxy	19	49	49	81	3		3	4	9						
kube-system	DaemonSet/kindnet	19	60	60	81	3		3	5	5						
kube-system	Pod/kube-scheduler-kind-control-plane					5		2	4	9						
kube-system	ConfigMap/extension-apiserver-authentication								1							
kube-system	Pod/kube-controller-manager-kind-control-plane					5		2	4	11						
kube-system	ControlPlaneComponents/k8s.io/apiserver			3												
	Node/kind-control-plane		4	7	2		1	4		1						

Severities: C=CRITICAL H=HIGH M=MEDIUM L=LOW U=UNKNOWN

Identify - Threat Landscape (1)



Identify - Threat Landscape (2)

WIZ Platform Solutions Pricing Resources Customers Company Get a demo

← Blog

NVIDIAScape – Critical NVIDIA AI Vulnerability: A Three-Line Container Escape in NVIDIA Container Toolkit (CVE-2025-23266)

New critical vulnerability with 9.0 CVSS presents systemic risk to the AI ecosystem, carries widespread implications for AI infrastructure.

BBC

Home News Sport Business Innovation Culture Arts Travel Earth Audio Video Live

Tesla investigates claims of crypto-currency hack

21 February 2018

Share Save

Researchers Uncover ECScape Flaw in Amazon ECS Enabling Cross-Task Credential Theft

Aug 06, 2025 Ravie Lakshmanan DevOps / Container Security

— Trending News

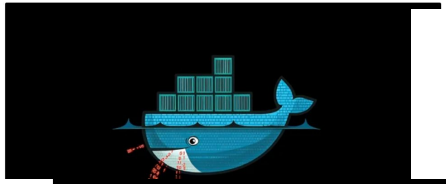
Second Sha1-Hulud Wave Affects 25,000+ Repositories via npm Preinstall Credential Theft

Shai-Hulud v2 Spreads From npm to Maven, as Campaign Exposes Thousands of Secrets

Researchers Spot XZ Utils Backdoor in Dozens of Docker Hub Images, Fueling Supply Chain Risks

Aug 12, 2025 Ravie Lakshmanan Malware / Container Security

— Trending News



'Azurescape' Attack on Azure Container Instances Highlights Risks of Using Multitenant Services

By Kurt Mackie | 09/10/2021

Researchers Uncover ECScape Flaw in Amazon ECS Enabling Cross-Task Credential Theft

Aug 06, 2025 Ravie Lakshmanan DevOps / Container Security

— Trending News

Second Sha1-Hulud Wave Affects 25,000+ Repositories via npm Preinstall Credential Theft

Shai-Hulud v2 Spreads From npm to Maven, as Campaign Exposes Thousands of Secrets

China-Linked APT31 Launches Stealthy Cyberattacks on Russian IT Using Cloud Services

New Albiriox MaaS Malware Targets 400+ Apps for On-Device Fraud and

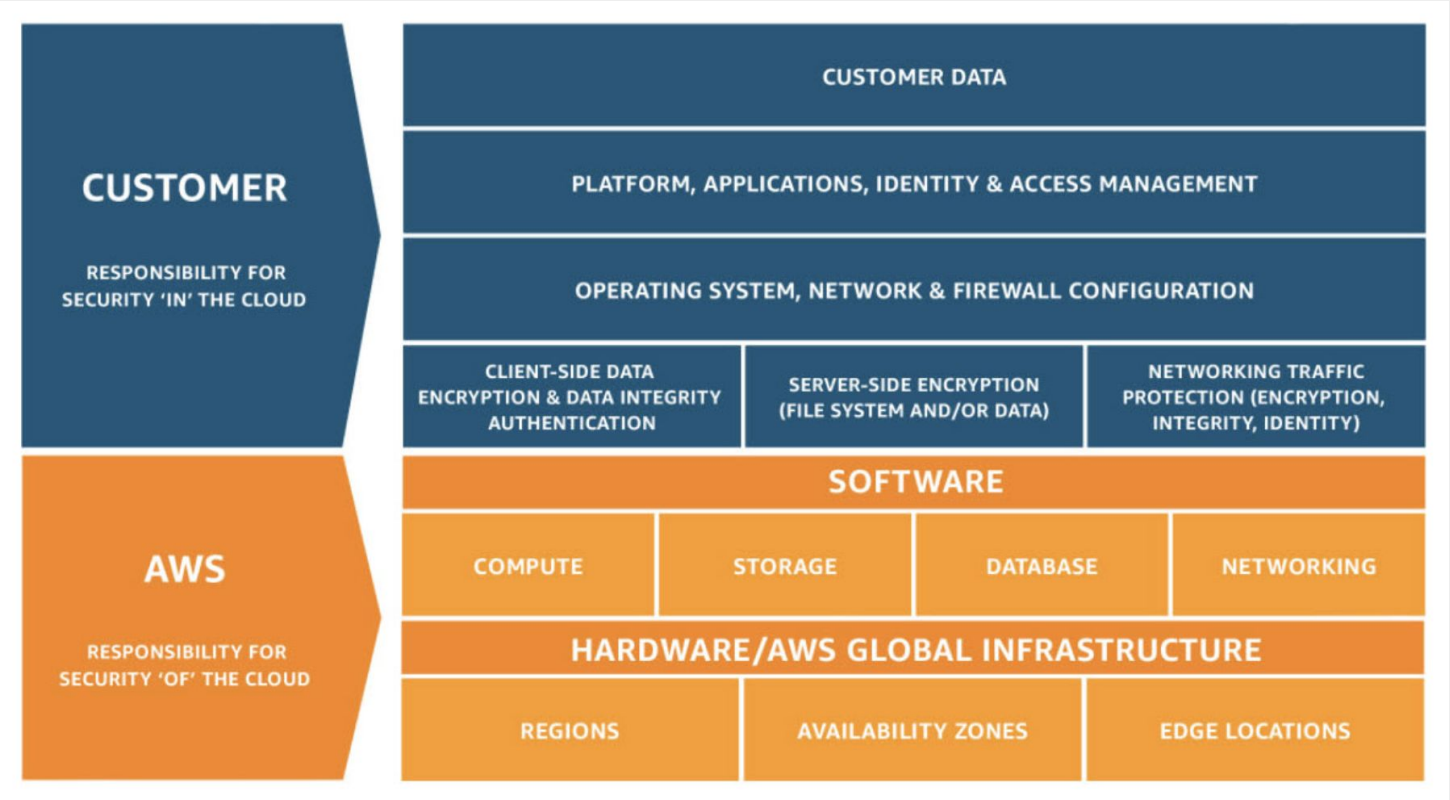
Security

Kong urges action after Docker account compromised, malware uploaded

A cryptominer and trolling in logs? It could have been a LOT worse. Here's what builders should do...

The Stack
Jan 13, 2025 · 2 min read

Share Responsibility Model



Protect - Proteger

Hardening de nuestras imágenes y orquestadores, no debemos pensar solo en el contenedor, sino también en el contexto:

- Asignar usuarios no-root
- Reducir la superficie de ataque - Imágenes “distroless”
- Mínimo Privilegio
- Diseño de Sistemas Seguros - 3 Tier Arch | Auth
- Hardenizar todo el flujo de despliegue, incluidos los registries
- Segmentación de Networking
- Segmentaciones lógicas
- SELinux / AppArmor
- **Las buenas prácticas van antes que las herramientas**

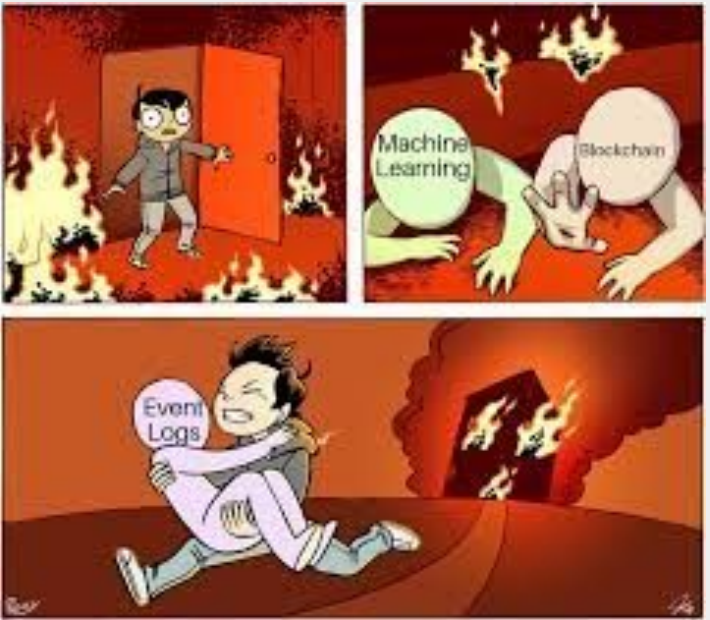
Detect - Detectar

El proceso de detección de amenazas es vital, pero es importante incorporarlo en el momento adecuado para evitar falsos positivos:

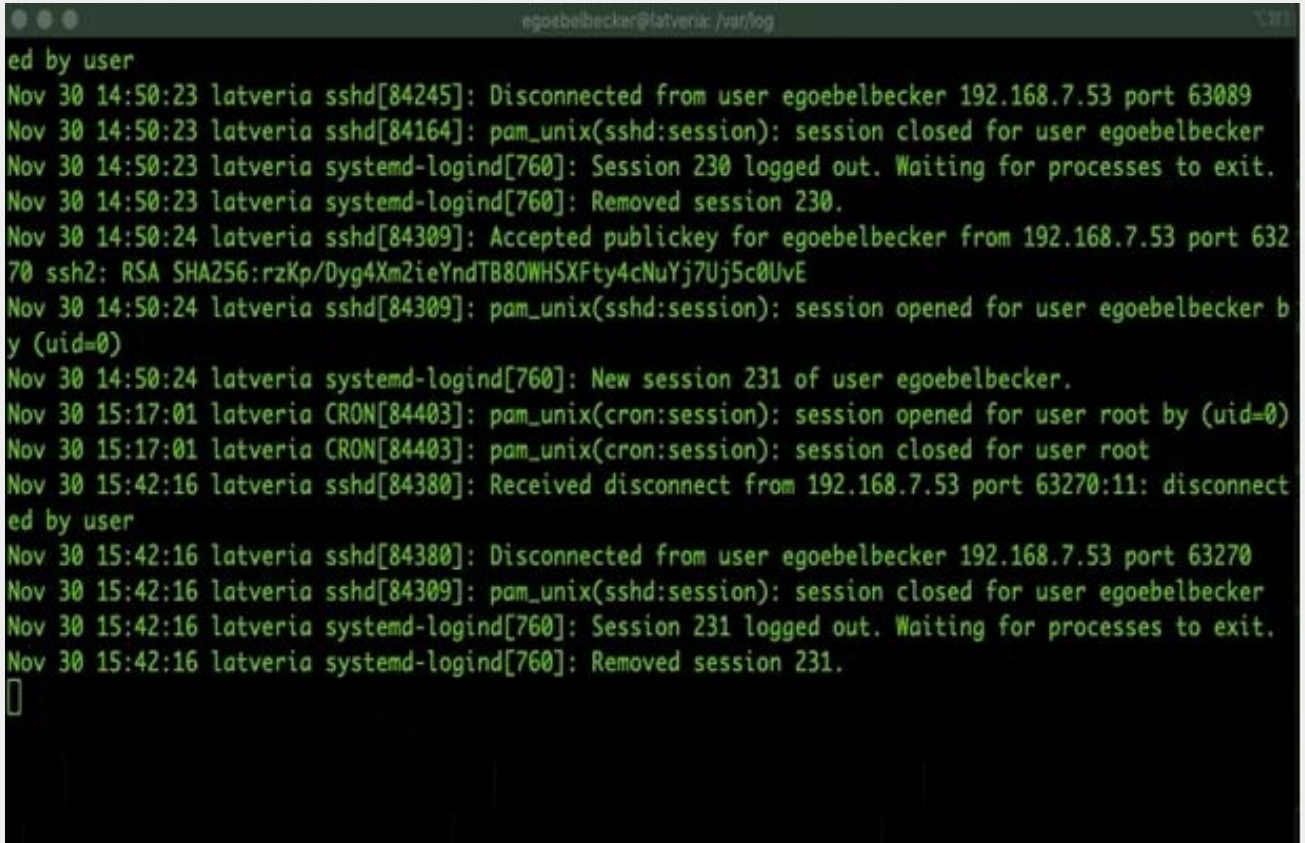
- Debe implementarse para controlar la efectividad de nuestros controles preventivos.
- También puede utilizarse como control compensatorio, en caso de que una necesidad de negocio no nos permita implementar controles preventivos.
- Debemos testear nuestras reglas de detección.
- Debemos conocer nuestro % de cobertura.

Log collection (1)

La única verdad es la realidad, y no podemos tener todos los logs que quisiéramos



Log collection (2) - Docker Logs



Log collection (3) - k8s Logs

1. Container / Pod Log
    a. Log de la App (docker)

2. Logs del Cluster
    a. **Kube-apiserver**: Logs de Kubectl
    b. **Kube-scheduler**: Donde corren los pods
    c. **Kube-controller-manager**: Controladores para administración del cluster
    d. **Etcd**: Acceso a la base de datos de estado de k8s

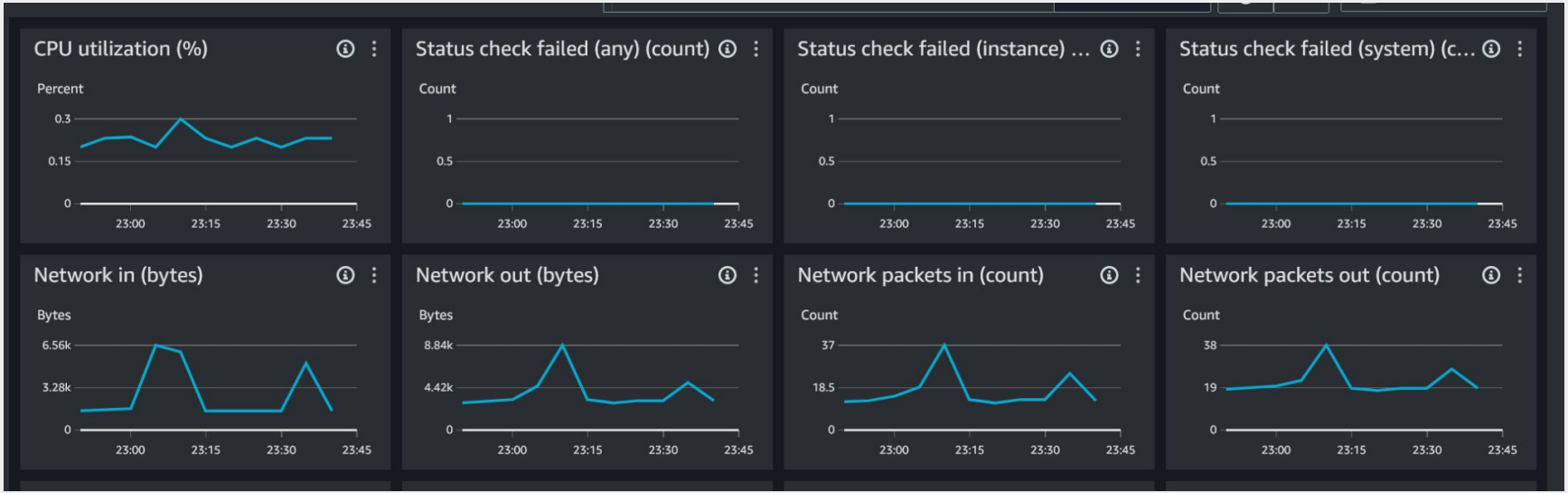
Log collection (4) - Donde se almacenan?

1. On prem?
    a. En los nodos

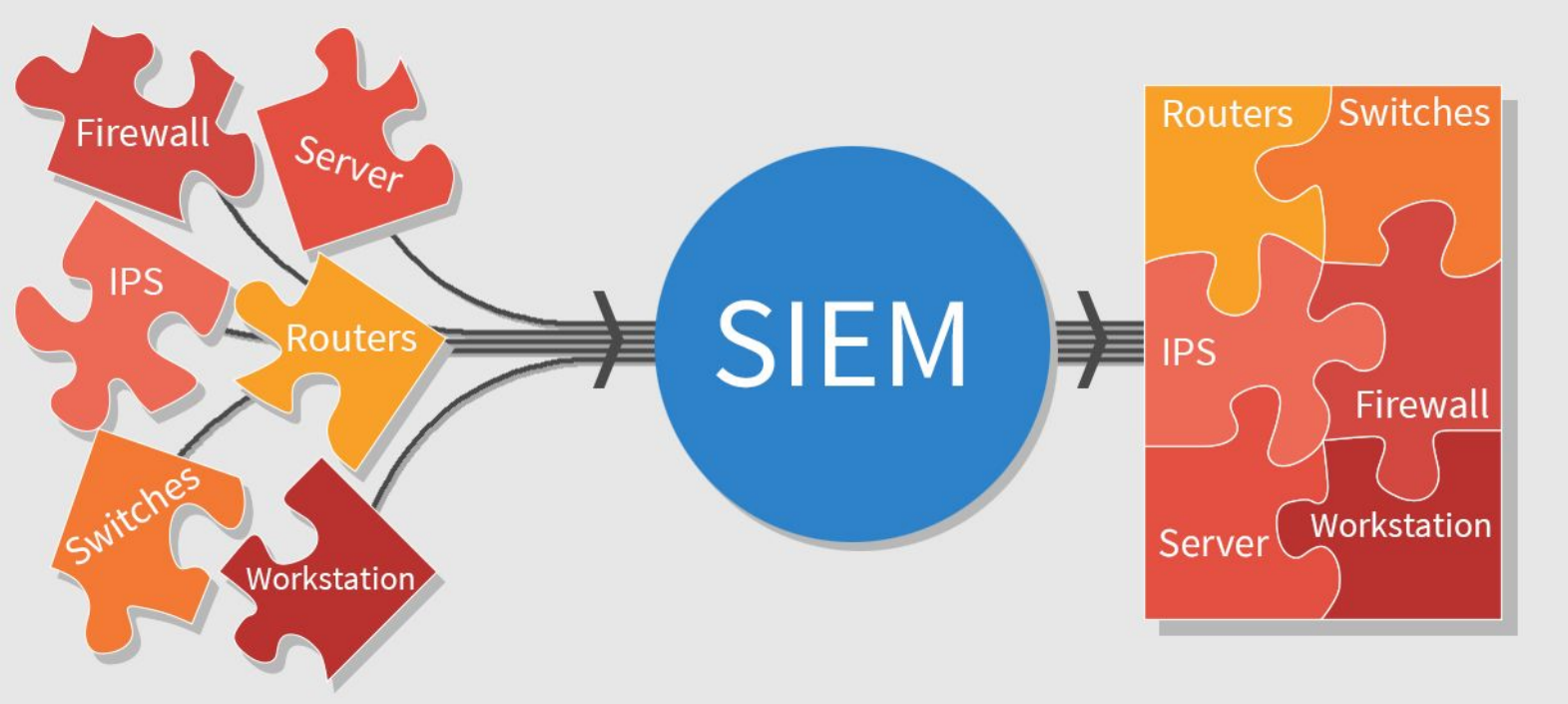
2. En la nube
    a. **AWS - EKS / ECS / Lambda:**
        i. Cloudwatch Logs
    b. **GCP - GKS:**
        i. Google Cloud Logging
    c. **Azure - AKS:**
        i. Azure Monitor / Log Analytics Workspace

Other Registries - Metricas

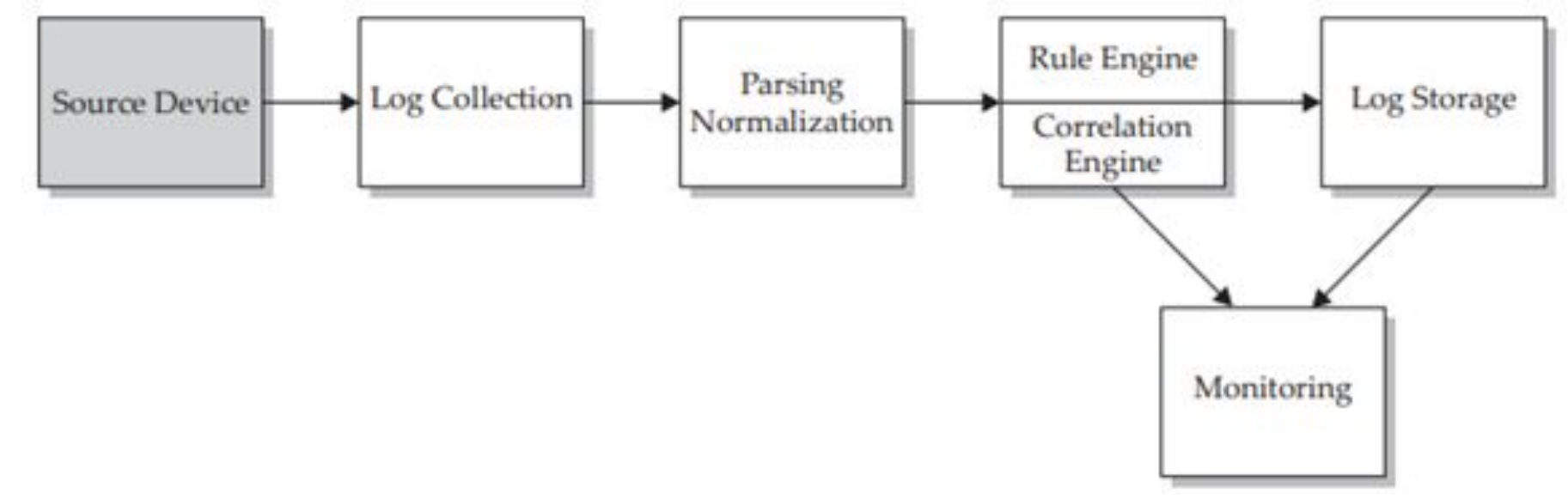
Las métricas en el contexto de la infraestructura se refieren a los datos numéricos y las estadísticas que representan el rendimiento, la utilización y el estado de los recursos de infraestructura, como la CPU, la memoria, la red y el almacenamiento.



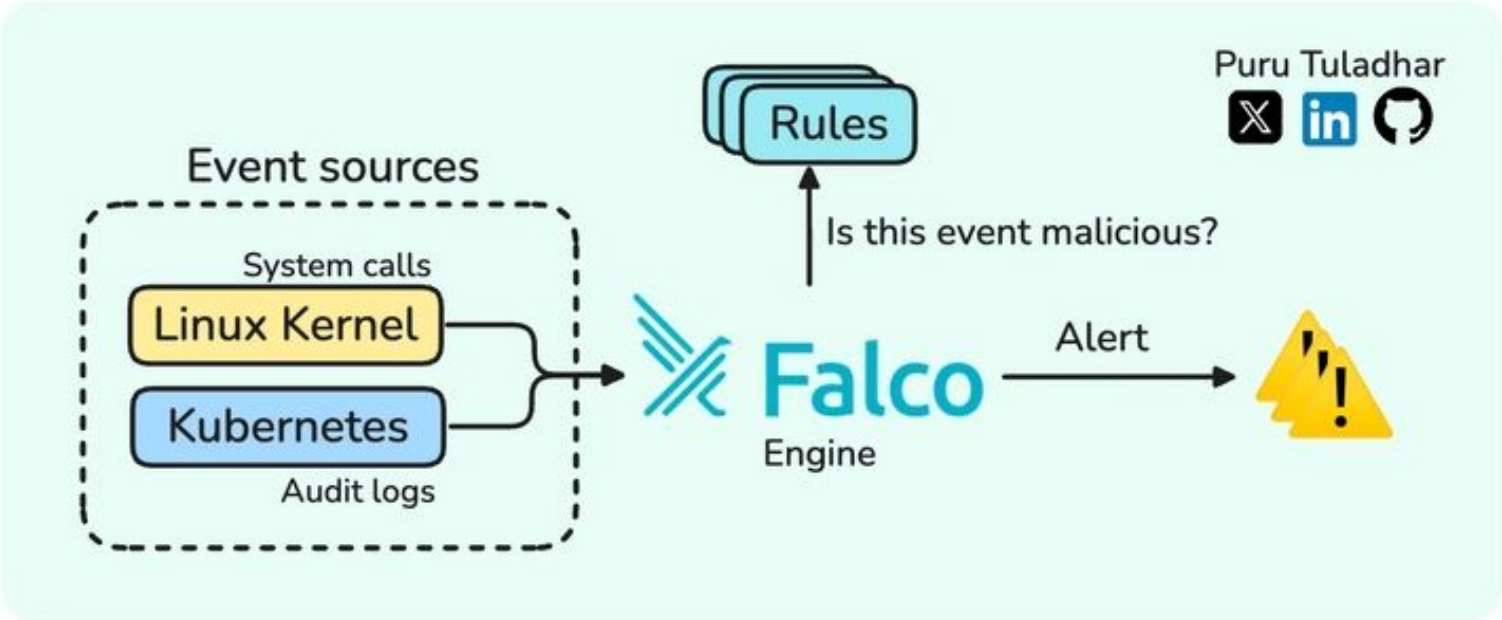
Log collection (5) - SIEM



Log collection (6) - Procesar Logs



Event Driven Detection - Falco k8s



Respond

Incident Response es un proceso que deviene de la detección de eventos anómalos, en este caso de seguridad. El Incidente puede originarse por causa de un ataque externo, pero también puede ser el resultado de mal uso interno o fallas de sistema.

- **Investigación** - Revisar los logs
  - Revisar los logs de k8s y de la app
  - Revisar eventos y métricas

- **Contención** - eliminar la causa del incidente
  - Reiniciar pods
  - Aislar un container comprometido

- **Resolución** - Arreglar lo que dió origen al incidente
  - Arreglar un deployment vulnerable
  - Parchear

/Capítulo \5

Atacando Docker y Orquestadores

-

Docker - Privileged Container

- docker run -it --privileged -v /:/host alpine sh

- Docker Capabilities
  - **CAP_SYS_ADMIN** → mount filesystems, manage kernel parameters
  - **CAP_NET_ADMIN** → change network interfaces
  - **CAP_SYS_PTRACE** → read processes memory

- Podemos usar un docker container para modificar el host:
  - leer /etc/shadow
  - modificar /root/.ssh
  - installar a rootkit
  - reemplazar system binaries
  - borrar logs

Docker - Robar secretos de las Layers

- Cada acción (RUN / COPY / ADD) que hagamos al crear un container genera una layer. Esas layers permanecen por siempre

- Docker Capabilities
  - **CAP_SYS_ADMIN** → mount filesystems, manage kernel parameters
  - **CAP_NET_ADMIN** → change network interfaces
  - **CAP_SYS_PTRACE** → read processes memory

- Podemos usar un docker container para modificar el host:
  - leer /etc/shadow
  - modificar /root/.ssh
  - installar a rootkit
  - reemplazar system binaries
  - borrar logs

k8s - Robar Service Accounts

- Cada pod tiene una service account
  - cat /var/run/secrets/kubernetes.io/serviceaccount/token
  - cat /var/run/secrets/kubernetes.io/serviceaccount/namespace
  - cat /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

- Podemos usar esto para autenticarnos y trabajar con el cluster.

- **El pod necesita permisos sobre el cluster**

k8s - Abuso de Volumenes

conferences > docker > docker-security > training_ground > ejercicio_7 > ! vuln,yml

```
1   volumes:
2   - name: root
3     hostPath:
4       path: /
```

ECS - Cloud Attacks

- Ataques a nivel Container:
  - **Todo lo que vimos de container y registros aplica.**
  - **Todo lo que vimos de networking aplica**
- Ataques a nivel Cloud Management Plane
  - Buscar Variables de entorno en las ECS Task Definitions
  - Exec al Container
  - Modificar las Tasks
  - Escapar

Los ataques de Management Plane necesitan acceso a AWS Session Manager plugin en nuestro pc es requerido

Ejercicio Final (1)

1. ENTREGABLE 1: 3 containers SEGUROS subidos a un registry:
   a. Front End
   b. API Layer
   c. DB

2. ENTREGABLE 2: Escaneo de vulnerabilidades mostrando que las apps son seguras (Usen el del registry o la tool que quieran.

3. ENTREGABLE 3: Docker Compose en repo de Github con las instrucciones de como usarlo y desplegar que cualquiera pueda desplegar las apps.

Ejercicio Final (2)

1. ENTREGABLE 4: Kubernetes manifest en repo de Github con las instrucciones de como usarlo.

2. ENTREGABLE 5: Les voy a dar acceso a un tenant de AWS para que exploten el cluster. Van a tener que:
   a. Hacer exec al container.
   b. Obtener la metadata.
   c. Conseguir credenciales.
   d. Logearse desde su computadora.
   e. Dejar un mensaje para quien sigue.

3. ENTREGABLE 6: Plan de Respuesta a Incidente: Sos parte del equipo de Incident Response y te avisan que se detectó comportamiento sospechoso en el cluster de ECS. **¿Cómo investigarías el incidente?**

/the\ end/

Preguntas?

Happy to answer any questions!

Contact Info:

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

@Solidaritylabs