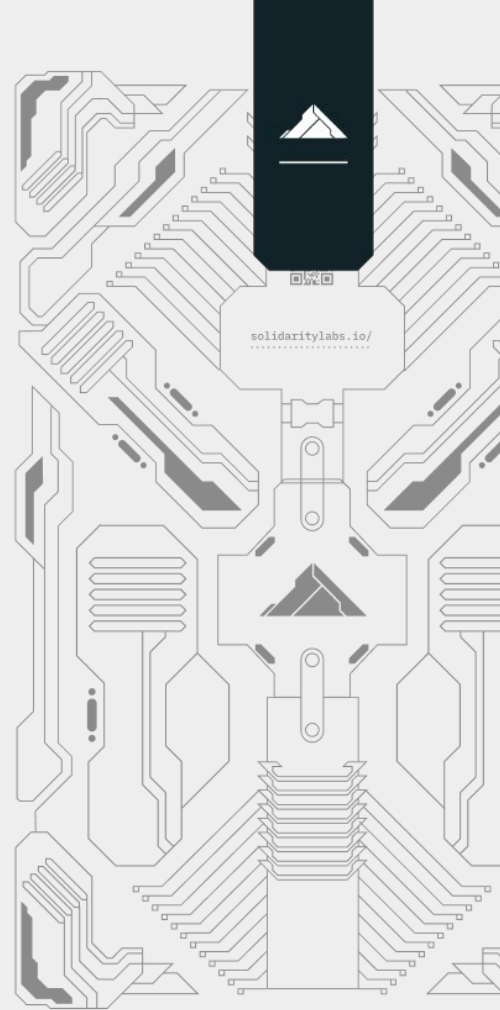


Introducción a ~~Docker~~ Contenedores

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

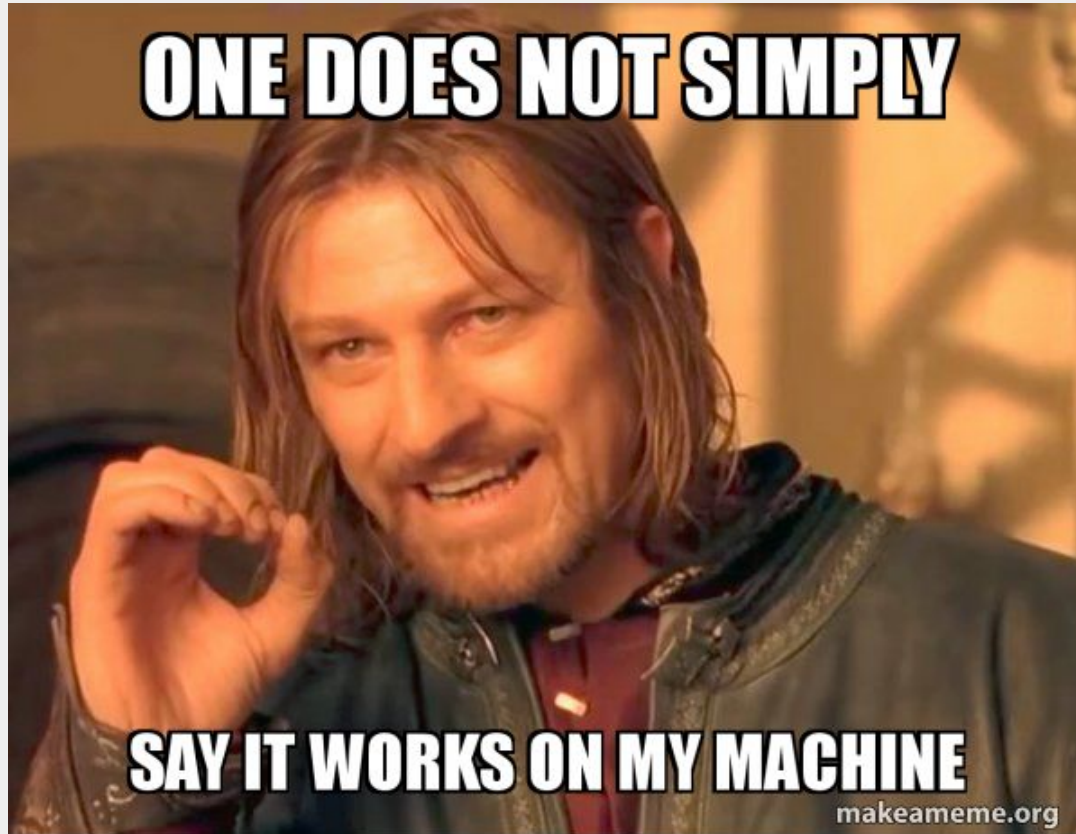
Fundamentos de Contenedores

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

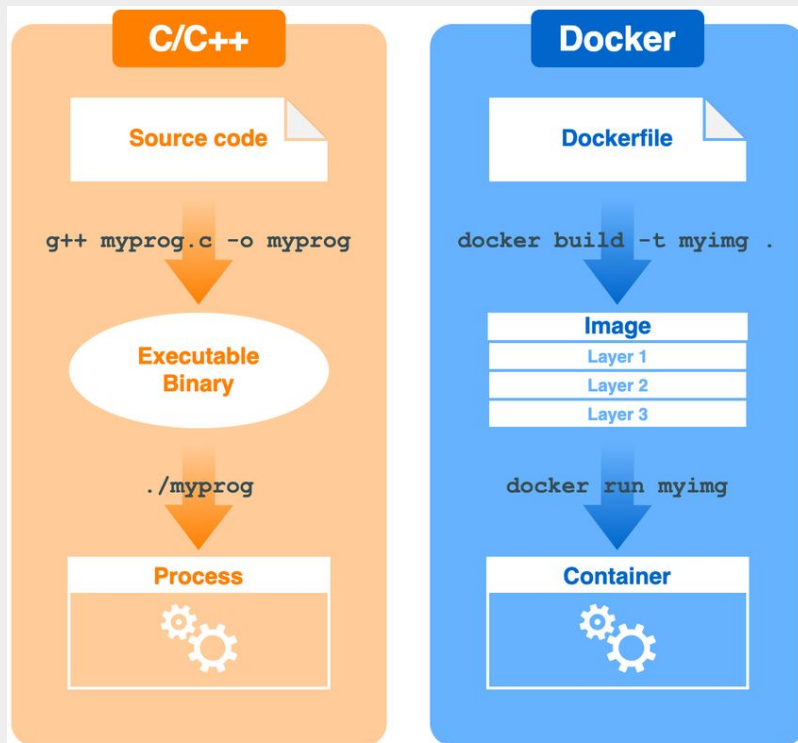
@Solidaritylabs



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.

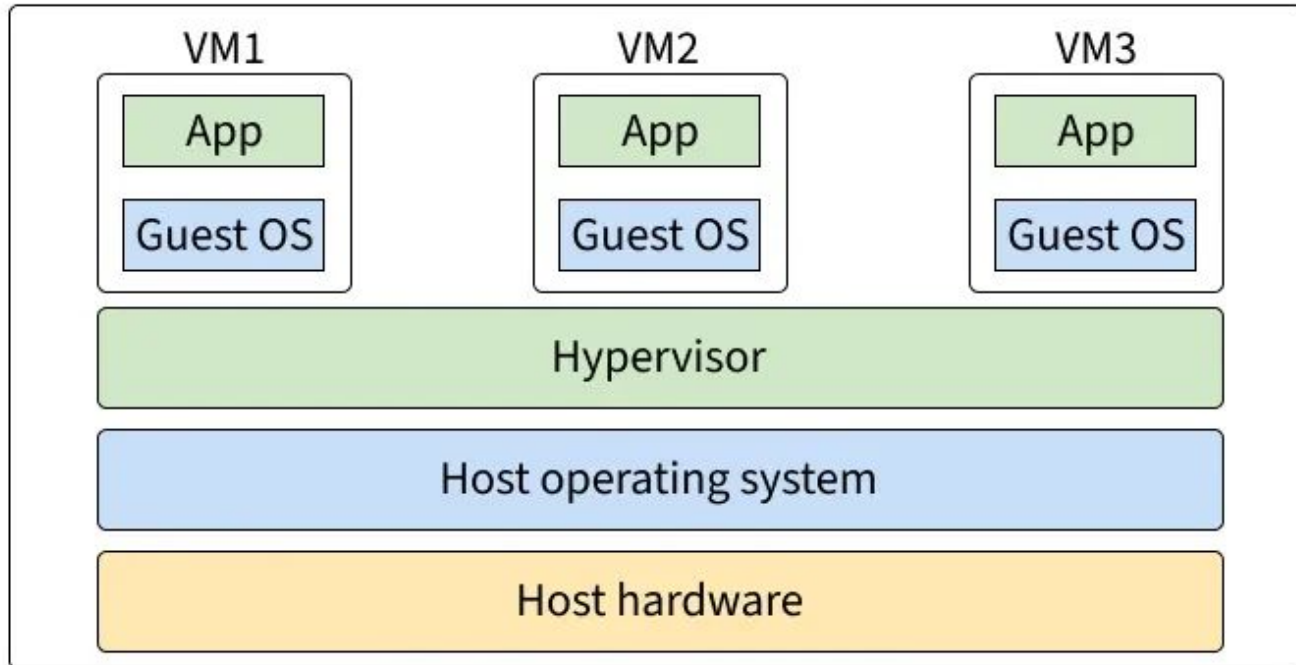
Las soluciones existentes no eran suficientes:

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

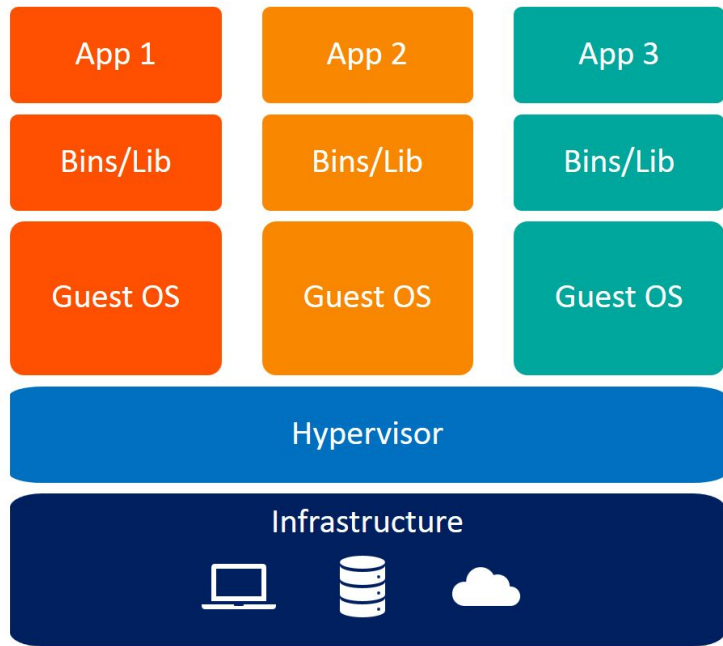
VMs y Contenedores (1)



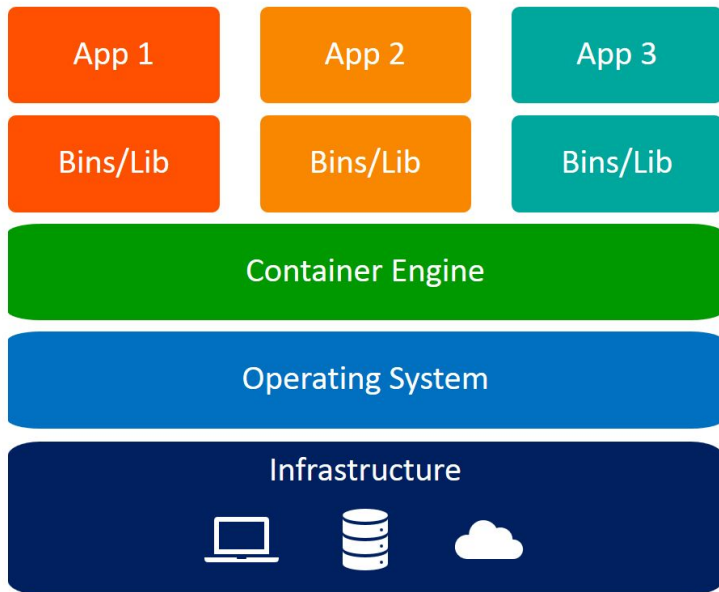
Virtual machines



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)	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



6 Phases of the Software Development Life Cycle





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

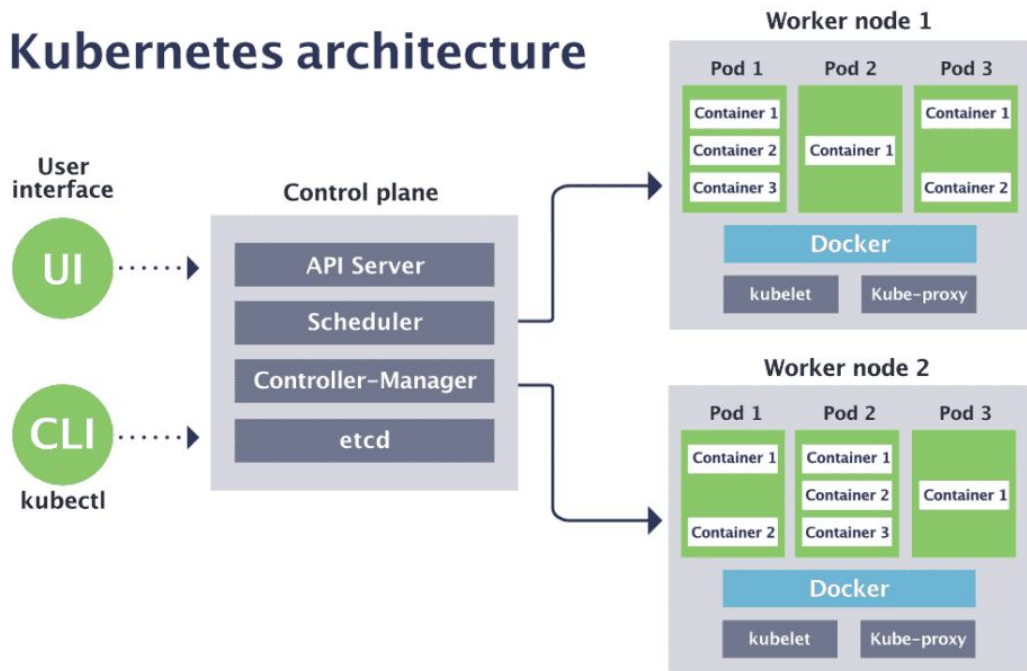
4. **Escalabilidad y arquitecturas modernas**

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

Como se ve el futuro?



Kubernetes architecture



Entonces Docker?



*“Docker nos 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

[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



Pular 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`

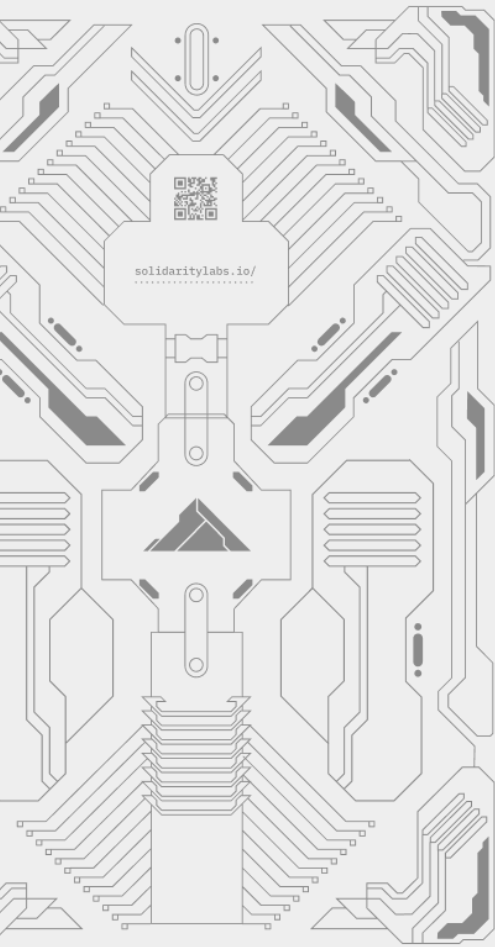
Purgar

- `Docker system prune`



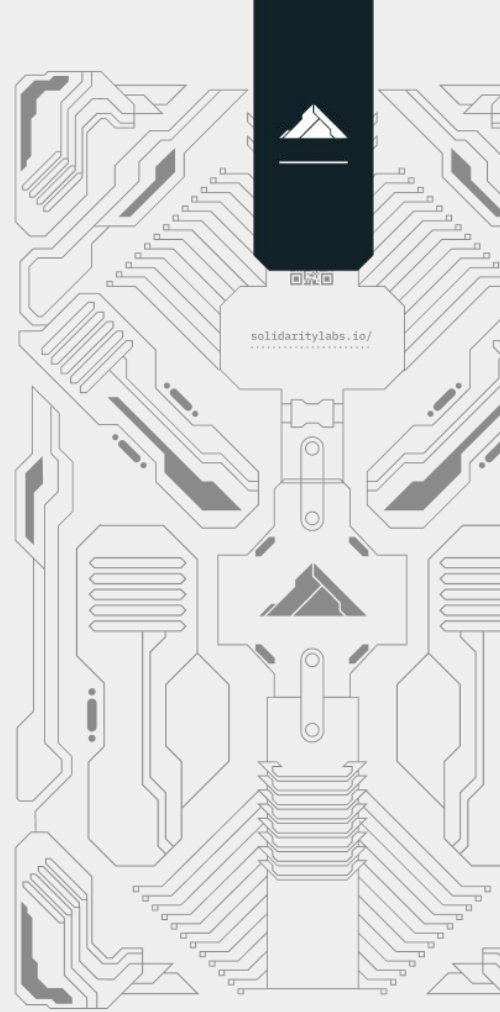
Flags (Docker RUN)

- 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



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

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

@Solidaritylabs

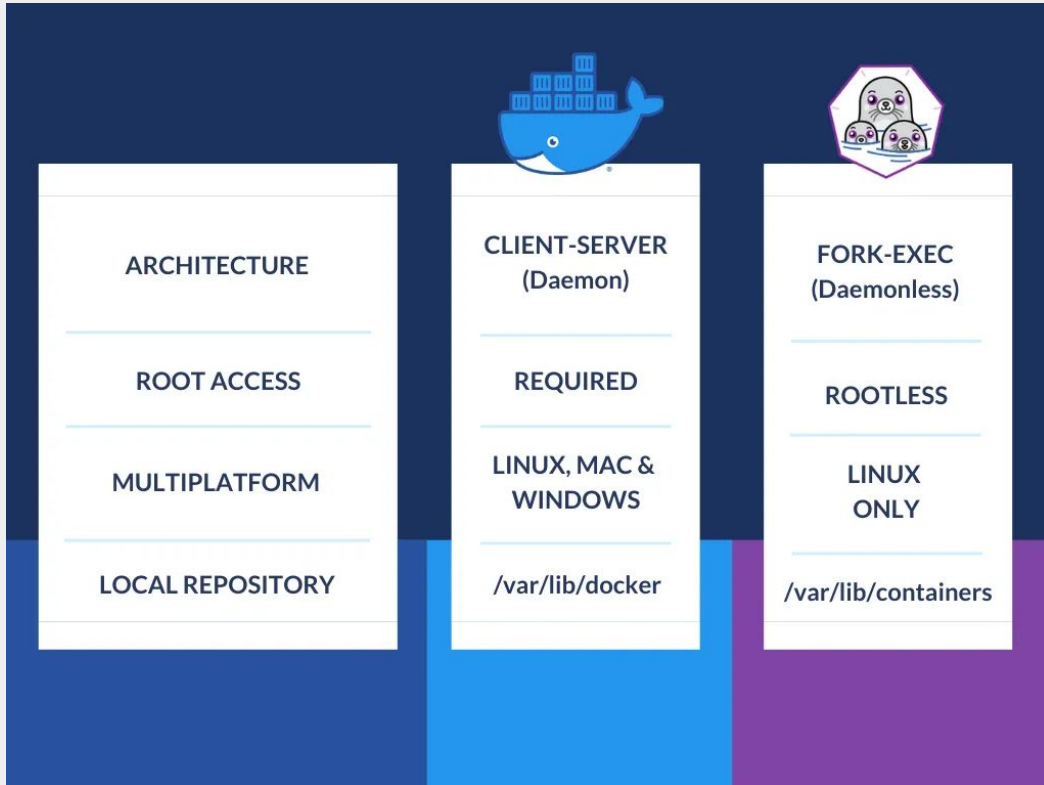




Docker vs Containers (1)

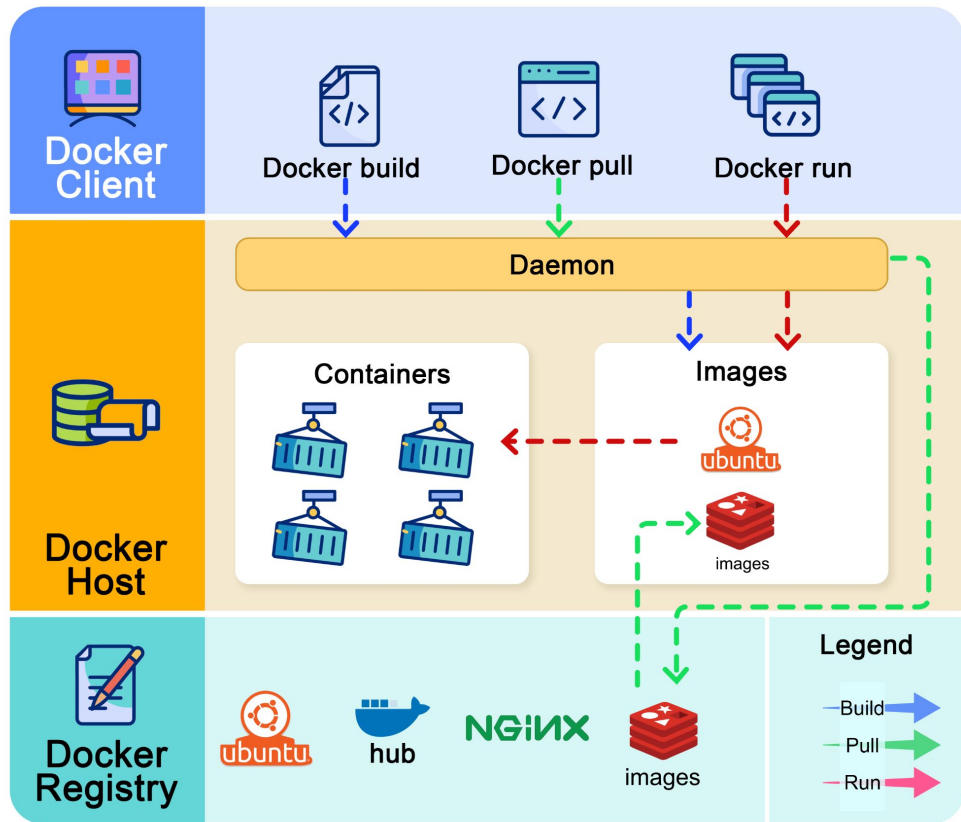
Feature	Docker	Podman	Containerd	CRI-O
Architecture	Client-server (daemon-based)	Daemonless	Low-level runtime (core)	Kubernetes-native
Security	Runs as root	Rootless, daemonless	Secure, low-level	Secure, lightweight
Kubernetes Support	Requires additional integration	Directly compatible	Native integration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

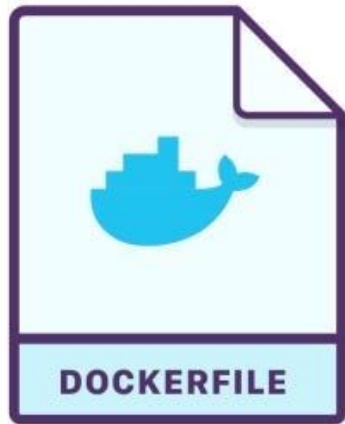


How does Docker Work ?

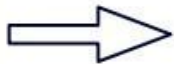
 blog.bytebytego.com



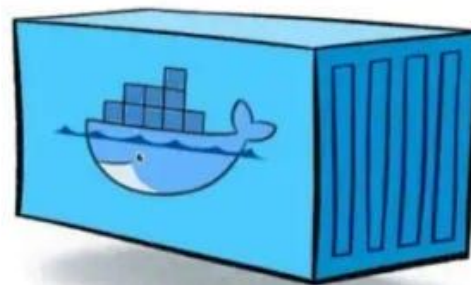
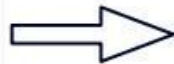
Cómo funciona? (1)



Docker file



Docker Image



Docker Container

Dockerfile - Ejemplo Mínimo



```
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)



≡ commands.txt U

 Dockerfile U ●

conferences > docker >  Dockerfile > ...

```
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"]
```


Optimización



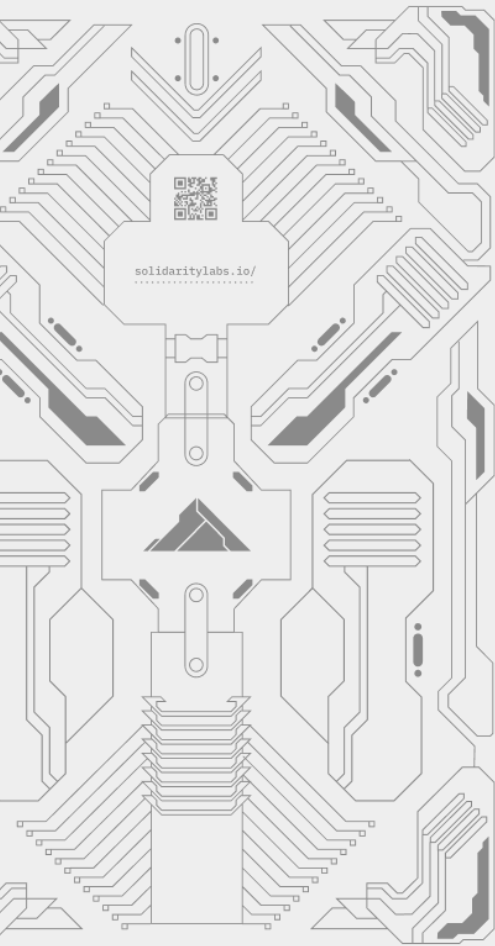
1. Cada línea genera una **LAYER**
2. El orden importa, el layering va de **ARRIBA hacia ABAJO**
3. Siempre poner **las dependencias primero**, o lo que es menos probable que cambie
4. Nunca correr el container como **ROOT**
5. Usar imágenes **mínimas** como ALPINE
6. Usar imágenes **DISTROLESS**, sin 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



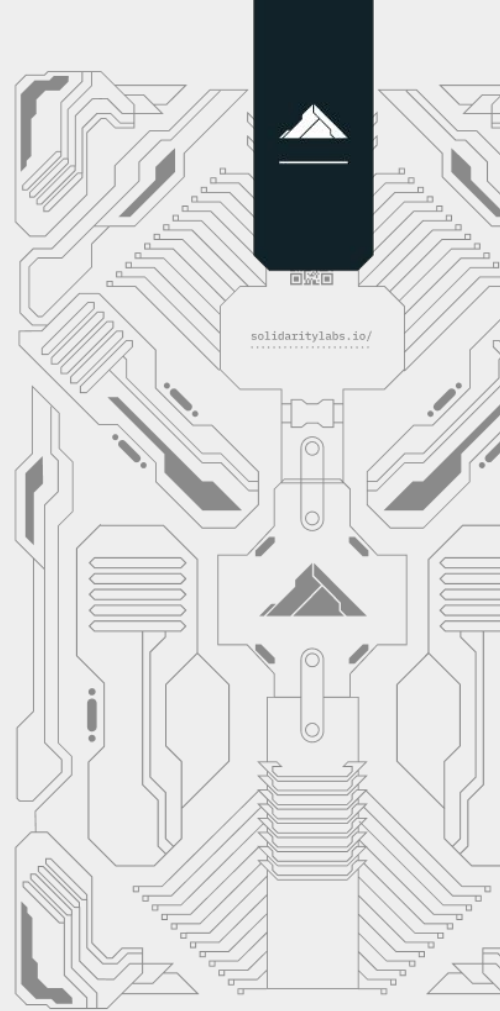
conferences > docker > Dockerfile > ...

```
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

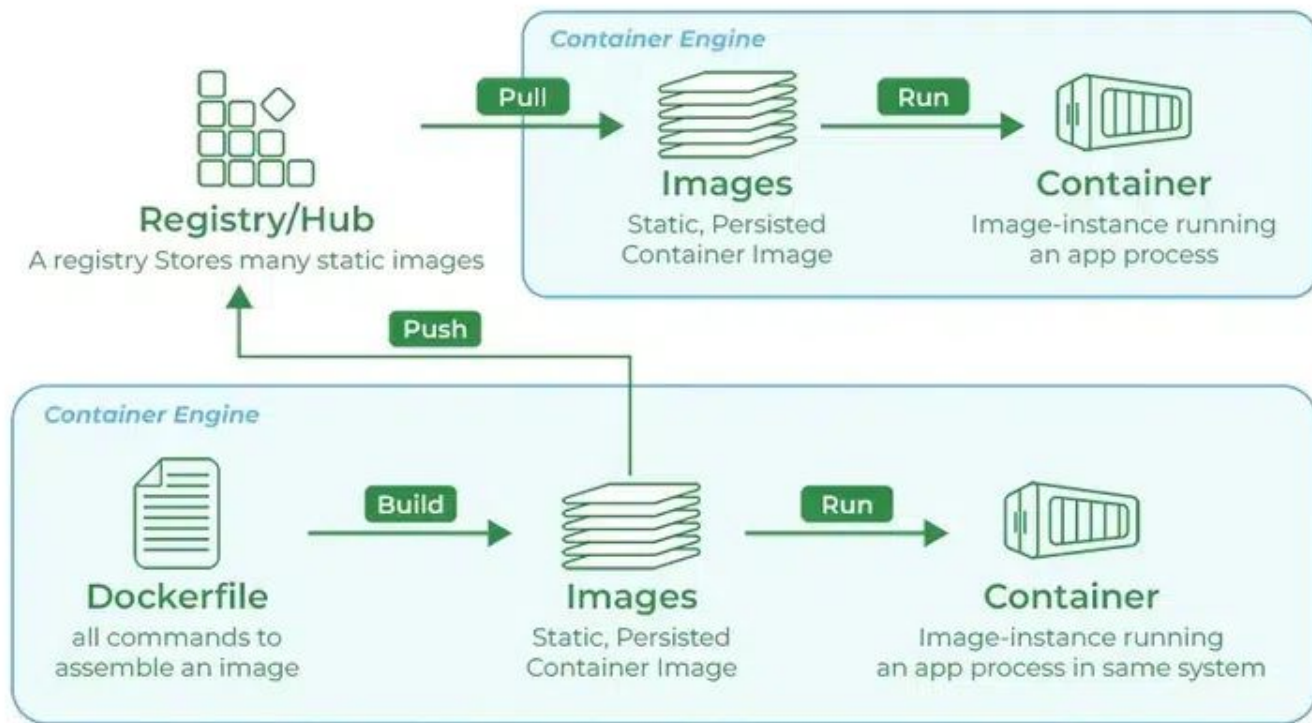




Ejercicio 2 - Docker Images

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)



hub.docker.com/search?q=nginx

hub

nginx

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

nginx Docker Official Images

Official build of Nginx.

Pulls 1B+
Stars 21061
Last Updated 5 days

nginx/nginx-ingress NGINX Inc.

NGINX and NGINX Plus Ingress Controllers for Kubernetes

Pulls 1B+
Stars 110
Last Updated 23 minutes

nginx/nginx-prometheus NGINX Inc.

NGINX Prometheus Exporter for NGINX

Pulls
Stars
Last Updated

nginx/unit NGINX Inc.

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

Pulls 10M+
Stars 66
Last Updated over 2 years

nginx/nginx-ingress-operator NGINX Inc.

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

Pulls 1M+
Stars 2
Last Updated 14 days

nginx/docker-extension NGINX Inc.

Pulls
Stars
Last Updated

Docker Registries (3)



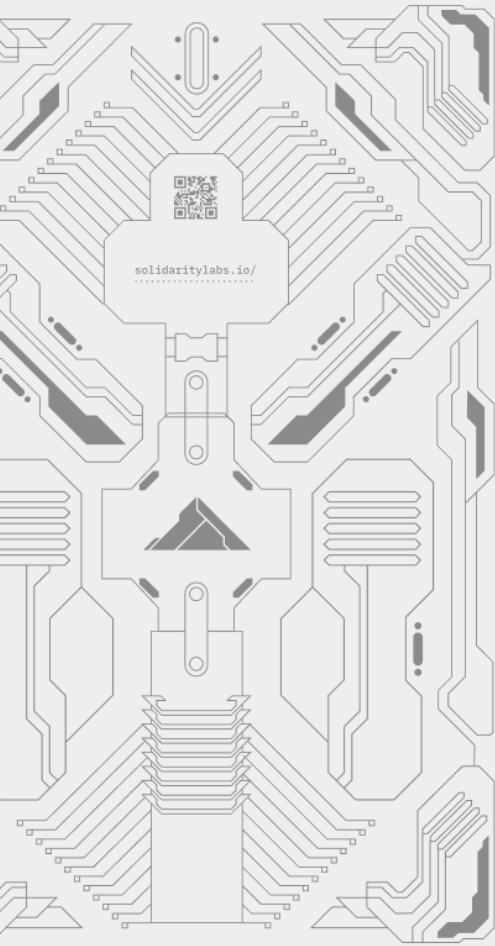
- docker.io/library/nginx:1.25
- `123456789012.dkr.ecr.us-east-1.amazonaws.com/myapp:prod`

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

App: myapp

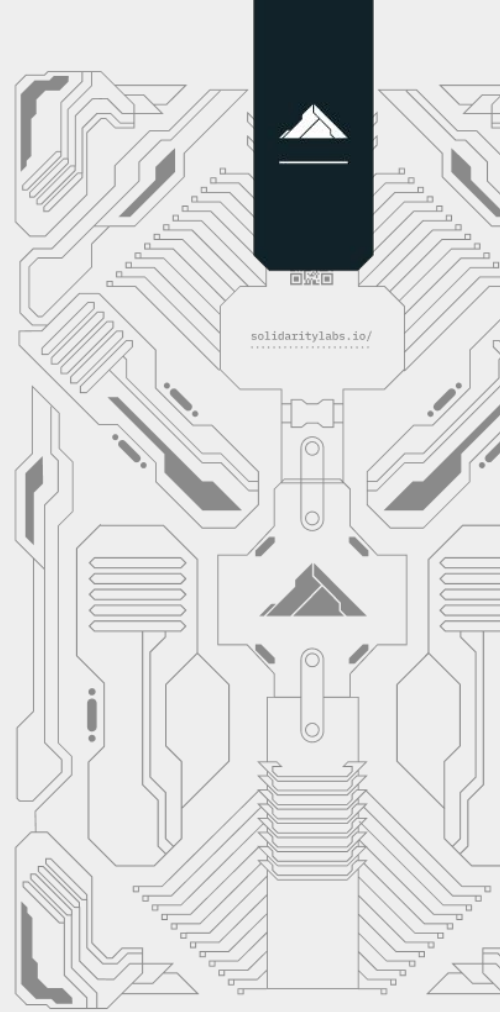
Tag: prod

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 compañero@
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



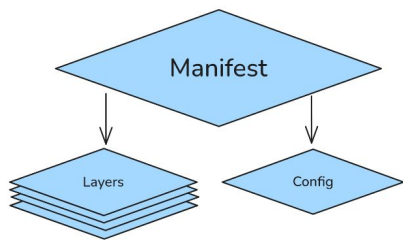
- `denied: requested access to the resource is denied`
→ Error en el nombre de repo / No estás logeado@ .
- `name invalid: ...`
→ Nombre de repo invalido, quizás una mayúscula?.
- `unauthorized: authentication required`
→ Expiró o falta el Token, puede ser que no tengas suficientes permisos

Build de Múltiple Arquitectura

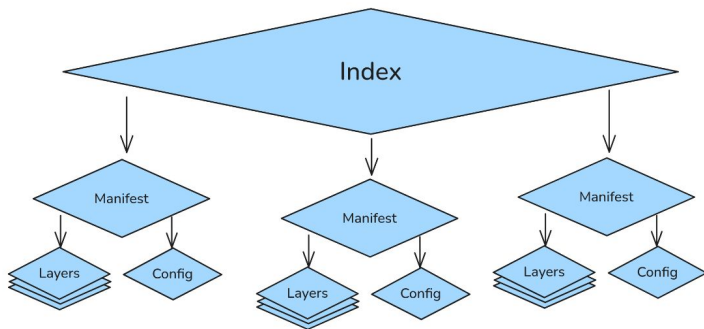


```
docker buildx create --use  
docker buildx build --platform linux/amd64,linux/arm64 \  
-t YOUR_DOCKERHUB_USER/ekohack-api:1.0.0 --push .
```

Single-platform image




Multi-platform image



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

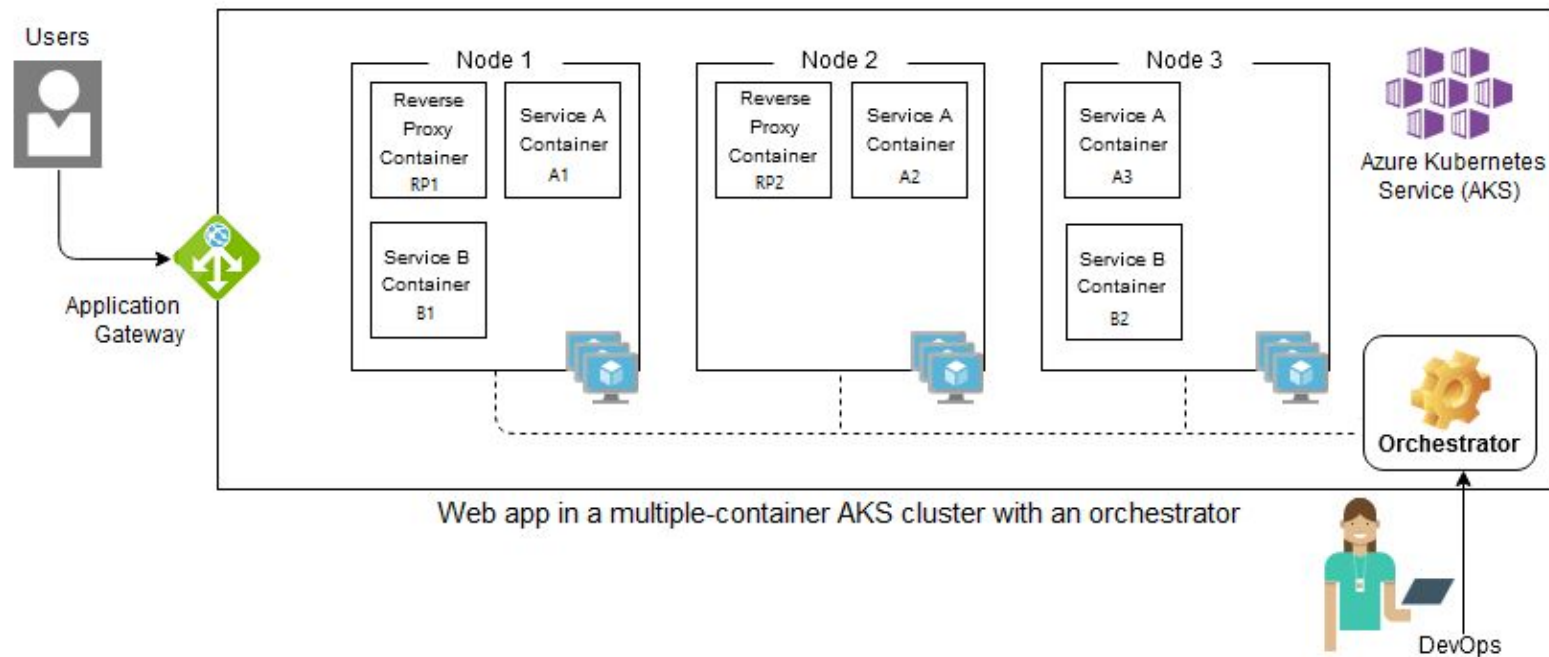
Orquestación con Compose

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





Por qué existen los Orquestadores? (1)





Por qué existen los orquestadores? (2)

Correr containers manualmente es difícil por:

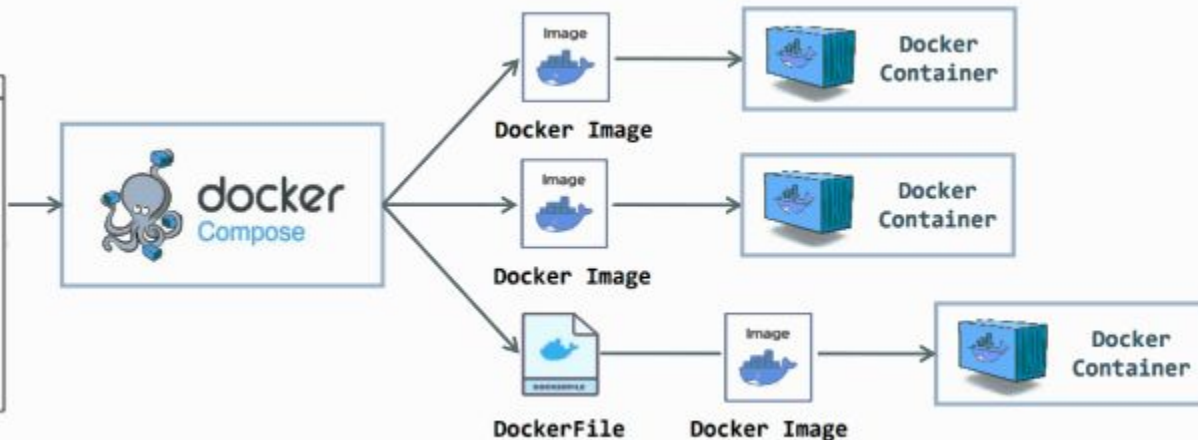
- *Gestionar múltiples servicios (frontend, backend, db);*
- *Administrar restarts automáticos;*
- *Load balancing;*
- *Deploys con Zero-downtime;*
- *Infraestructura distribuida;*
- *Manejo de Secretos;*
- *Rolling updates;*
- *Rollbacks.*

Docker Compose (1)



docker-compose.yml

```
*** docker-compose.yml ***
version: "3.7"
services:
  db:
    image: mysql:5.0.19
    restart: always
    environment:
      - MYSQL_DATABASE=example
      - MYSQL_ROOT_PASSWORD=password
  app:
    build: app
    restart: always
  web:
    build: web
    restart: always
    ports:
      - 80:80
```



Docker Compose (2)



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`

```
compose.yaml U x
conferences > docker > docker_compose > compose.yaml
1  version: "3.9"
   ↳ Run All Services
2  services:
   ↳ Run Service
3    api:
4      image: ghcr.io/example/api:1.0.0
5      ports:
6        - "8080:8080"
7      environment:
8        - DATABASE_URL=postgres://user:pass@db:5432/app
9      depends_on:
10       - db
11
12   ↳ Run Service
13   db:
14     image: postgres:16
15     environment:
16       - POSTGRES_PASSWORD=pass
17     volumes:
18       - db_data:/var/lib/postgresql/data
19
20 volumes:
21   db_data:
```

Docker Compose (3)



```
conferences > docker > docker_compose > ! compose2.yaml
```

```
1  version: "3.9"
2
3  services:
4    nginx:
5      image: nginx:1.25
6      ports:
7        - "8080:80"
8      volumes:
9        - ./nginx.conf:/etc/nginx/nginx.conf:ro
10     depends_on:
11       - api
12
13   api:|
14     build: ./api
15     environment:
16       - DB_HOST=db
17       - DB_USER=app
18       - DB_PASSWORD=pass
19     ports:
20       - "3000:3000"
21     healthcheck:
22       test: ["CMD", "curl", "-f", "http://localhost:3000/health"]
23       interval: 10s
24       timeout: 3s
25       retries: 3
26
27   db:
28     image: postgres:16
29     environment:
30       - POSTGRES_PASSWORD=pass
31     volumes:
32       - pgdata:/var/lib/postgresql/data
33
34   volumes:
35     pgdata:
```

Comandos:

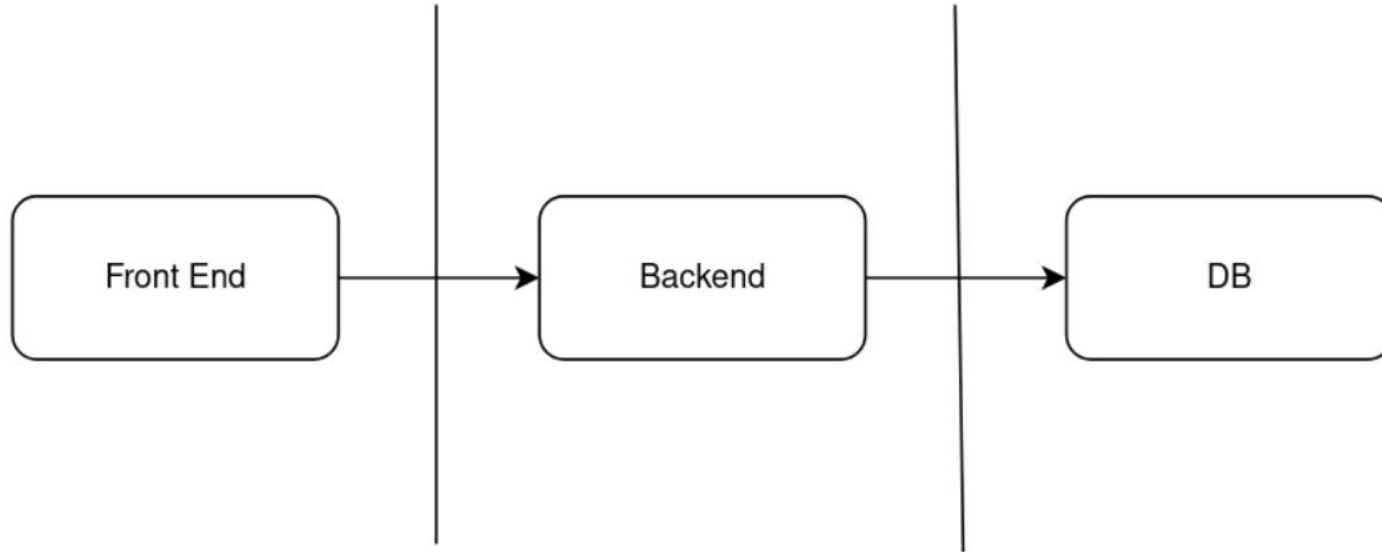
- *docker compose up*
- *docker compose up -d*
- *docker compose down*
- *docker compose ps*
- *docker compose logs -f*
- *docker compose exec api sh*

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
- Manifestos
- Networking
- Volumes



Kubernetes (1)



Por que Kubernetes?

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

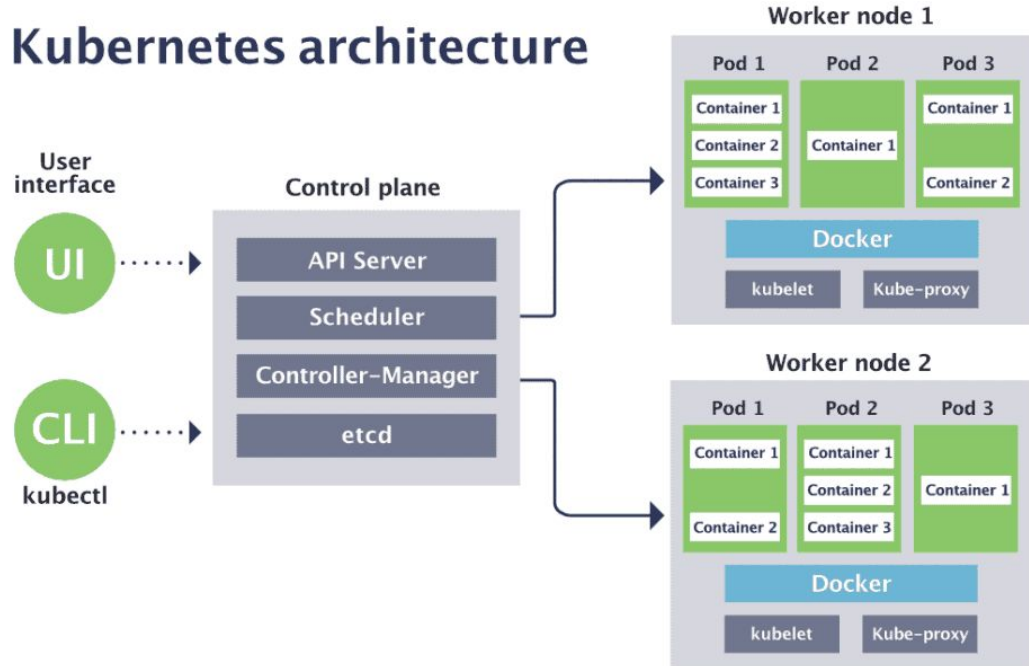
Regla de 3 simple:

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

Kubernetes (2)



Kubernetes architecture



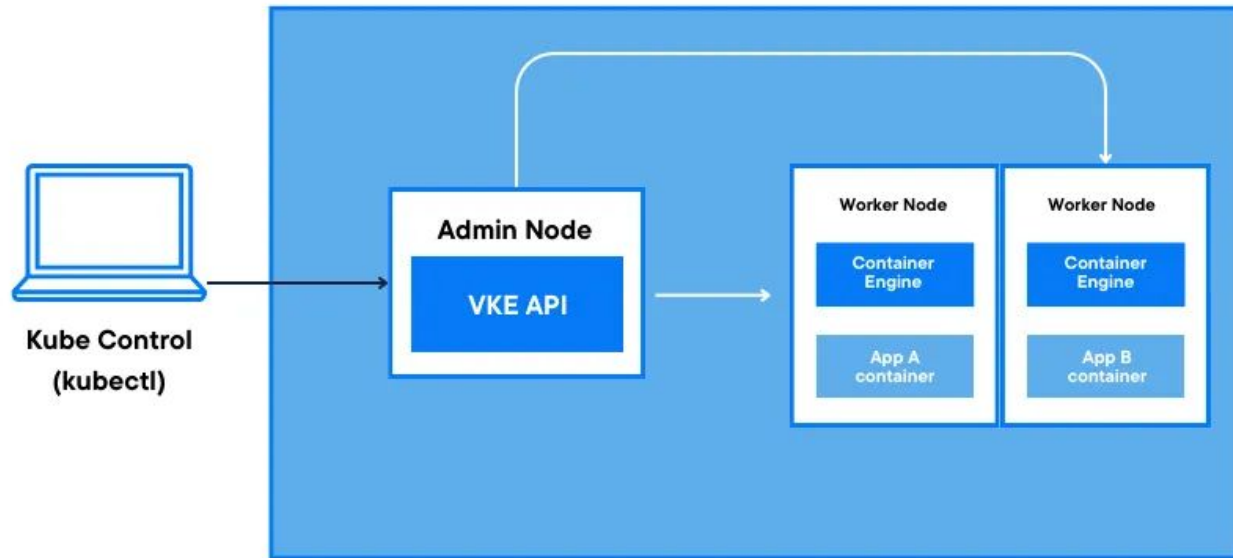
Kubernetes (3)



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

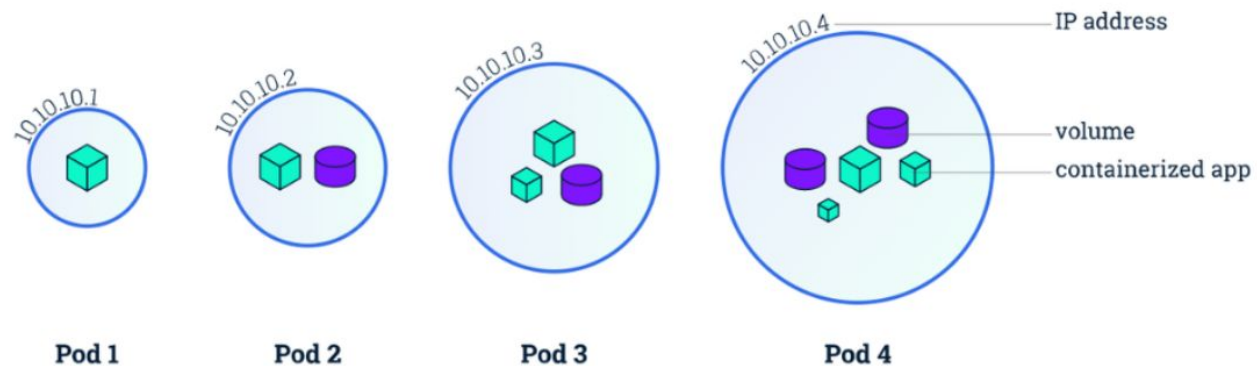


kubeconfig

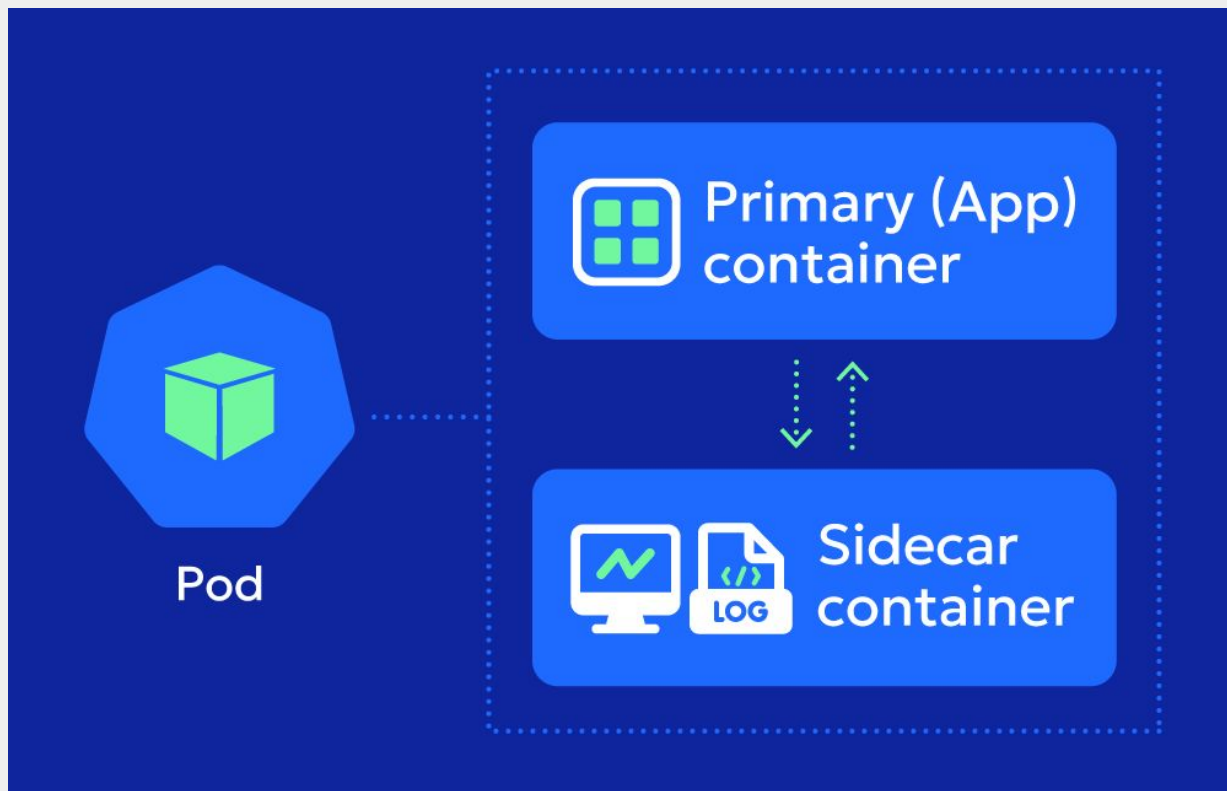


```
santi@fedora:~$ cat .kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0t
nc2T3U3NHN3RFFZSkvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhNS2EzVm1aV
F4TXpGYUZ3MHP0VEV4TWpNeE5UQTJNekZhTUJVeApFekFSQmd0VkJBTVRDbXQxWW1WeV
RRUJBUBVBQTRJQkR3QXdnZ0VLCkFvSUJBUUN4SnhGNzF1aGJDc3N0OUtFU2RmVVZYY09
QkNoT1kwT28KTjhuN2VCbjJvbXhTK3B2a3RTRVNYN2xsM2NIbDhJZGRKbk1KbWZIdG5T
WYweU0vbFhUMGNNa1NBT3h5WEZzYThLeTBacDV0RW0rcjcwbnDNvTHRlUEkxM04xcGZDQ
dIQXBibGRpSDJGMGptZm5ScndCb08vYXBiaGdtendtR1BPMFptMlozcGVZWUNXRUpTW
Qb2V6TzB2d3kyR0lDNW12SXBRCUkra1ZDTG90Y2hkRGVuZUEzZ0RYSQpZYmw3dzFWZWd
R2pXVEJYTUE0R0ExVWREd0VCL3dRRUF3SUNwREFQckJnTlZIUK1CQWY4RUJUQURBUUgv
nNFdzhpWXRFa0RxSE5FRkVUQVYKQmd0VkhSRUVEakFNZ2dwcmRXSmxjbTVsZEdWek1BM
EycCs0VGhPcQpUdFYzRzJ2ZW5PMG1CVFdFSTRZSGdFZFRKTVJQNFJBZCtMeG9GMjZ4a2
2cm5oU0ErM1K3FuVEVQUjVM1NwMlI3N1NvMEExXM09QZmVWVWnBxYVhQTzZmUGxsYVY
SzR00E9hY0ZiNm16S2FPZHFue1pneFRnUE5VZUk0L2RnbVo4L2swMUJmUkd0UklQT1VF
TF1WkcyTTBwBkFYT0lRbTZGTGFBZ0HA1RVRFM25hdkFQdmVqeG9GWlVkcKlaNG5RN1JNT
M1L1F5QW1iD3JlM1NwMlI3N1NvMEExXM09QZmVWVWnBxYVhQTzZmUGxsYVY
SzR00E9hY0ZiNm16S2FPZHFue1pneFRnUE5VZUk0L2RnbVo4L2swMUJmUkd0UklQT1VF
TF1WkcyTTBwBkFYT0lRbTZGTGFBZ0HA1RVRFM25hdkFQdmVqeG9GWlVkcKlaNG5RN1JNT
```

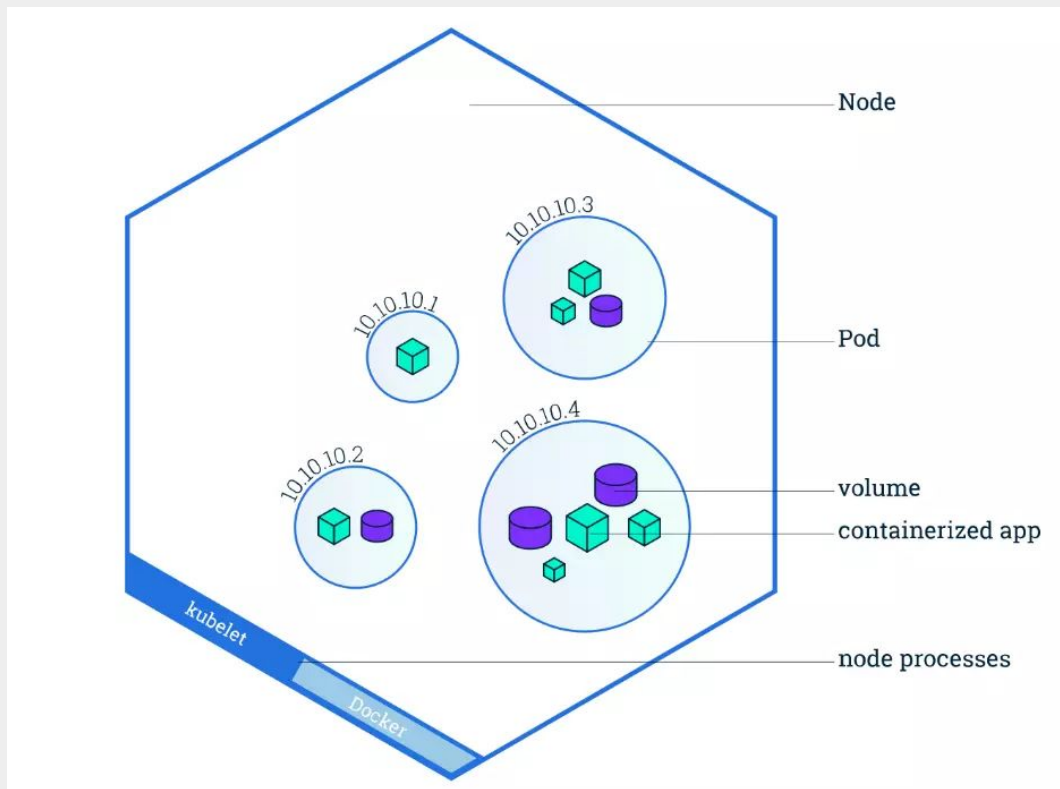
Pods (1)



Pods (2)



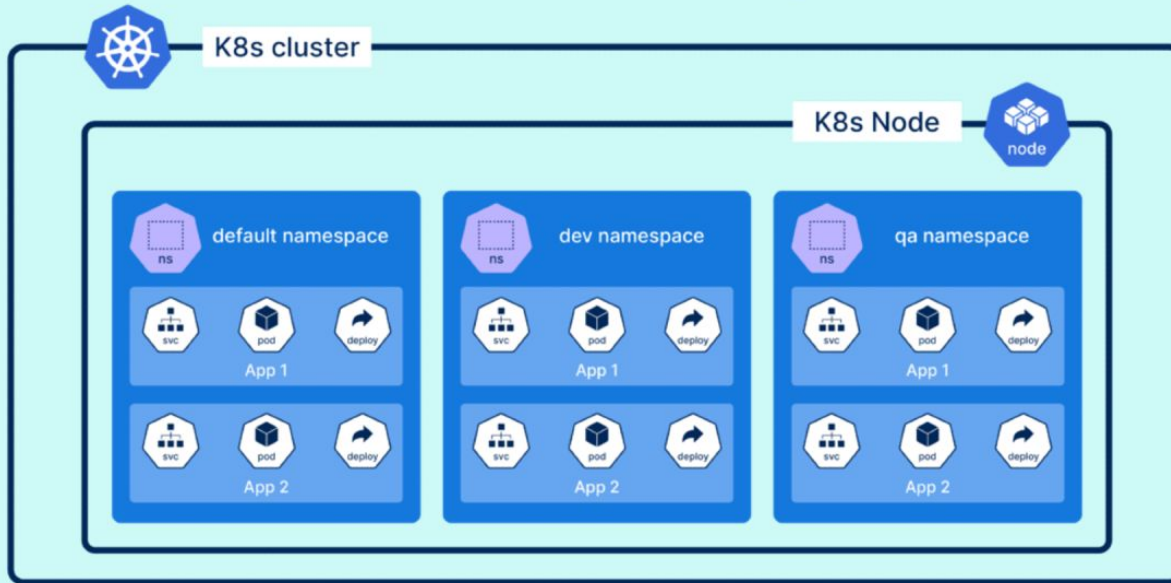
Nodes



Namespaces

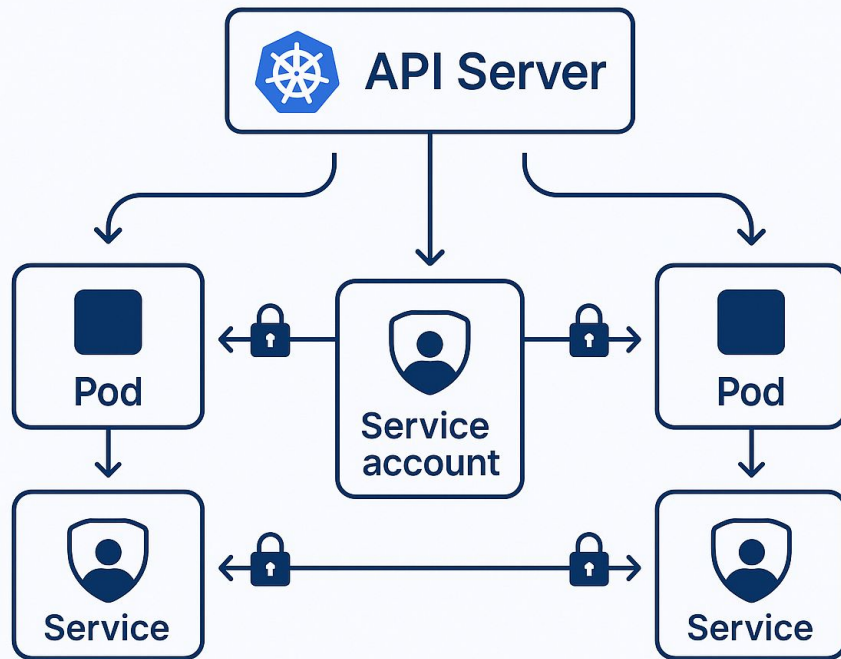


Kubernetes - Namespaces



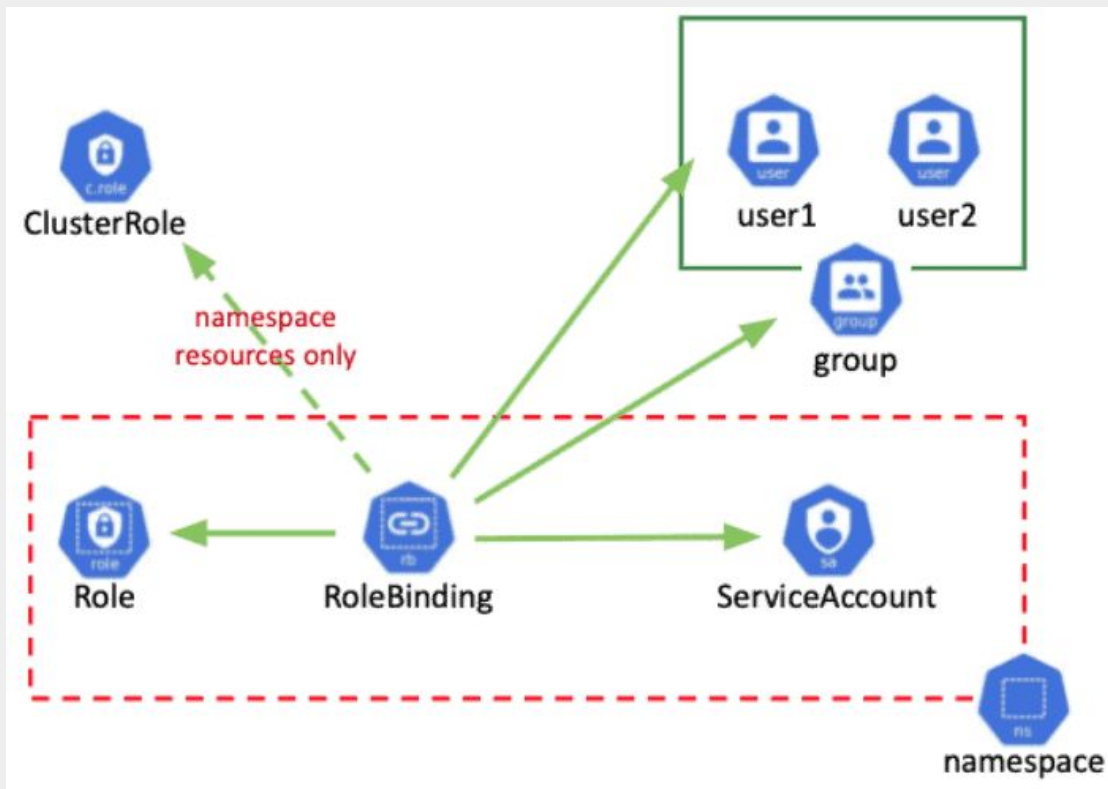


Identity and Access Management (1)





Identity and Access Management (2)





Identity and Access Management

```
1  apiVersion: rbac.authorization.k8s.io/v1
2  kind: Role
3  metadata:
4    name: team-a-readwrite
5    namespace: team-a
6  rules:
7    - apiGroups: [""]
8      resources: ["pods"]
9      verbs: ["get", "list", "watch", "create", "delete"]
```

Manifiestos!



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

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: nginx
10   template:
11     metadata:
12       labels:
13         app: nginx
14     spec:
15       containers:
16         - name: nginx
17           image: nginx:stable
18           ports:
19             - containerPort: 80
20   ---
21   apiVersion: v1
22   kind: Service
23   metadata:
24     name: nginx-service
25   spec:
26     selector:
27       app: nginx
28     ports:
29       - port: 80
30       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)



```
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

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)



yaml

 Copy code

```
apiVersion: v1
kind: Service
metadata:
  name: demo-nginx-nodeport
  namespace: demo-net
spec:
  type: NodePort
  selector:
    app: demo-nginx
  ports:
    - port: 80          # service port
      targetPort: 80
      nodePort: 30080   # external node port (30000-32767)
```


Networking (2.2)



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
```

```
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

```
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

```
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
```

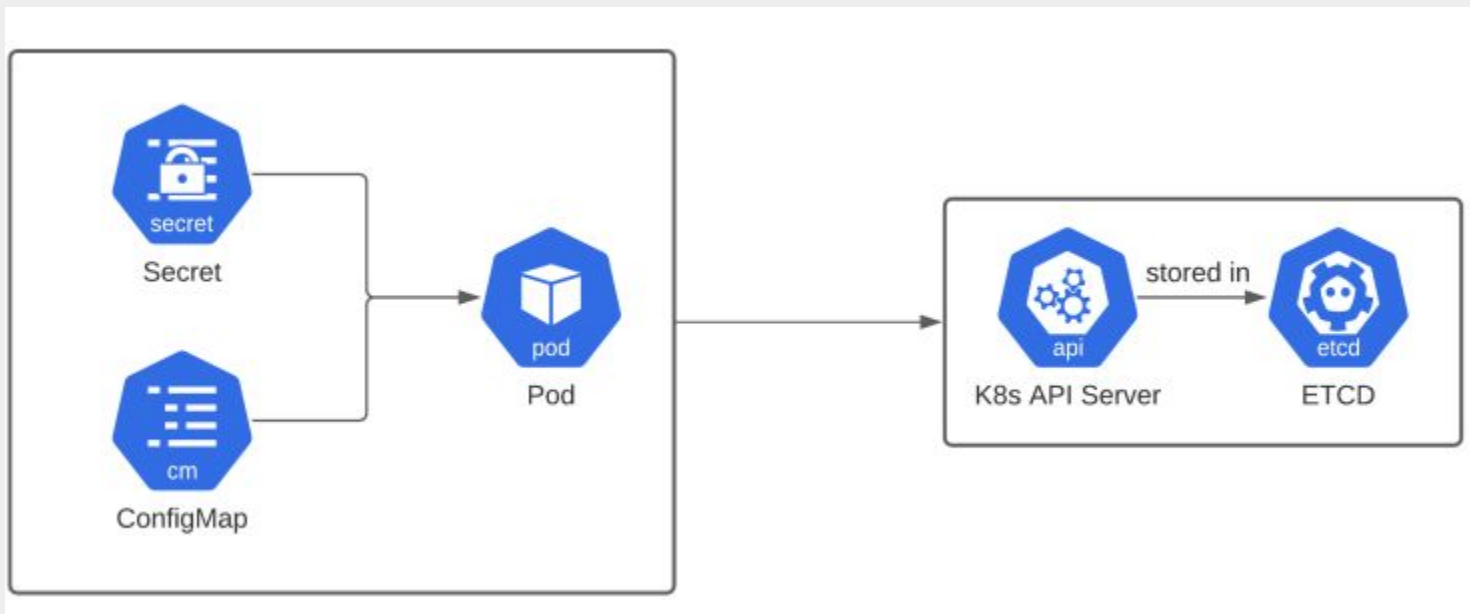
Volumes

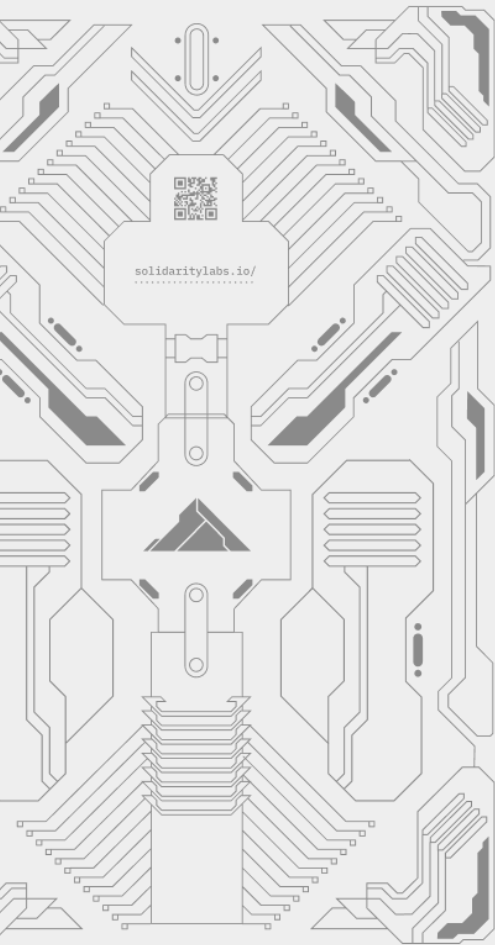


- **Ephemeral Volumes** (Se pierde cuando el pod muere)
 - emptyDir
 - Compartido entre containers
 - Configmap/Secrets
 - Se usan para configuraciones
 - downwardAPI
 - Expone la metadata de la API
- **Persistent Volumes**
 - El storage del cluster
-

- **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

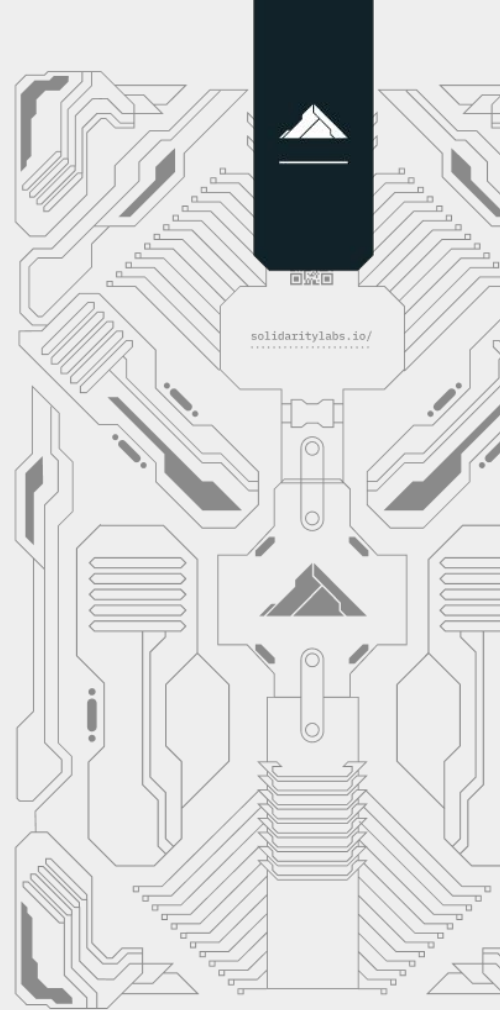
Configmaps / Secrets





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. Securitizar el API Server
8. Usar herramientas de seguridad en runtime - AKA Falco



Ejercicio 6.1 - Kubernetes RBAC

1. Levantar tu cluster de k8s con minukube
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.

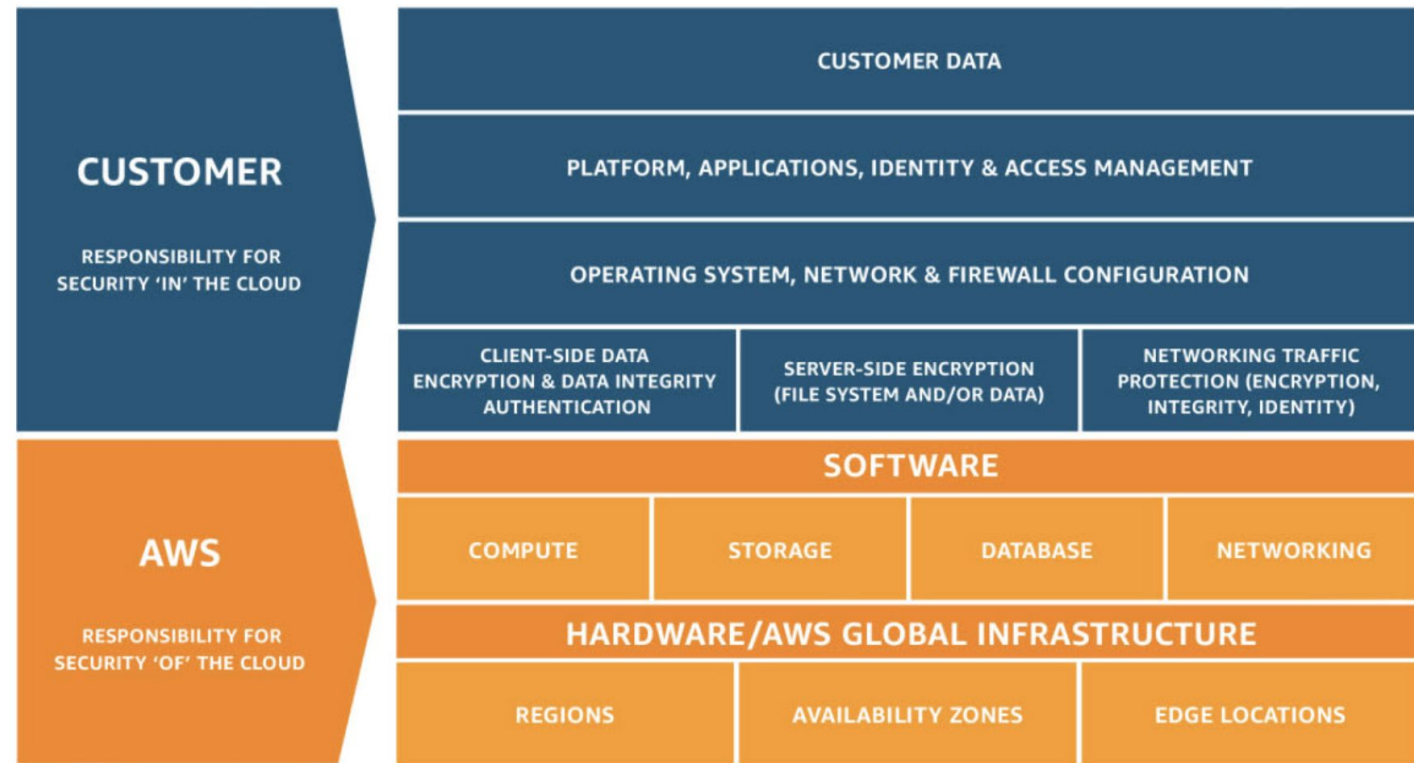
/Capítulo \5

Orquestando Containers en AWS - ECS

- Modelo de responsabilidad compartida
- Terraform



Share Responsibility Model



Terraform



DOCUMENTS

conferences

Trainings

terraform

ecs

admin_app

mobile_app

.terraform.lock.hcl

alb.tf

aws_secrets_manager.tf

ecr.tf

ecs.tf

iam.tf

locals.tf

main.tf

outputs.tf

s3.tf

secrets_ctf_dev_admin.json

secrets_ctf_dev_mobile.json

security_groups.tf

terraform.tfstate

terraform.tfstate.backup

variables.tf

conferences > Trainings > terraform > ecs > ecs.tf > resource "aws_ecs_service"

```
1 resource "aws_ecs_cluster" "main" {
2   name = "${var.product_name}-services-ecs-cluster"
3 }
4
5 resource "aws_ecs_task_definition" "task_definition" {
6   for_each           = var.services
7   family             = each.key
8   network_mode       = "awsvpc"
9   requires_compatibilities = ["FARGATE"]
10  cpu                 = each.value.cpu
11  memory              = each.value.memory
12  task_role_arn       = aws_iam_role.ecs_task_execution_role.arn
13  execution_role_arn  = aws_iam_role.ecs_execution_role.arn
14
15  container_definitions = jsonencode([
16    {
17      name       = each.key
18      image      = each.value.image
19      essential  = true
20      portMappings = [
21        {
22          containerPort = each.value.port
23          protocol      = "tcp"
24          name          = "${each.key}-sc"
25        }
26      ]
27      environment = [
28        {
29          name  = "ENVIRONMENT"
30          value = "dev"
31        }
32      ]
33    }
34  ])
35 }
```

/the\ end/

Preguntas?

Happy to answer any questions!

Contact Info:

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

@Solidaritylabs

