Introducción a Docker

#WLPCMexico2022

Sección 1:

Qué es Docker

Qué NO es Docker

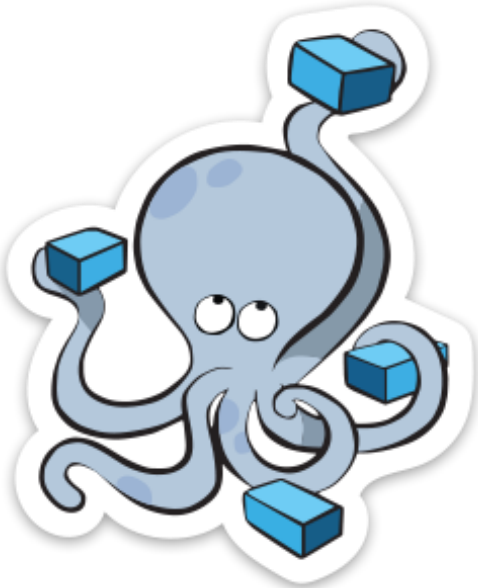
Comandos básicos

Dockerfiles

Sección 2:

Anatomía de una imagen

Volúmenes Docker



Sección 3:

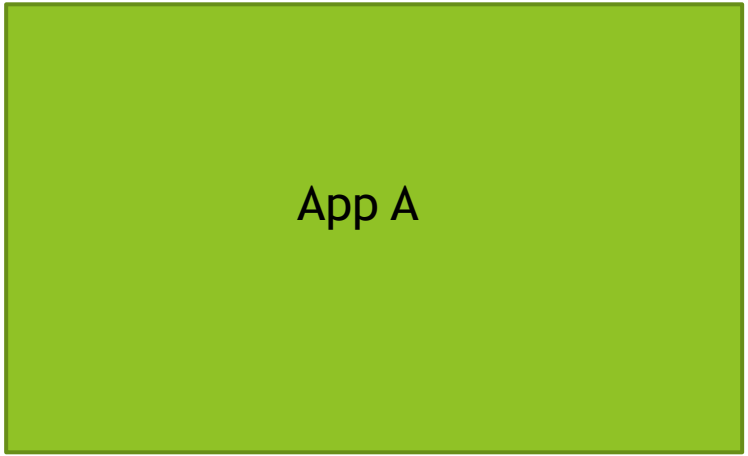
Networking

Sección 4:

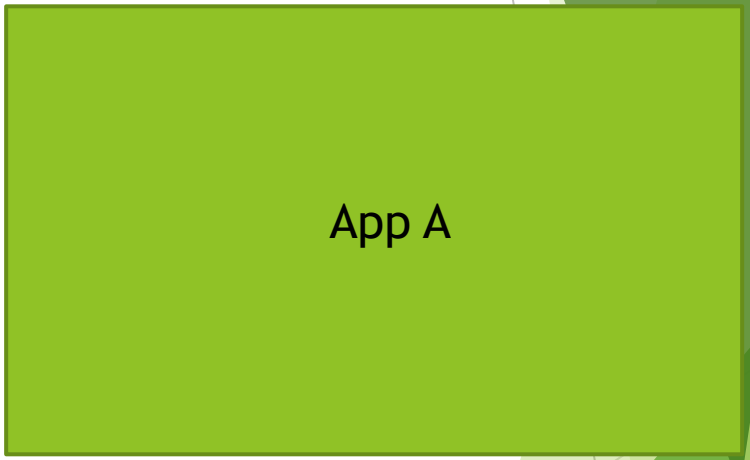
Docker compose / stacks

Demo

Objetivo



Maquina programador/Entorno desarrollo

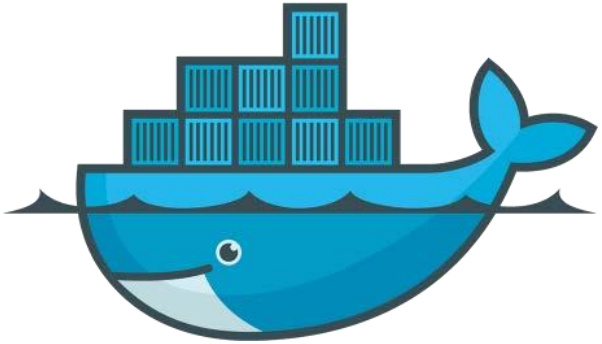


Servidor/Entorno producción

Sección 1:

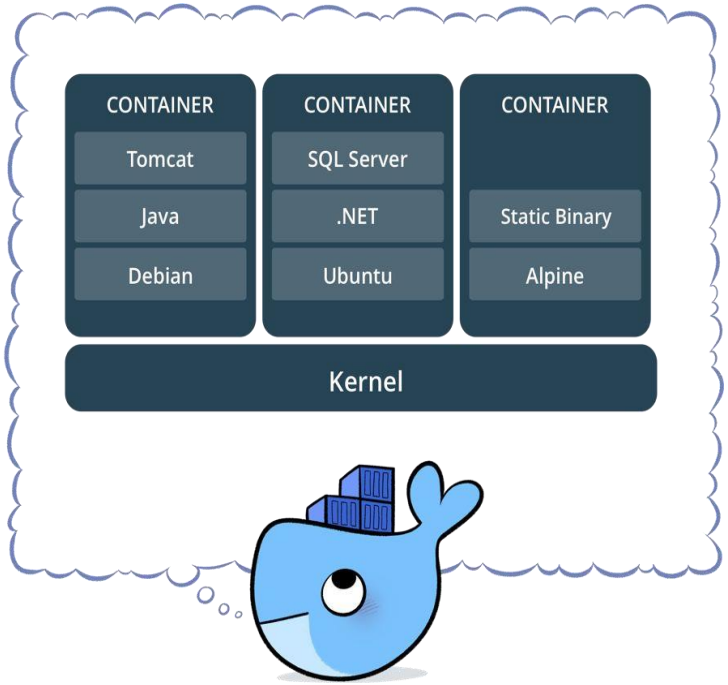
Qué es Docker
Comandos básicos
Dockerfiles

¿Qué es Docker?



- Plataforma ligera, abierta y seguram
- Simplifica la construcción, despliegue y ejecución de aplicaciones
- Corre nativamente bajo Linux o Windows Server (>2016)
- Corre en máquinas de desarrollo Windows o Mac (a través de una VM)
- Depende de "imágenes" y "contenedores"

¿Qué es un contenedor?



- Empaquetamiento estandarizado de software y dependencias
- Aislamiento entre aplicaciones
- Todos los contenedores comparten el mismo núcleo del SO
- Funciona en todas las distribuciones principales Linux
- Funciona de manera nativa a partir de Windows Server 2016

El rol de las Imágenes y los Contenedores

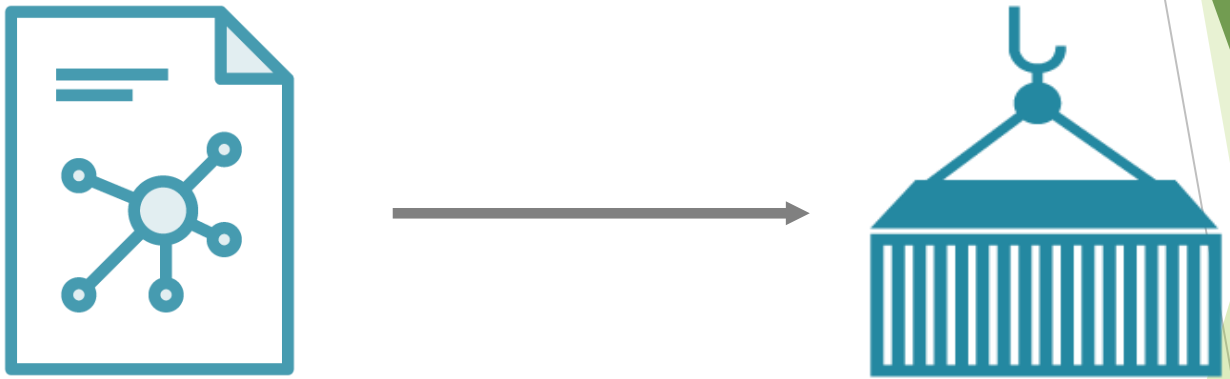


Imagen Docker

Ejemplo: Ubuntu con Node.js y código de aplicación

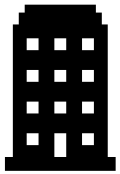
Contenedor Docker

Se crea a partir de una imagen.
Ejecuta una instancia de tu aplicacion.

Los contenedores NO SON VMs

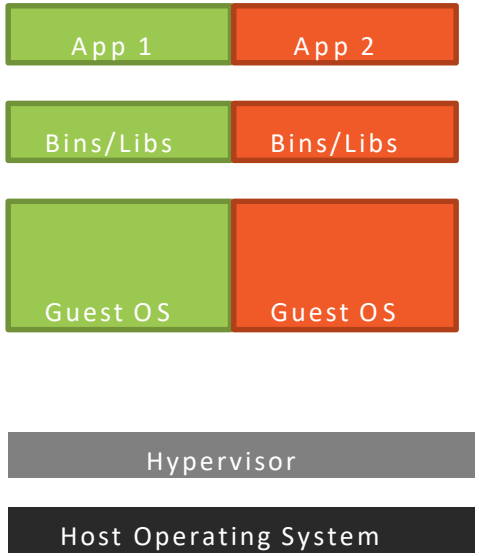
- Es fácil que nos confundamos
- Arquitecturas diferenciadas
- Ventajas diferentes

Maquina Virtual

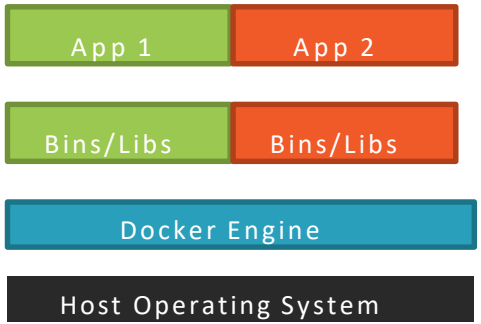


Contenedores

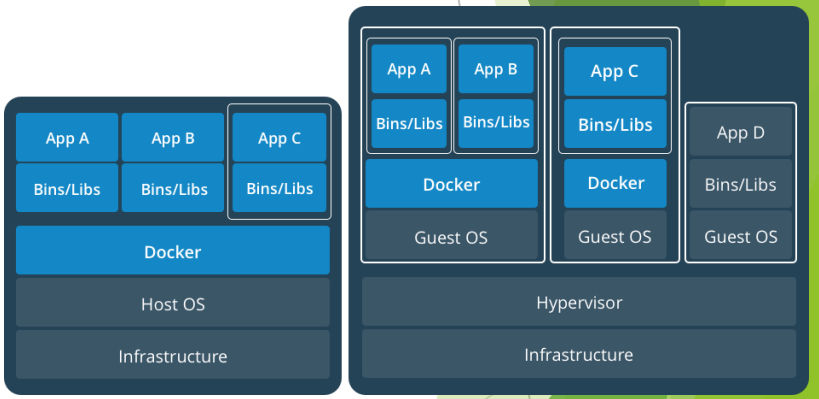
Contenedores VS Máquinas Virtuales



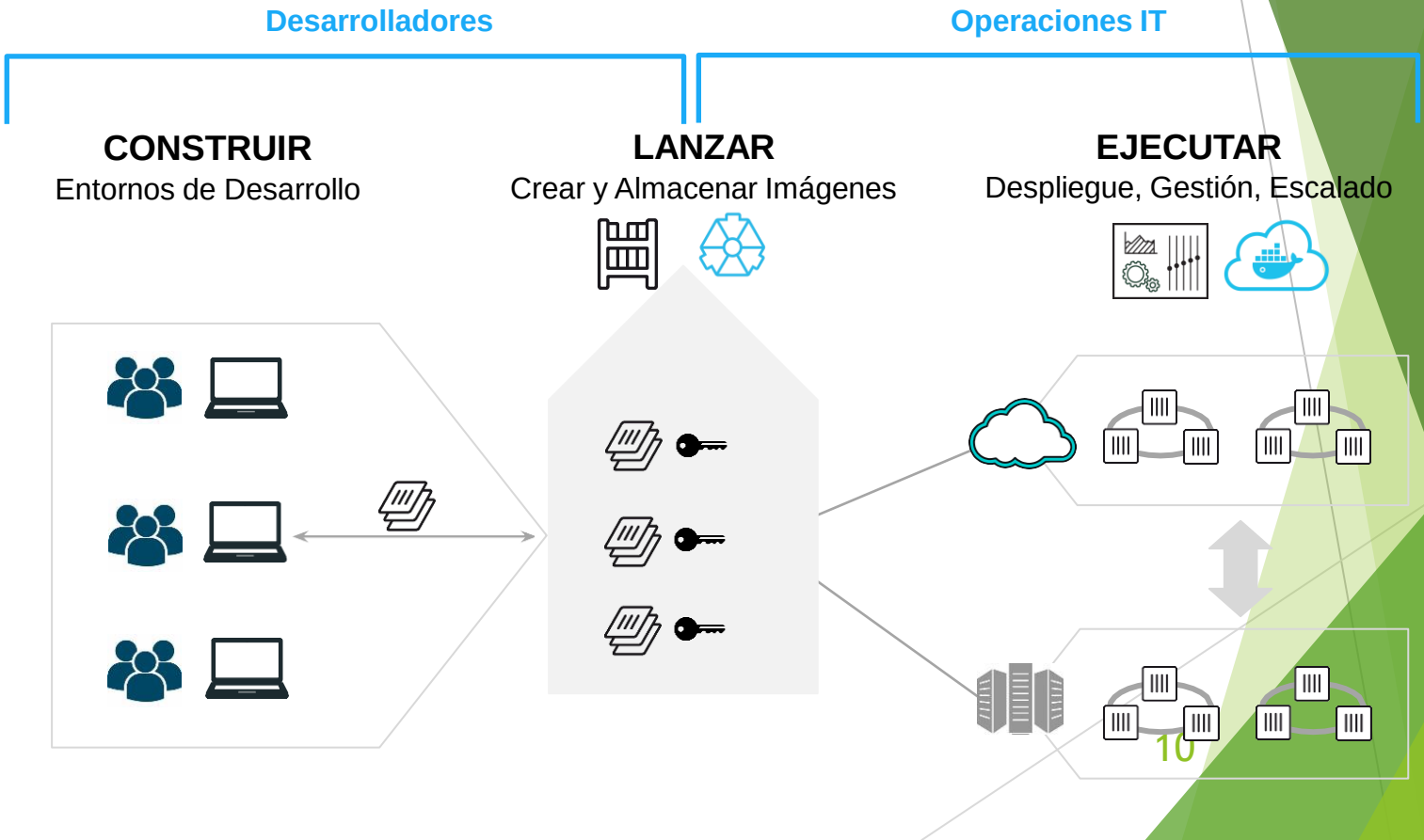
Virtual Machines



Docker Containers



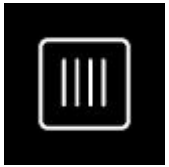
Usando Docker: Construir, Lanzar, Ejecutar Flujo de Trabajo



Vocabulario Docker

Imagen Docker

La base de un contenedor. Representa una aplicación completa



Contenedor Docker

Unidad estándar en la que reside y se ejecuta la aplicación

Motor Docker

Crea, lanza y ejecuta contenedores Docker desplegables en un host local, virtual, en un datacenter o en un proveedor Cloud

Servicio de Registro (Docker Hub (Público) o Docker Trusted Registry(Privado))

Almacenamiento basado en nube o local y servicios de distribución de tus imágenes

Comandos Básicos Docker

```
$ docker image pull node:latest    //  descarga imagen del repositorio

$ docker image ls  // lista imágenes almacenadas
$ docker container run –d –p 5000:5000 –-name node node:latest
// ejecuta un contenedor en segundo plano, enlaza puertos, nombre e imagen

$ docker container ps // lista contenedores en ejecución


$ docker container stop node(or <container id>) // detiene un contenedor

$ docker container rm node (or <container id>) // elimina un contenedor

$ docker image rmi (or <image id>) // elimina una imagen

$ docker build –t node:2.0 . // construye una imagen (normalmente de un Dockerfile)

$ docker image push node:2.0  // sube una imagen al repositorio

$ docker –help  // ver la ayuda
```

Dockerfile - Ejemplo

Dockerfile

```
# Create image based on the official Node 6 image from dockerhub
FROM node:latest

# Create a directory where our app will be placed
RUN mkdir -p /usr/src/app

# Change directory so that our commands run inside this new directory
WORKDIR /usr/src/app

# Copy dependency definitions
COPY package.json /usr/src/app

# Install dependecies
RUN npm install

# Get all the code needed to run the app
COPY . /usr/src/app

# Expose the port the app runs in
EXPOSE 4200

# Serve the app
CMD ["npm", "start"]
```

- Instrucciones para construir una imagen Docker
- Sintaxis similar a comandos Linux
- Es importante optimizar el Dockerfile

Sección 2:

Anatomia de un contenedor

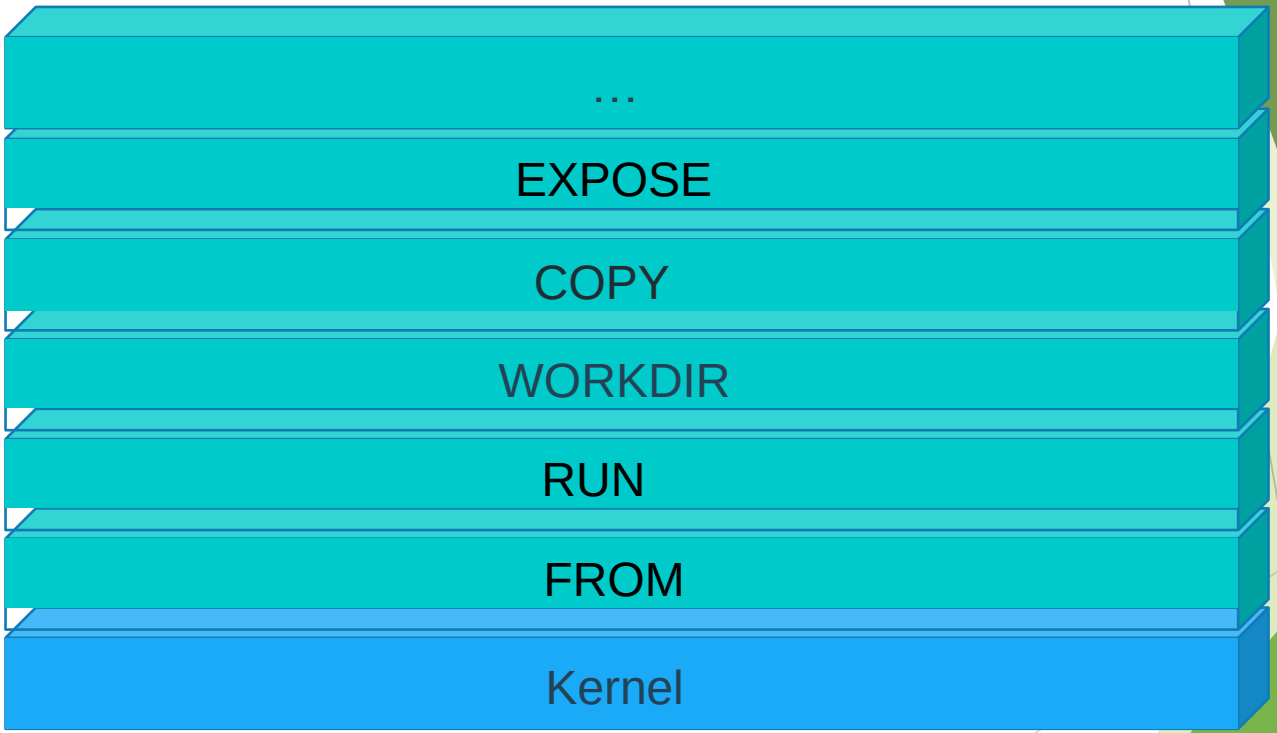
Montajes / Volúmenes

Vamos a volver al Dockerfile

Dockerfile

```
# Create image based on the official Node 6 image from dockerhub
FROM node:latest

# Create a directory where our app will be placed
RUN mkdir -p /usr/src/app

# Change directory so that our commands run inside this new directory
WORKDIR /usr/src/app

# Copy dependency definitions
COPY package.json /usr/src/app

# Install dependecies
RUN npm install

# Get all the code needed to run the app
COPY . /usr/src/app

# Expose the port the app runs in
EXPOSE 4200

# Serve the app
CMD ["npm", "start"]
```

Cada comando del Dockerfile crea una capa



Docker Image Pull: Descarga Capas

```
Alexander@DESKTOP-90ATKET MINGW64 ~/Docker/Demo
$ docker pull nginx:latest
latest: Pulling from library/nginx
bc95e04b23c0: Pull complete
f3186e650f4e: Pull complete
9ac7d6621708: Pull complete
Digest: sha256:b81f317384d7388708a498555c28a7cce778a8f291d90021208b3eba3fe74887
Status: Downloaded newer image for nginx:latest
```

Montajes en Docker

- Los montajes montan un directorio del host en una ubicación específica del contenedor
- Pueden ser usados para compartir (y persistir) datos entre contenedores (o con el host)
- El directorio permanece aunque el contenedor sea eliminado
    - A menos que explicitamente lo elimines
- Pueden ser creados en el Dockerfile o via CLI

Tipos de Montajes

- Los Volúmenes montan un directorio del host en una ubicación *gestionada por Docker (/var/lib/docker/volumes/ en Linux)*
- Los Enlaces de Montaje (bind mounts) montan un directorio del host en una ubicación *arbitraria* del host. Es posible montar archivos especiales (sockets, dispositivos específicos en /dev…)
- Pueden ser usados para compartir (y persistir) datos entre contenedores (o con el host)
- El directorio permanece aunque el contenedor sea eliminado
    - A menos que explicitamente lo elimines
- Pueden ser creados en el Dockerfile o via CLI

Para qué usamos los Montajes

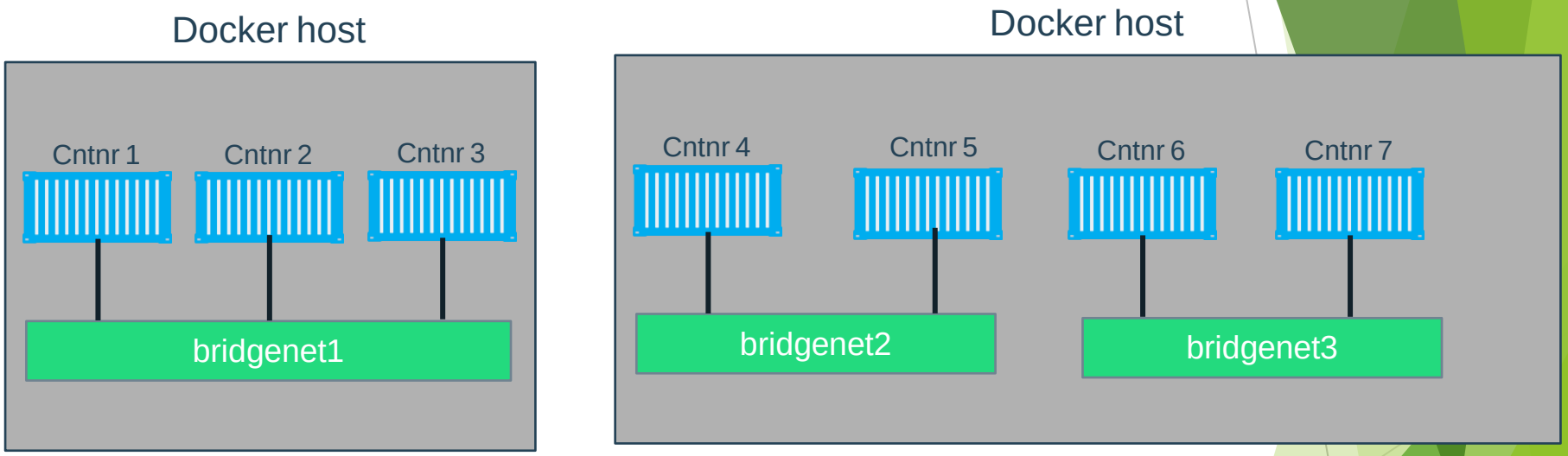
- Podemos montar código fuente local en un contenedor en ejecución

```
docker container run -v
$(pwd):/usr/src/app/  myapp
```

- Mejoras del rendimiento
- Persistencia de datos

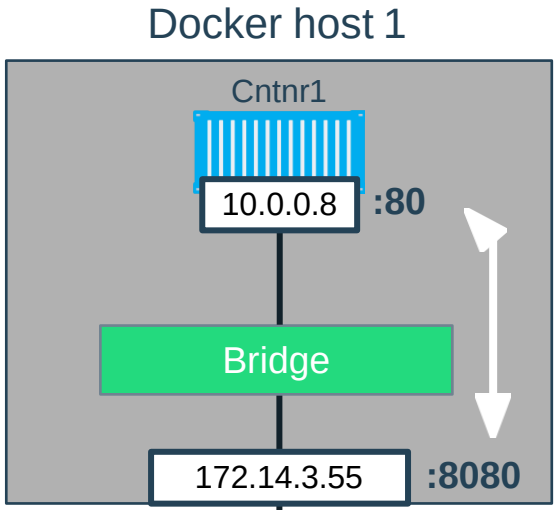
Sección 3: Networking

Qué es el Docker Bridge Networking




```
docker network create -d bridge --name bridgenet1
```

Docker Bridge Networking y Port Mapping

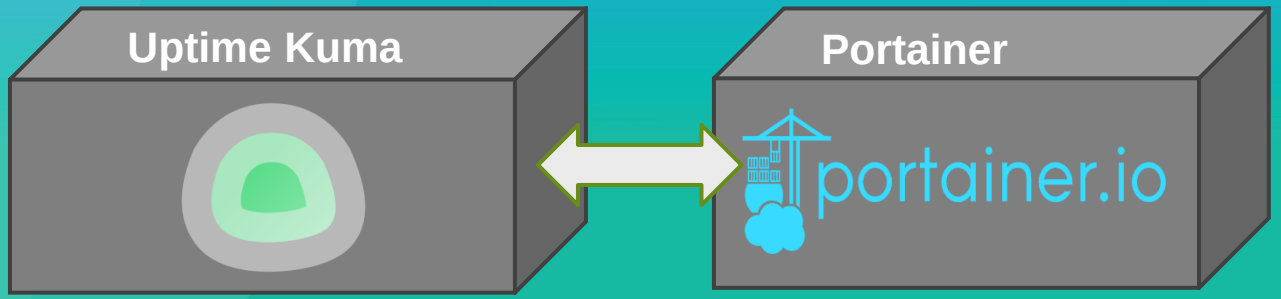


Puerto Host

Puerto Contenedor

```
$ docker container run -p 8080:80 ...
```

Demo



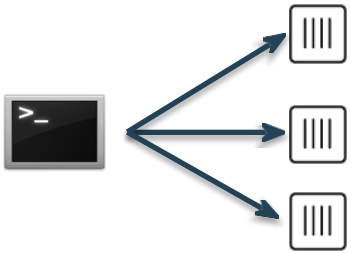
docker

Sección 4: Docker Compose

Docker Compose: Aplicaciones Multi Contenedor

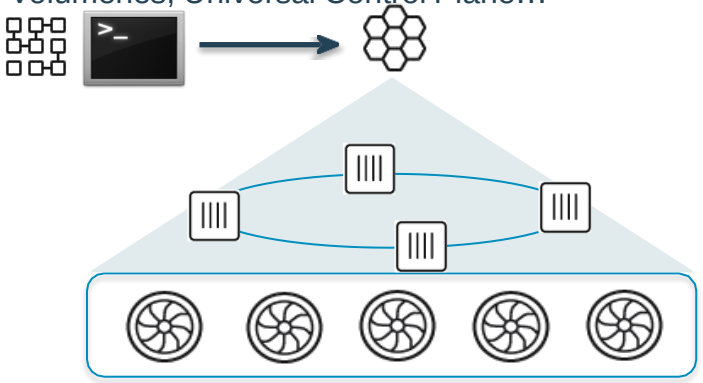
Docker-cli

- Construye y ejecuta un único contenedor
- Conecta contenedores manualmente
- Hay que tener cuidado con las dependencias y el orden de arranque

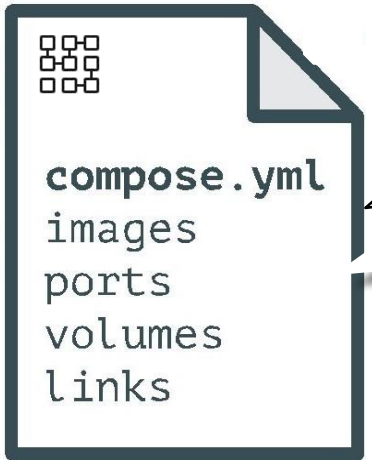


Docker Compose

- Puedes definir una aplicación multi-contenedor en un único archivo compose.yml
- Un único comando para desplegar la aplicación al completo
- Gestiona las dependencias entre contenedores
- Puedes utilizar Docker Swarm, Networking, Volúmenes, Universal Control Plane…



Docker Compose: Aplicaciones Multi Contenedor




```
version: '2' # declara la version de docker-compose

# Definimos los servicios y contenedores a ejecutar
services:
  angular: # nombre del primer servicio
    build: client # definimos el directorio del Dockerfile
    ports:
      - 4200:4200 # definimos el reenvío de puertos
  express: # nombre del segundo servicio
    image: api # definimos la imagen a utilizar
    ports:
      - 3977:3977 #definimos el reenvío de puertos
  database: # nombre del tercer servicio
    image: mongo # definimos la imagen a utilizar
    volumes:
      - ./dbvolume:/etc/dbapp # definimos un volumen
(bind)
    ports:
      - 27017:27017 # definimos el reenvío de puertos
```

Docker Compose:
Escalar Contenedores de Aplicación

Demo

docker

docker