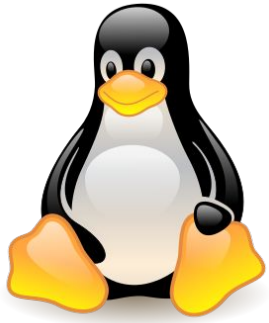
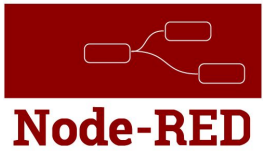


Interconectado APIs y Servicios con Node-RED



Mariano García Mattío
Twitter: @magm3333
email: magm@iua.edu.ar

Agenda

Introducción

Entorno de Servicios

- Linux
- Docker
  - Node-Red
  - MySQL
  - MongoDB
  - Mosquitto
    - MQTT
- Docker-compose (ejemplo)

GNU/Linux

O simplemente Linux!

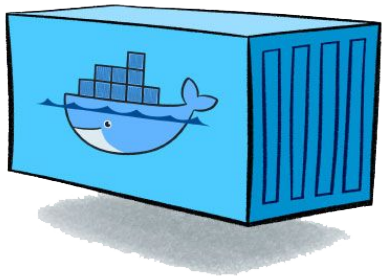
Las distros

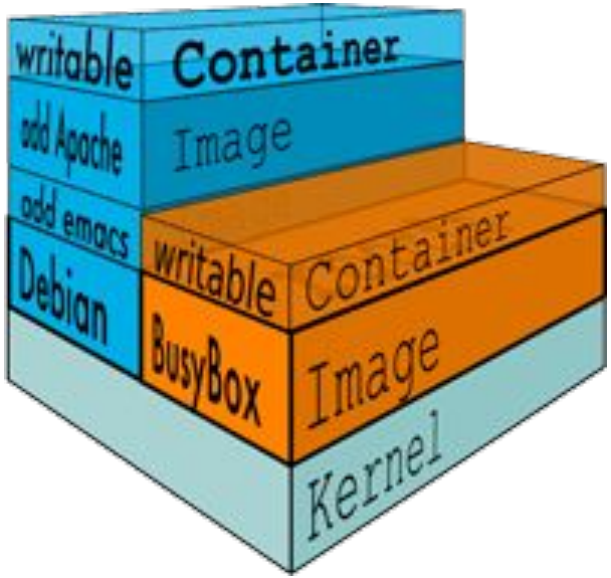
Docker

- Que es Docker?

"La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues." ← *Según docker*

- Que es un contenedor?



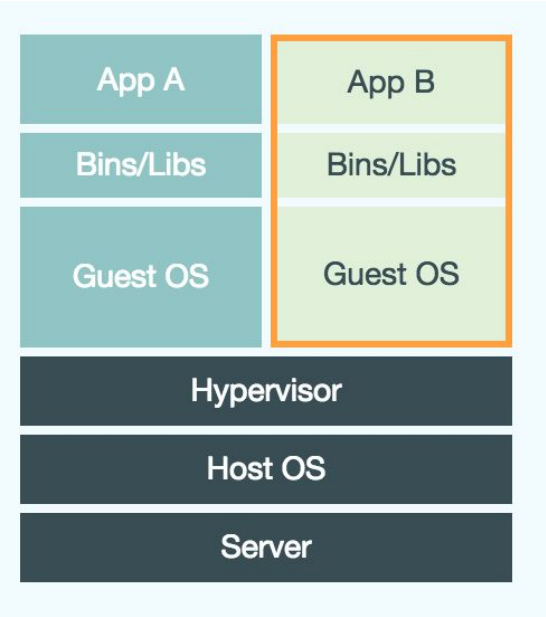


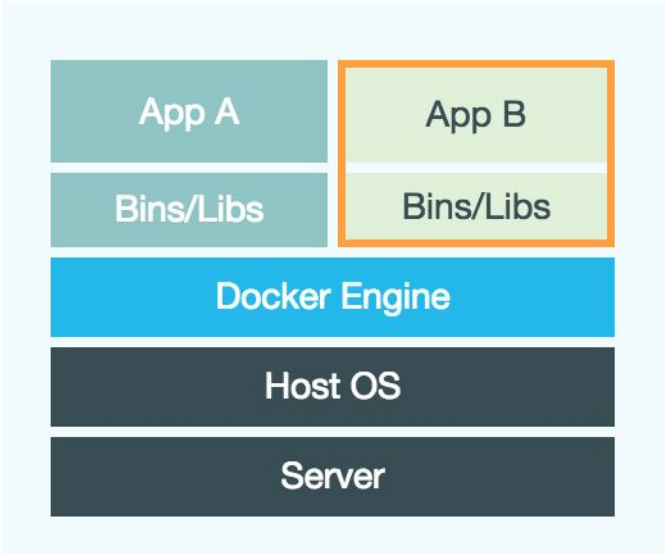
DOCKER

- Ventajas y beneficios
  - Desarrolladores
  - Testers
  - Sysadm
  - Menos pesado que una VM
  - Es Open Source!
  - Prototipado (servicios, microservicios, redes, interacciones complejas y heterogéneas)

DOCKER

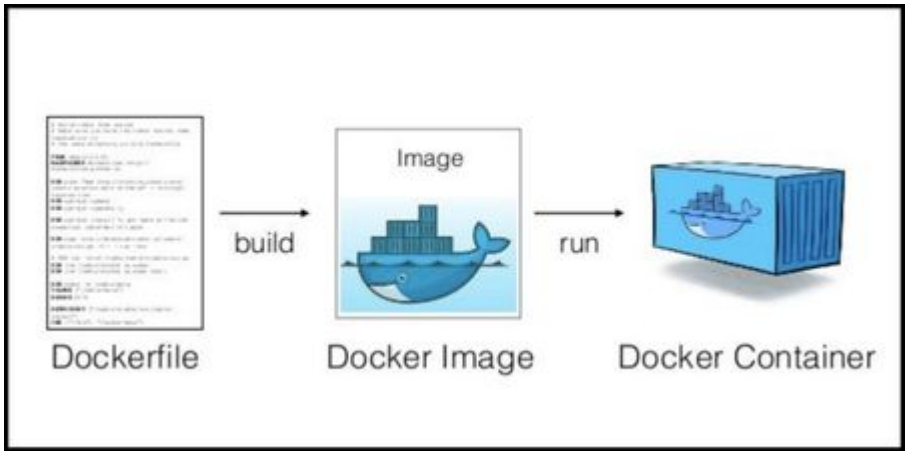
- Diferencia con una Máquina Virtual (VM)

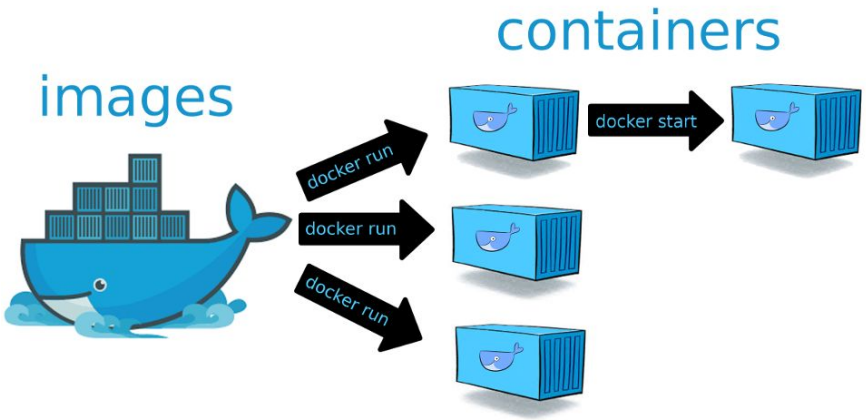




DOCKER

- Qué es una Imágen Docker?





DOCKER

- Qué es un Volumen?

Es el espacio físico que almacena los datos de cada contenedor.

Se crean en el host real y se asocian a cada contenedor creado.

Locación habitual: **/var/lib/docker/volumes**

Se pueden crear volúmenes con nombres familiares para facilitar la gest**ión**

DOCKER - PRÁCTICA

```
docker pull hello-world

docker run hello-world

docker run -it ubuntu bash

    cat /etc/lsb-release

docker image ls [--all]

docker rmi [hello-world | image_id | primeros 4 dígitos de image_id]

docker container ls [--all]  ||| docker ps [--all]

docker container rm [hello-world | container_id | primeros 4 dígitos de
container_id]
```

DOCKER - PRÁCTICA

```
docker inspect container_name

docker start container_name

docker exec -i -t [container_name | container_id] comand (ej: /bin/bash)

docker stop [container_name | container_id]
```

DOCKER - PRÁCTICA

Volúmenes:

```
docker volume create mis_datos

docker volume inspect mis_datos

docker volume rm mis_datos
```

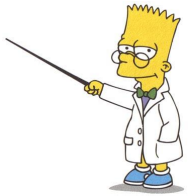
docker - Armado del Entorno

Node Red

Se puede usar un volumen (ejemplo anterior), con: **mis_datos**:/data

Mapeo de volúmenes (bind):
volumen host : volumen container

```
mkdir /home/ubuntu/node-red-data

sudo chown -R 1000:1000 /home/ubuntu/node-red-data/

docker run -it -p 1880:1880 -v /home/ubuntu/node-red-data:/data --name mynodered
nodered/node-red

docker start mynodered

docker stop mynodered
```



Mapeo de puertos:
puerto host : puerto container

Probar en el navegador:
http://localhost:1880

docker - Armado del Entorno

MySQL

Creamos el archivo **/home/ubuntu/dockerconfig/mysql57/mysqld.cnf**

```
[mysqld]
pid-file    = /var/run/mysqld/mysqld.pid
socket      = /var/run/mysqld/mysqld.sock
datadir           = /var/lib/mysql
log-error   = /var/log/mysql/error.log
bind-address= 0.0.0.0
symbolic-links=0
```

```
chmod 0444 ~/dockerconfig/mysql57/mysqld.cnf
```

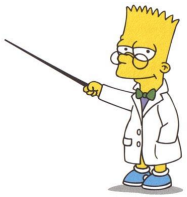
```
docker run --name mysql57 -v /home/ubuntu/dockerconfig/mysql57:/etc/mysql/mysql.conf.d -e
MYSQL_ROOT_PASSWORD=root -d mysql:5.7.27
```

Variable de entorno que define la password del usuario root, existen otras variables. Ver documentación: https://hub.docker.com/_/mysql

Docker - Armado del Entorno

MySQL

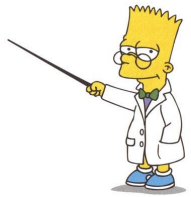
```
docker inspect mysql57 //Obtener IP

docker exec -i -t mysql57 /bin/bash

mysql -h 172.17.0.3  -uroot -proot
```



Docker - Armado del Entorno

MongoDB

```
docker run --name mongo -d mongo:4.2.0-bionic

docker exec -i -t mongo /bin/bash
     mongo
```



Docker - Armado del Entorno

Mosquitto

```
mkdir ~/dockerconfig/mosquitto
```

```
cd ~/dockerconfig/mosquitto
```

```
wget https://raw.githubusercontent.com/eclipse/mosquitto/master/mosquitto.conf
```

```
docker run --name mosquitto -it -p 1883:1883 -p 9111:9001 -v
/home/ubuntu/dockerconfig/mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf
eclipse-mosquitto:latest
```

```
docker start mosquitto
```

Abrir dos pestañas de terminal y probar:

Pestaña 1:

```
mosquitto_sub -t test
```

Pestaña 2:

```
mosquitto_pub -t test -m hola
```

docker - Armado del Entorno

Mosquitto

```
//Archivo de passwords de mosquitto
echo "ubuntu:ubuntu" > ~/dockerconfig/mosquitto/mosquitto.password


//Copiar en el filesystem del contenedor
docker cp ~/dockerconfig/mosquitto/mosquitto.password mosquitto:/mosquitto/config

//Abrir un shell en el contenedor
docker exec -i -t mosquitto /bin/sh

    cd /mosquitto/config
    mosquitto_passwd -U mosquitto.password
    cat mosquitto.password
```

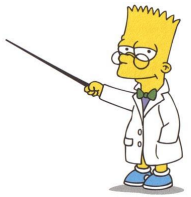
docker - Armado del Entorno

Mosquitto

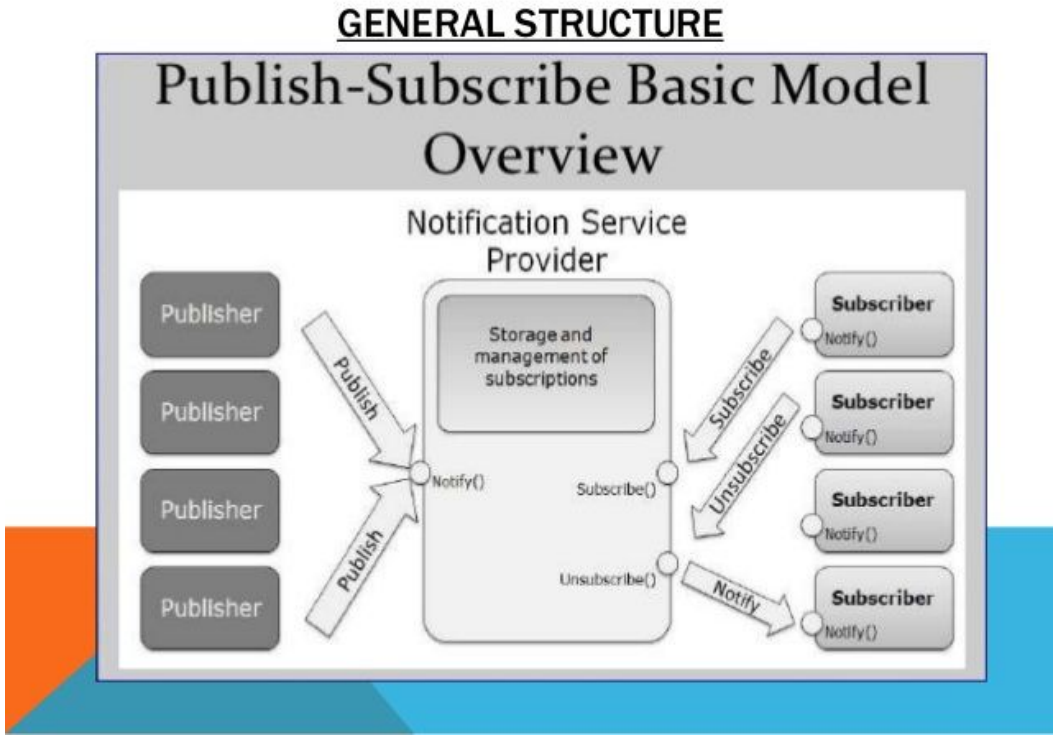
```
//Copiar el archivo de claves nuevamente al host
docker cp mosquitto:/mosquitto/config/mosquitto.password .

//Agregar al final del archivo mosquitto.conf
allow_anonymous false
password_file /mosquitto/config/mosquitto.password
```

Probar nuevamente en dos pestañas:

```
mosquitto_sub -t test -u ubuntu -P ubuntu
mosquitto_pub -t test -m hola -u ubuntu -P ubuntu
```



DOCKER - PUB/SUB - MQTT

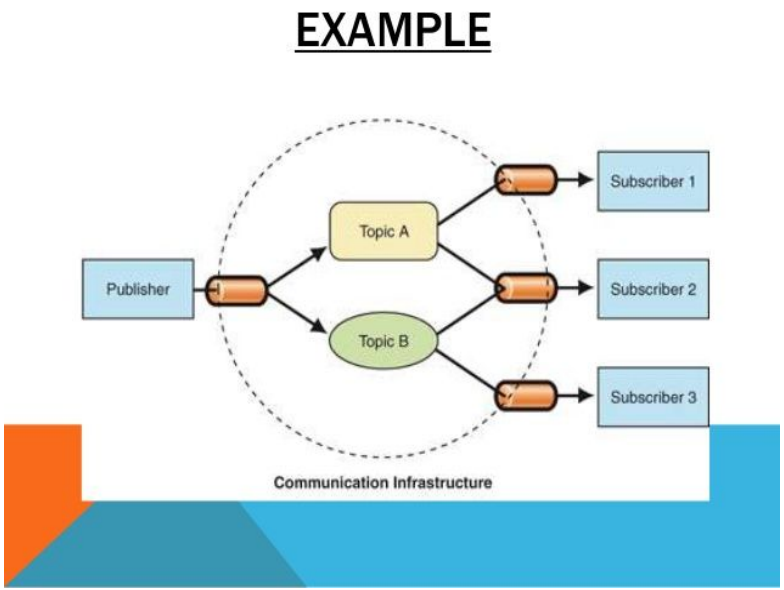


Patrón de comunicación por mensajería que permite a sistemas distribuidos comunicarse de manera desacoplada, anónima y asíncrona.

docker - Definiciones - MQTT

Middleware: o lógica de intercambio de información entre aplicaciones ("inter lógica"). Software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware y/o sistemas operativos. Conexiones y sincronizaciones que son necesarias en los sistemas distribuidos. Capa de abstracción de software distribuida entre las capas de aplicaciones y las capas inferiores (SO y red).
Message queue: componentes de software utilizados para lo comunicación entre procesos (IPC), o entre hilos dentro del mismo proceso (como mailboxes). Protocolos de comunicación asíncrona.
Message-oriented middleware: (MOM) infraestructura que permite envío y recepción de mensajes entre sistemas distribuidos, permitiendo a la aplicación ser distribuida en diferentes plataformas heterogéneas (SO y protocolos de red). El middlewere crea la capa de comunicación distribuida que aísla al desarrollador de la aplicación con los detalles del OS y la interfaz de red.

DOCKER - ELMENTOS PUB/SUB - MQTT



Publicadores: publican mensajes a la estructura de comunicación.
Suscriptores: se suscriben a una categoría de mensaje.
Infraestructura de comunicación: en la mayoría de los casos existe un BROKER de mensajes, encargado de mediar entre los publicadores y los suscriptores: filtrado, enrutado de mensajes y guardado o “encolado”.
Filtrado de mensajes: suscriptores reciben solo una serie de mensajes:

- Tópico: responsabilidad del publicador.
- Contenido: responsabilidad suscriptor.
- Híbridos

Enrutado: Unicast, Multicast y Server push - Client pull

docker - Ventajas y Desventajas - MQTT

Ventajas

- **Bajo Acoplamiento**:
  - Publicador y suscriptor no se conocen
  - Actúan de manera independiente
- **Escalabilidad**
- **Adaptabilidad en ambientes dinámicos**
- **Asíncrono**
- **Robusto**

Desventajas

- **Tiempos e intentos de entrega** del mensaje
- **Publicador asume que el receptor está “escuchando”**. Confiabilidad
- **Suscriptores pueden saturar la red**, bajando el rendimiento del ancho de banda.
- **Retardos** (ralentiza)
- **Seguridad**

DOCKER - MQTT

Protocolo de comunicación del tipo publicar/suscribir desarrollado por IBM.

Características:

- Funciona sobre la capa TCP/IP.
- Bajo costo (procesamiento)
- Mínimo ancho de banda.
- Aplicable en sistemas embebidos, ambientes IoT y edge computing.
- Implementación sencilla
- Altamente escalable.

Infraestructura

- Broker
- Cliente (publicador/suscriptor)
- Tópicos

DOCKER - MQTT

Tópicos.

La estructura jerárquica (forma de árbol) de tópicos se utiliza para el direccionamiento de los mensajes.

La notación es similar a un sistema de archivos (un path), se separan con barra "/"

No hace falta crearlo antes de enviar el mensaje, se genera en el mismo momento.

Especificaciones precisas de tópicos reducen el filtrado y ruteo de mensajes

DOCKER - MQTT

Wildcards

`casa/primerpiso/+/temperatura` ⇒ *suscribe a todas las temperaturas medidas en el primer piso*

Ejemplos de tópicos que serán recibidos:

`casa/primerpiso/living/temperatura`
`casa/primerpiso/comedor/temperatura`

Ejemplos de tópicos que NO serán recibidos:

`casa/segundopiso/habitación/temperatura`
`casa/primerpiso/living/humedad`

DOCKER - MQTT

Wildcards

`casa/primerpiso/living/` **#** ⇒ *suscribe a todos los tópicos que suceden a living*

Ejemplos de tópicos que serán recibidos:

casa/primerpiso/living/**temperatura**
casa/primerpiso/living/**humedad**
casa/primerpiso/living/**luces/puerta**

Ejemplos de tópicos que NO serán recibidos:

casa/segundopiso/habitación/temperatura
casa/primerpiso/comedor/humedad

DOCKER - MQTT

Quality of service (QoS)

La calidad de servicio (QoS) se refiere a un acuerdo entre el emisor y receptor de un mensaje con el foco en la garantía de la entrega de este.

En MQTT existen 3 niveles de QoS:

0. Como mucho una sola vez
1. Por lo menos una vez
2. Exactamente una sola vez

Docker - MQTT en Mosquitto

Tópicos especiales, comienzan con $

`$SYS/broker/clients/connected` ⇒ *Cantidad de clientes conectados*

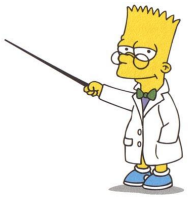
`$SYS/broker/clients/disconnected` ⇒ *Cantidad de clientes desconectados*

`$SYS/broker/clients/total` ⇒ *Cantidad de total de clientes*

`$SYS/broker/messages/sent` ⇒ *Cantidad de mensajes enviados*

`$SYS/broker/uptime` ⇒ *Tiempo de funcionamiento del servidor*

Más info: https://www.eclipse.org/mosquitto/man/mosquitto-8.php



docker - Compose

Herramienta que permite simplificar el uso de Docker, generando scripts que facilitan el diseño y la construcción de servicios.

- Se pueden crear varios contenedores al mismo tiempo
- En cada contenedor diferentes servicios
- Se pueden definir redes
- Se pueden asignar ips a los diferentes contenedores
- etc

docker - Compose

Nuestro **docker-compose.yml** (Parte 1/4)

```
version: '3.7'
services:
  mysql57:
    image: mysql:5.7.27
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: 'root'
    ports:
      - '3306:3306'
    expose:
      - '3306'
    volumes:
      - ~/dockerconfig/mysql57:/etc/mysql/mysql.conf.d
    networks:
      mired1:
        ipv4_address: 172.18.1.3
```

Solo se expone el puerto a los otros servicios, no al host

docker - Compose

Nuestro **docker-compose.yml** (Parte 2/4)

```
mynodered:
  image: nodered/node-red
  ports:
    - '1880:1880'
  expose:
    - '1880'
  volumes:
    - /home/ubuntu/node-red-data:/data
  networks:
    mired1:
      ipv4_address: 172.18.1.2
```

docker - Compose

Nuestro **docker-compose.yml** (Parte 3/4)

```
mongo:
  image: mongo:4.2.0-bionic
  ports:
    - '27017:27017'
  expose:
    - '27017'
  networks:
    mired1:
      ipv4_address: 172.18.1.5
```

docker - Compose

Nuestro **docker-compose.yml** (Parte 4/4)

```
  mosquitto:
     build: mosquitto/.
     ports:
       - '1883:1883'
       - '9111:9001'
     expose:
       - '1883'
       - '9001'
     networks:
       mired1:
         ipv4_address: 172.18.1.4
networks:
  mired1:
    driver: bridge
    ipam:
     config:
       - subnet: 172.18.1.0/24
```

docker - Compose

Archivo **mosquitto/Dockerfile**

```
FROM eclipse-mosquitto:latest

WORKDIR /mosquitto/config

COPY mosquitto.conf .
COPY mosquitto.password .
```

Archivo **mosquitto.password** debe estar cifrado

```
ubuntu:$6$Od/bIVEnCxodqMom$lVUjbNf4F7nOjX+y1m6L5cB81Xo9q04PVd6gGbRfErpFSDIcO7TyrhaiWz/vHR4OU+wWZtCDYUNUjiIiQ0
fHuA==
```

docker - Compose

Acciones:

docker-compose build

docker-compose up

docker-compose up -d → corre como demonio

docker-compose down → si se corrió como demonio

docker - Compose -Anexar Portainer

Nuestro **docker-compose.yml** (Parte extra)

```
  portainer:
    image: portainer/portainer
    restart: always
    environment:
      - VIRTUAL_HOST=dev.portainer
    networks:
      mired1:
        ipv4_address: 172.18.1.6
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - portainer_data:/data
volumes:
  portainer_data:

networks:
    ..............
```