




DOCKER



DOCKER INTEGRA TODO LO
NECESARIO PARA
EMPAQUETAR NUESTRO
SOFTWARE, CREANDO UNA
IMAGEN

UNA VEZ CREADA LA
IMAGEN, SE PUEDE
EJECUTAR UN
CONTENEDOR

EL CONTENEDOR SE PUEDE
EJECUTAR ALLÁ DONDE
HALLA UNA MÁQUINA CON
DOCKER

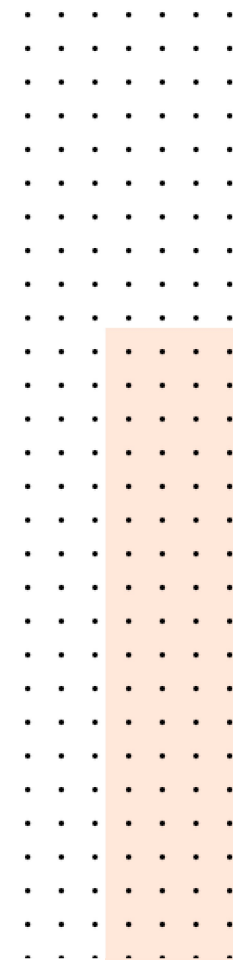


DOCKER VS MÁQUINAS VIRTUALES

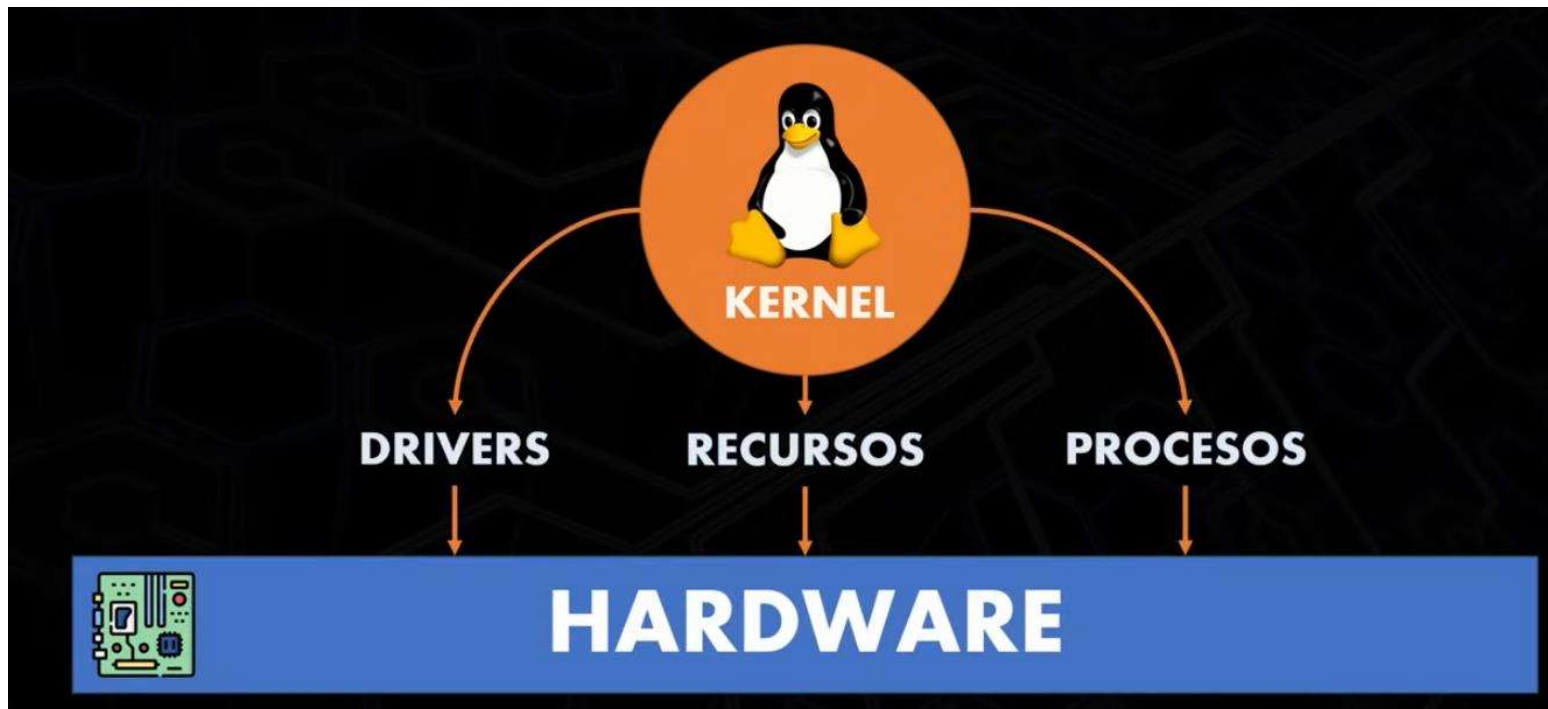
**EN LAS MÁQUINAS VIRTUALES, LA CONFIGURACIÓN
REQUIERE DE UN SISTEMA OPERATIVO COMPLETO**

**LA MÁQUINA VIRTUAL VIRTUALIZA EL HARDWARE,
MIENTRAS QUE LOS CONTENEDORES VIRTUALIZAN A
TRAVÉS DEL SISTEMA OPERATIVO Y SON MÁS
LIGEROS**

**EL KERNEL ES COMÚN PARA DISTINTOS
CONTENEDORES. AHÍ RESIDE LA CLAVE**



DOCKER VS MÁQUINAS VIRTUALES




DOCKER VS MÁQUINAS VIRTUALES



DOCKER VS MÁQUINAS VIRTUALES

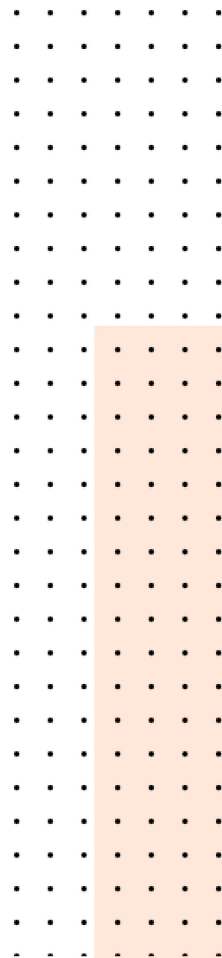




DOCKER VS MÁQUINAS VIRTUALES

LOS GRUPOS DE CONTROL Y LOS ESPACIOS DE NOMBRE SON LAS CARACTERÍSTICAS DEL KERNEL DE LINUX QUE NOS DAN EL AISLAMIENTO Y LA ASIGNACIÓN DE RECURSOS

DE MANERA QUE A TRAVÉS DE FUNCIONES INTERNAS DEL SISTEMA OPERATIVO, SE GESTIONAN RECURSOS HARDWARE EXCLUSIVOS PARA CADA CONTENEDOR, SIN NECESIDAD DE TENER EL SO COMPLETO



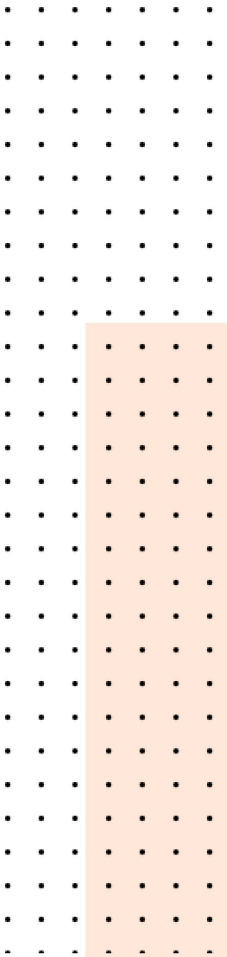


DOCKERFILE

**DOCKERFILE ES EL FICHERO QUE DESCRIBE CÓMO
COMPONER LA IMAGEN Y LA PLANTILLA DEL
PROYECTO**

**PARTIENDO DE UNA IMAGEN BASE (YA SEA UN SO O
UN ENTORNO COMO JDK) SE AGREGA EL PROYECTO
COMPILADO (JAR)**

**LA IMAGEN QUEDA LISTA PARA USARSE EN NUESTRO
ENTORNO**



DOCKERFILE

**EJEMPLO DE IMAGEN PARTIENDO DE LA ÚLTIMA
VERSIÓN DE JAVA, AGREGAMOS NUESTRA APP Y
ESPECIFICAMOS EL PUNTO DE ENTRADA**

```
FROM OPENJDK
```

```
ADD REST.JAR RESTCONT.JAR
```

```
ENTRYPOINT JAVA -JAR RESTCONT.JAR
```


DOCKER COMANDOS

DOCKER IMAGES => LISTADO DE IMÁGENES LOCAL

DOCKER IMAGES RM <IMAGEN>:<VERSIÓN> => LISTADO DE IMÁGENES LOCAL

DOCKER PS => CONTENEDORES EN EJECUCIÓN

DOCKER BUILD -T <IMAGEN> . => CONSTRUIMOS UNA IMAGEN

DOCKER RUN -P 8085:9456 <IMAGEN>


DOCKER PULL <NOMBRE_IMAGEN> => DESCARGA LA ÚLTIMA VERSIÓN

DOCKER TAG <IMAGEN> USUARIO/REPO:VERSION => CREAMOS UNA VERSIÓN

DOCKER PUSH USUARIO/REPO:VERSION => SUBIMOS VERSIÓN A DOCKERHUB



CICLO
COMPLETO
CI/CD



DE CÓMO GENERAR
VERSIONES DE UN
PROYECTO SIGUIENDO EL
FLUJO DE GIT FEATURES

PASOS CICLO COMPLETO

0 TENER ACTUALIZADO EL REPO LOCAL

GIT PULL

1 CREO Y SALTO A LA RAMA DE DESARROLLO

GIT CHECKOUT -B <RAMA>

2 MODIFICAMOS LA VERSIÓN EN POM

0.2.3-SNAPSHOT

3 TRABAJAMOS HASTA DAR POR BUENA LA MEJORA

GIT ADD .

GIT COMMIT -M <MENSAJE>



PASOS CICLO COMPLETO

4 ACTUALIZAMOS LA RAMA REMOTA DE DESARROLLO

GIT PUSH -U ORIGIN <RAMA>

5 CREAMOS LA PULL REQUEST EN GITHUB

ASIGNAMOS REVISORES OPCIONALES

6 LOS REVISORES RECIBIEN UN AVISO POR CORREO

DEBEN SER COLABORADORES PREVIAMENTE

7 EL REVISOR SE ACTULIZA SU REPOSITORIO

GIT PULL



PASOS CICLO COMPLETO

9 EL REVISOR SALTA A LA RAMA DE DESARROLLO

GIT CHECKOUT <RAMA>

10 AÑADE SUS CORRECCIONES (OPCIONAL)

GIT ADD .

GIT COMMIT -M <MENSAJE>

GIT PUSH -U ORIGIN <RAMA>

11 UNA VEZ QUE TODOS DAN EL VISTO BUENO

ACTUALIZAMOS LA VERSIÓN EN LOS FICHEROS MAVEN
Y HACEMOS EL ADD, COMMIT Y PUSH EN LA RAMA DE
DESARROLLO

PASOS CICLO COMPLETO

12 APROBAMOS EL PULL REQUEST VÍA GITHUB

SI LO HACEMOS A MANO SERÍA

GIT CHECKOUT MASTER

GIT MERGE <RAMA>

GIT PULL -U ORIGIN MASTER

CON ESTO SE INTEGRAN LOS CAMBIOS EN MASTER

13 GENERAMOS LA VERSIÓN

GIT TAG <VERSION>

14 CREAMOS LA VERSIÓN EN REMOTO

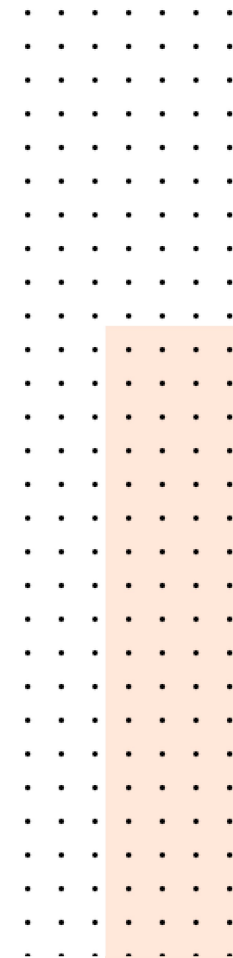
GIT PUSH --TAGS



APUNTES EXTRA

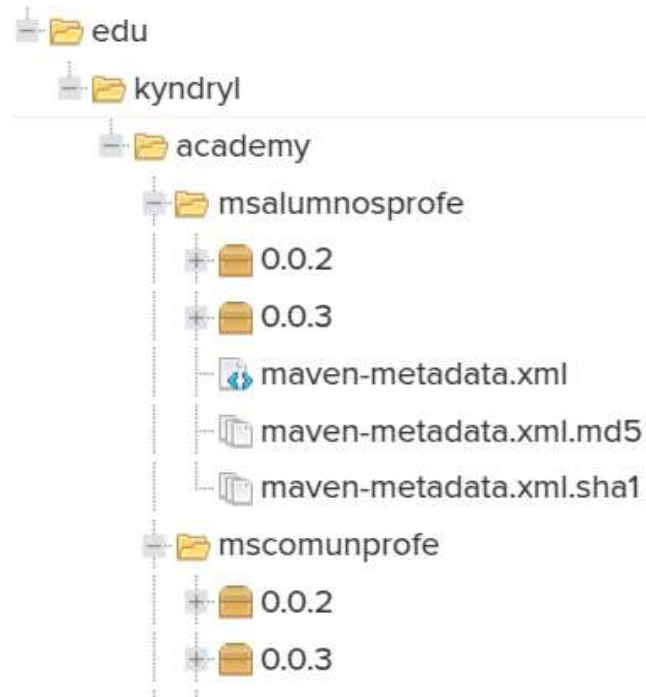
**CON LA CONFIGURACIÓN DE NUESTRO JENKINS,
SALTA LA EJECUCIÓN AL DETECTAR UNA NUEVA
VERSIÓN PÚBLICA**

**DEBIDAMENTE CONFIGURADO NEXUS, LOS NUEVOS
MICORSERVICIOS SE DESPLEGAN EN EL
REPOSITORIO CORPORATIVO**




APUNTES EXTRA

**ACCEDAMOS AL REPOSITORIO PÚBLICO EN NEXUS Y
DEBERÍAMOS VER GENERADA LA NUEVA VERSIÓN**





DOCKER COMPOSE



DOCKER COMPOSE ES UNA
HERRAMIENTA QUE NOS
PERMITE DESPLEGAR
VARIAS IMÁGENES DE
DOCKER SIMULTÁNEAMENTE

A TRAVÉS DE UN FICHERO
YML, SE INSTANCIAN Y
CONFIGURAN LAS
IMÁGENES A COORDINAR

DOCKER COMPOSE

DOCKER COMPOSE ES UN PLUGIN DE DOCKER QUE SE INSTALA APARTE MEDIANTE EL COMANDO

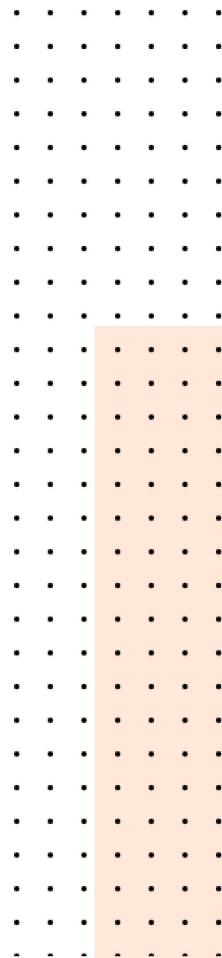
```
$ sudo apt-get update  
$ sudo apt-get install docker-compose-plugin
```



D O K C E R C O M P O S E

**CON COMPOSE VOY A PODER CREAR VARIAS
IMÁGENES Y CORRERLAS EN SU CONTENEDOR
SIMULTÁNEAMENTE**

PARA ELLO, DEBO TENER UN FICHERO YML, EN EL
DIRECTORIO RAÍZ, QUE REFERENCIA A CADA UNA DE
LAS IMÁGENES (UNA POR MICROSERVICIO) EN SU
RESPECTIVA SUBCARPETA



D O K C E R C O M P O S E

LA ESTRUCTURA QUEDARÍA ASÍ



D O K C E R C O M P O S E

CADA SUBCARPETA CONTIENE SU DOCKERFILE Y EL JAR FINAL CORRESPONDIENTE



```
from eclipse-temurin:17.0.14_7-jre-ubi9-minimal
add msalumnosprofe-0.0.3.jar alumnos.jar
entrypoint java -jar -Dspring.profiles.active=prod alumnos.jar
```

INCLUIMOS EL PERFIL DE PRODUCCIÓN COMO PARÁMETRO

D O K C E R C O M P O S E

EN EL FICHERO DE YML, REFERENCIAMOS A CADA IMAGEN ASÍ

```
docker-compose.yml x
services:
  eureka:
    build: ./eureka
    ports:
      - 8761:8761
  gateway:
    build: ./gateway
    ports:
      - 9090:9090
    depends_on:
      - eureka
    environment:
      EUREKA_HOST: 172.17.0.1
```

DOCKER COMPOSE

```
docker-compose.yml x
services:
  eureka:
    build: ./eureka
    ports:
      - 8761:8761
  gateway:
    build: ./gateway
    ports:
      - 9090:9090
    depends_on:
      - eureka
    environment:
      EUREKA_HOST: 172.17.0.1
```

SERVICE: ENUMERA CADA IMAGEN
CON EL NOMBRE QUE QUERAMOS

PORTS: HACEMOS CORRESPONDER
EL PUERTO PÚBLICO DE LA MÁQUINA
CON EL PUERTO INTERNO DE DOCKER

HOST_PORT:CONTAINER_PORT

D O K C E R C O M P O S E

```
docker-compose.yml x
services:
  eureka:
    build: ./eureka
    ports:
      - 8761:8761
  gateway:
    build: ./gateway
    ports:
      - 9090:9090
    depends_on:
      - eureka
    environment:
      EUREKA_HOST: 172.17.0.1
```

DEPENDS_ON: PODEMOS INDICAR
QUE ARRANQUE UN CONTENEDOR
ANTES QUE OTRO

ENVIROMENT: INCLUIMOS
PARÁMETROS QUE SERÁN USADA
EN LOS FICHEROS DE PROPIEDADES

DOCKER COMPOSE LOCALHOST

**NUESTRA BASE DE DATOS, ESTÁ FUERA DE DOCKER.
POR TANTO, DENTRO DE UN CONTENEDOR, LA IP DE
NUESTRA MÁQUINA HOST ES PARA DOCKER
HOST.DOCKER.INTERNAL.**

ESA SERÁ LA IP Y NO LOCALHOST, COMO VENÍAMOS
USANDO, LO QUE USEMOS EN LA CADENA DE
CONEXIÓN DEFINIDA EN LAS *PROPERTIES*

DOCKER COMPOSE LOCALHOST

SIN EMBARGO, SI ANTES USABA LOCALHOST PARA REFERIRME DESDE UN MICROSERVICIO A OTRO, AHORA AL ESTAR EN EL CONTEXTO VIRTUAL, YA NO VALE. DEBO CONSULTAR CON IFCONFIG LA IP QUE ASIGNA EL SISTEMA A DOCKER Y USAR ESA INTERFAZ

USAREMOS ESA IP PARA CONFIGURAR EUREKA O CUANDO COMUNIQUEMOS ENTRE MICROSERVICIOS DIRECTAMENTE

DOCKER COMPOSE

LA ESTRUCTURA QUEDARÍA ASÍ



AHORA, EJECUTO DOCKER COMPOSE UP DESDE ESTE DIRECTORIO Y TODOS LAS IMÁGENES SE RECREAN Y SE LANZAN SUS CONTEDORES



D O K C E R C O M P O S E

**PODEMOS CONSULTAR EL RESTO DE COMANDOS DE
COMPOSE EN LA REFERENCIA OFICIAL**

[DOCKER COMPOSE | DOCKER DOCS](#)

