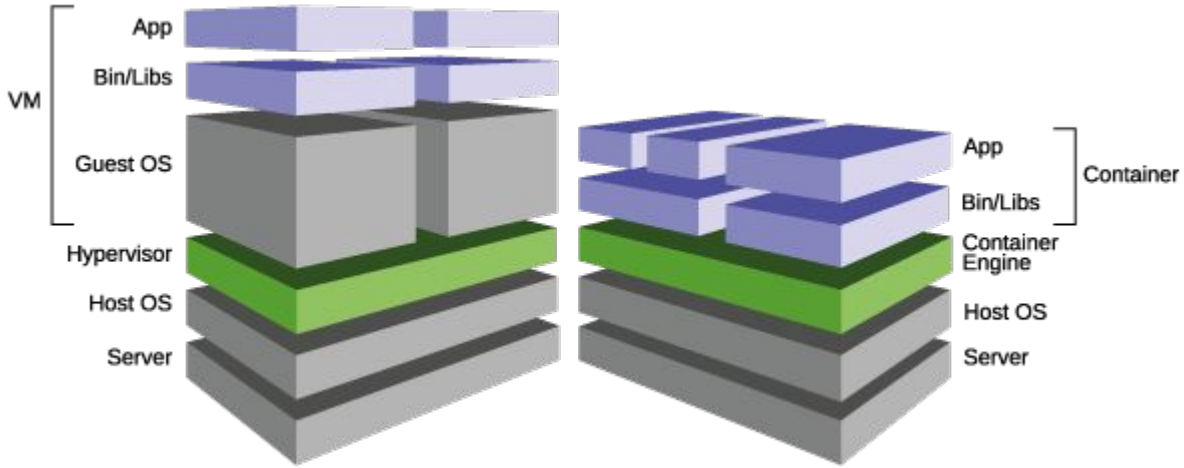
Docker & Kubernetes From Zero to (something like) Hero

¿Qué es Docker?

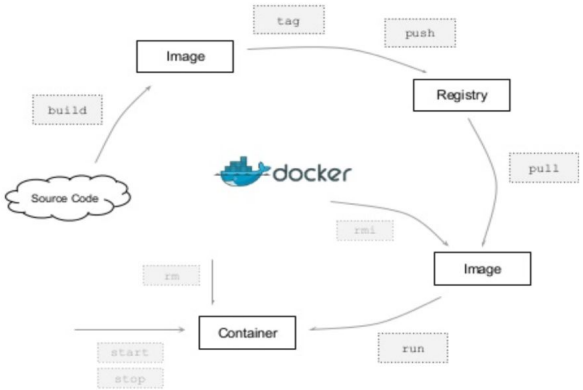
Docker propone virtualización a nivel de kernel:



¿Qué es Docker?

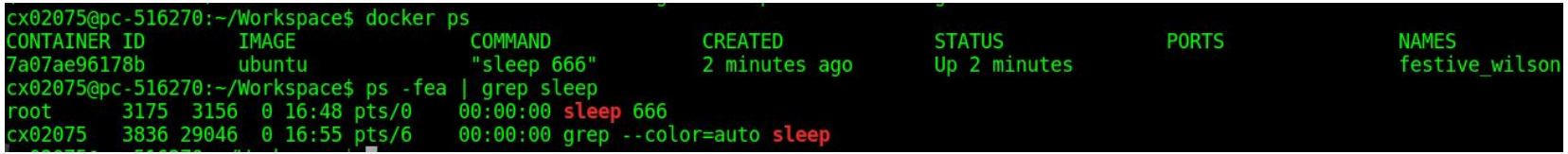
Docker es una tecnología que propone un paradigma basado en lanzar contenedores como procesos del host a partir de imágenes compartiendo las capas de SO, librerías y binarios si aplica.

Las **imágenes** son **elementos estáticos** (como una foto) y los **contenedores** nuestra **copia en runtime de esa imagen**, la cual podemos modificar a nuestro antojo, destruir o guardar como una nueva imagen.



Docker

Eso significa que todos los contenedores de docker corren sobre el mismo servidor y como procesos agrupados del host:



Pero con la ventaja de que dentro del contenedor sólo veremos los procesos del mismo:

¿Cómo lo hace Docker?

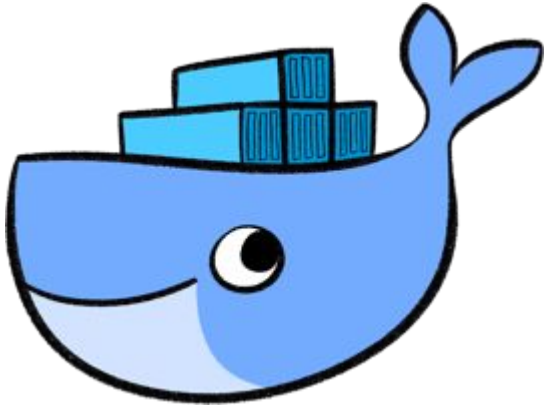
Docker puede aislar nuestros procesos apoyándose en las siguientes features del kernel:

- namespaces
  - pids
  - network interfaces
  - mount points
  - users (UIDs)
- cgroups (control groups)
  - hierarchical
  - memory, cpu, disk, disk i/o…
  - hard and soft limits
    - hard = OOMk por grupo
    - soft = WARNING! “mira macho…”
- chroot
  - Cambia la raíz del sistema para cada proceso

¿Y eso de los cgroups cómo se usa?

- Memory limits:
    - hard: --memory
    - soft: --memory-reservation
- CPU isolation
    - soft: --cpu-shares (uso ponderado)
    - hard: --cpuset-cpus (fija un proceso a N cpus)
    - hard*: --cpus=1.5 (como mucho usará 1.5)

Instalando docker



https://docs.docker.com

Docker en acción:

Docker tiene un repositorio público de imágenes en https://hub.docker.com/ donde encontraremos gran cantidad de imágenes creadas por la comunidad. La forma más fácil de iniciarnos será usar una imagen ya creada y arrancar un contenedor:

```
$ docker run -d --name hellouda-world python:3.6-alpine3.6
```

Docker en acción:

Una vez arrancado podemos ver el estado de los contenedores que se están ejecutando:

```
$ docker ps
```

O de todos los contenedores (aunque no estén running):

```
$ docker ps -a
```

Así como información concreta de nuestro contenedor:

```
$ docker inspect nombre-del-container
```

Volúmenes de Docker

Los contenedores pueden tener montados volúmenes del host, esto puede facilitar tareas como persistir directorios con datos, ficheros de configuración específicos, plantillas, etcétera.

Un volumen se usa añadiendo la bandera **-v** al comando **docker run** y después como argumento pondremos /path/volumen:/path/destino/en/container:modo

Ejemplos:

```
docker run -d --name=nginxtest -v nginx-vol:/usr/share/nginx/html nginx:latest
```

https://docs.docker.com/storage/volumes/

Binding de puertos en Docker

Los contenedores pueden tener puertos expuestos y la necesidad de que estos salgan por el host. Para hacer binding a un puerto añadimos la bandera **-p** al comando **docker run** y después como argumento pondremos puerto_host:puerto_contenedor.

Si ponemos -P mayúscula se asignará un puerto aleatorio libre del host a cada puerto definido por la imagen del servicio.

Ejemplos:

```
docker run -d --name=nginxtest -p 8000:80 -v nginx-vol:/usr/share/nginx/html nginx:latest
```

```
docker run -d --name=nginxtest -P -v nginx-vol:/usr/share/nginx/html nginx:latest
```

https://docs.docker.com/storage/volumes/

Docker en acción:

El comando docker run tiene varios modos:

Modo -d o detached: El contenedor muere si muere el PID 1.
Modo -t o pseudo-tty allocated: El contenedor se arranca adherido a un terminal.
Modo -i o interactive: El contenedor se vuelve interactivo y nos muestra la salida en primer plano aun sin estar “attached”

Kubernetes

Kubernetes es un orquestador de contenedores que propone abstracciones para los objetos clave de los despliegues de aplicaciones así como para la gestión de recursos. Todo se basa en el uso de sus objetos a través de su API.

kubernetes

Conceptos básicos de Kubernetes

Pod: Un pod es el objeto más simple del ecosistema de kubernetes. Representa únicamente un servicio dentro del cluster. Delimita los recursos que los contenedores usarán.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

Service: Es un conjunto de reglas para que los pods sean accedidos.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

Volume: Son volúmenes, mismo concepto que en docker, pero mejor gestionados para que no mueran hasta que su pod lo haga. Si muere un contenedor dentro de un pod, el nuevo que aparezca usará el mismo volumen que su predecesor.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-ebs
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-ebs
      name: test-volume
  volumes:
  - name: test-volume
    # This AWS EBS volume must already exist.
    awsElasticBlockStore:
      volumeID: <volume-id>
      fsType: ext4
```

Namespace: Un namespace sirve únicamente para poder generar múltiples entornos en un mismo cluster evitando la colisión de nombres entre objetos que deben ser únicos, pero sólo a nivel de namespace.

```
kubectl run nginx --image=nginx --namespace=<insert-namespace-name-here>
kubectl get pods --namespace=<insert-namespace-name-here>
```

High Level Abstractions de Kubernetes

ReplicaSet: Habitualmente se usa para asegurar que hay un mínimo de pods determinado corriendo.

Los replicaset nos permiten también aumentar el número de réplicas o disminuirlas para escalar nuestros servicios.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

```
kubectl apply -f https://kubernetes.io/examples/controllers/frontend.yaml
```

```
kubectl get rs
```

```
NAME       DESIRED   CURRENT   READY   AGE
frontend   3         3         3       6s
```

High Level Abstractions de Kubernetes

Deployment: Uno de los objetos más importantes de kubernetes y el más habitual. Se trata de una abstracción declarativa que impera sobre replicaset y pod.

```yaml
apiVersion: apps/v1 # Usa apps/v1beta2 para versiones anteriores a 1.9.0
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # indica al controlador que ejecute 2 pods
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

A jugar ^_^

1- Instalar minikube
2- Desplegar nuestro hello-kubernetes
3- Desplegar la Web-UI de k8s
4- Curiosear
5.- Desplegar mongodb sobre k8s (homework!)

```yaml
apiVersion: apps/v1 # Usa apps/v1beta2 para versiones anteriores a 1.9.0
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # indica al controlador que ejecute 2 pods
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```