

docker



Somos y formamos expertos en T.I.

¿Qué es Docker?

❖ <https://www.docker.com/>

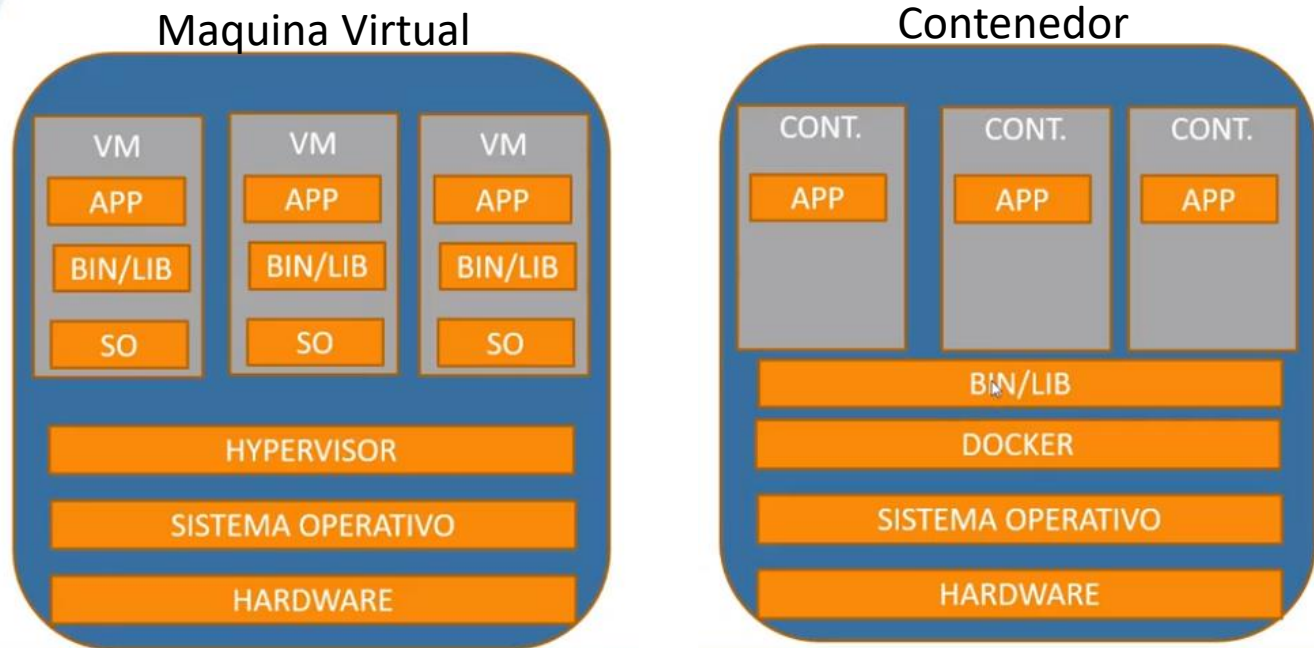
Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.

Ventajas de usar Docker

- ❖ Retorno de la inversión y ahorro de costos
- ❖ Estandarización y productividad
- ❖ Eficiencia de CI
- ❖ Compatibilidad y mantenibilidad
- ❖ Simplicidad y configuraciones más rápidas
- ❖ Despliegue rápido
- ❖ Plataformas multi-nube



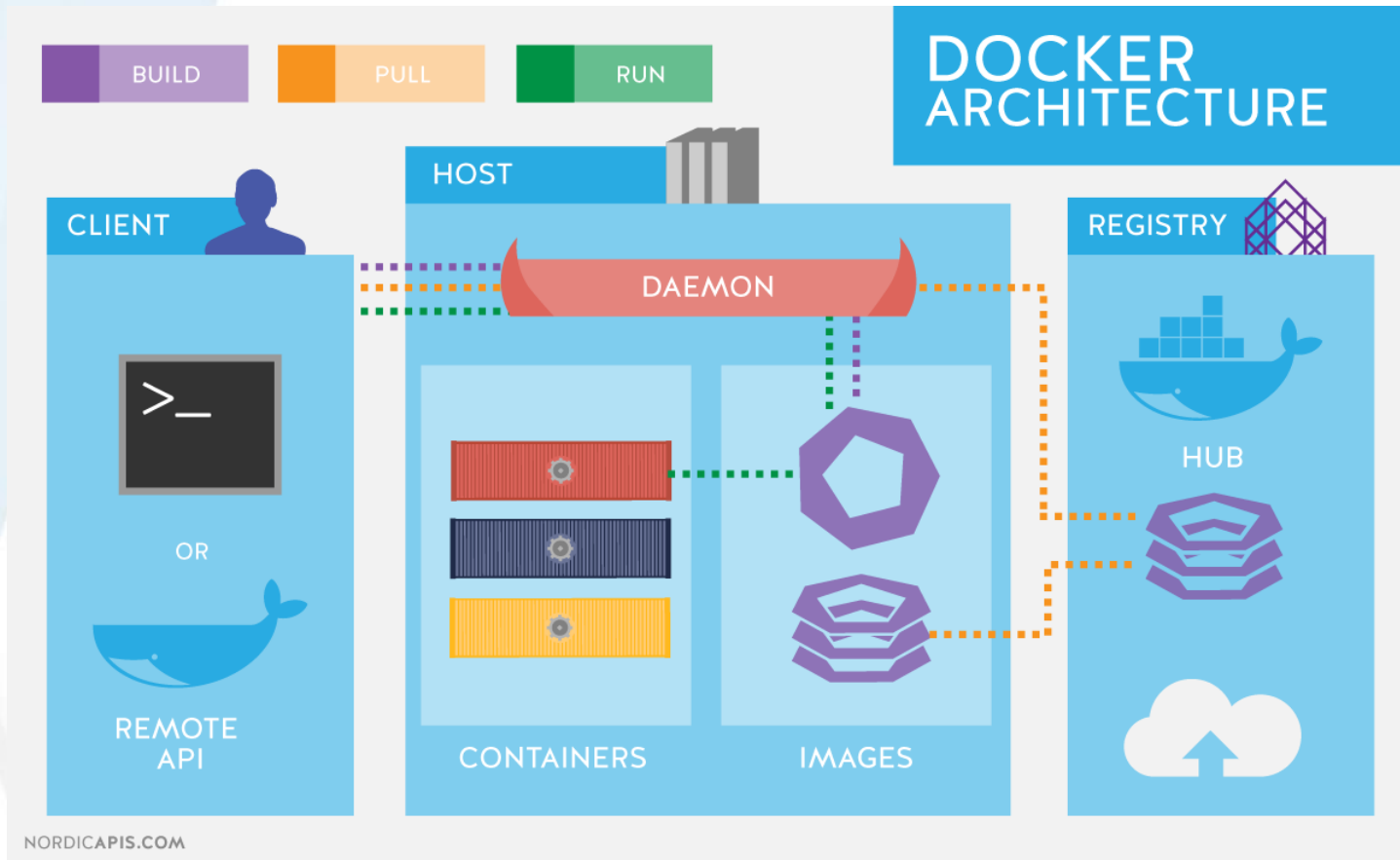
Diferencias con Maquinas Virtuales



Ideal para ...

- ☐ Docker se puede aplicar a distintas problemáticas que existen en la empresa:
 - ☐ Modernizar de forma sencilla aplicaciones tradicionales
 - ☐ CI (Continuous Integration) y CD (Continuous delivery) en DEVOPS
 - ☐ Fácil integración de entornos en la nube
 - ☐ Solución ideal para microservicios
 - ☐ Etc...

Arquitectura de Docker

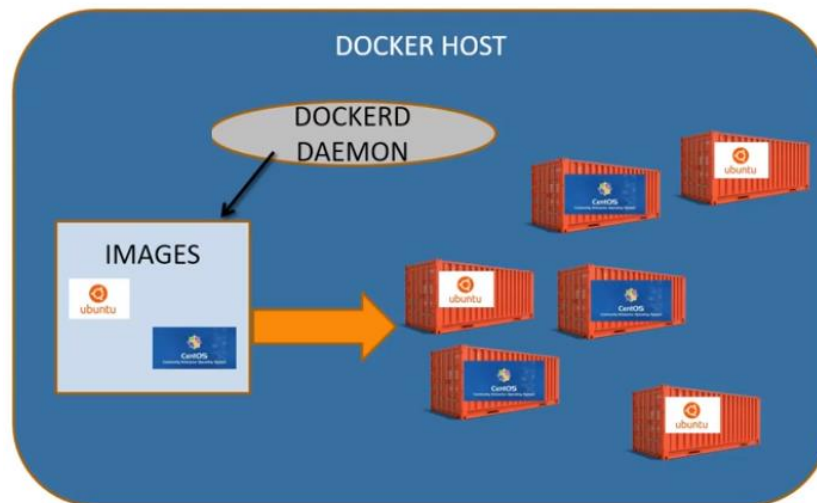


Componentes

Docker Client, regularmente usamos la consola aunque también se puede buscar alguna herramienta grafica (p.ej. Kitematic, Docstation, Portainer, Dockly).

Images, son plantillas que puedo descargar o crear a partir de otras, las imágenes son de solo lectura (para los Javeros, Image = Clase Java).

Containers, a partir de las imágenes se crean los contenedores (Container = Objeto Java).



Contenedores

Docker trabaja con algo que se llama “contenedores de Linux” estos son un conjunto de tecnologías que juntas forman un contenedor (de **Docker**), este conjunto de tecnologías se llaman:

Namespaces: Permite a la aplicación que corre en un contenedor de Docker tener una vista de los recursos del sistema operativo.

Cgroups: Permite limitar y medir los recursos que se encuentran disponibles en el sistema operativo.

Chroot: Permite tener en el contenedor una vista de un sistema “falso” para el mismo, es decir, crea su propio entorno de ejecución con su propio root y home.

Algunas de las características más notables de un contenedor son:

- Los contenedores son más livianos (ya que trabajan directamente sobre el Kernel) que las maquinas virtuales.
- No es necesario instalar un sistema operativo por contenedor.
- Menor uso de los recursos de la máquina.
- Mayor cantidad de contenedores por equipo físico.
- Mejor portabilidad.

Primeros Pasos con Docker

docker version

Obtener la versión actual del cliente y engine de Docker.

docker run hello-world

Crea un contenedor a partir de la imagen hello-world.

docker images

Obtener la lista de imágenes que se encuentran en nuestra maquina, IMAGE_ID es un número hash.

docker ps

Obtener la lista de los contenedores en ejecución.

docker ps -a

Obtener la lista de todos los contenedores (detenidos y en ejecución)
NAMES es un nombre aleatorio.

docker ps --help

Options:

-a, --all	Show all containers (default shows just running)
-f, --filter filter	Filter output based on conditions provided
--format string	Pretty-print containers using a Go template
-n, --last int	Show n last created containers (includes all states) (default -1)
-l, --latest	Show the latest created container (includes all states)
--no-trunc	Don't truncate output
-q, --quiet	Only display numeric IDs
-s, --size	Display total file sizes

**Practice
Makes
Perfect**

- 1) Mostrar el último contenedor ejecutado
- 2) Mostrar los últimos 4 contenedores ejecutados
- 3) Mostrar los ID de todos los contenedores
- 4) Mostrar el tamaño de todos los contenedores
- 5) Buscar por nombre un contenedor
- 6) Revisar el comando docker images --help

<https://docs.docker.com/engine/reference/commandline/ps/#filtering>

Contenedor Interactivo

- Consola 1: **docker run -it ubuntu**
- Consola 2: **docker ps**
- Consola 1: **ejecutar comandos Linux: ls, pwd, etc**

Dejar el contenedor activo en segundo plano:

CTRL + P + Q (Windows 10)

CTRL + C (Windows 7)

Regresar al contenedor

docker attach #idcontenedor

- Consola 2: **docker images** ¿Diferencia con docker ps?
- Consola 1: **exit**
- Consola 2: **docker ps** ¿Dónde quedo el contenedor?
- Consola 1: **Iniciar nuevamente el contenedor.**
- Consola 1: **docker stop #idcontenedor**
- Consola 1: **docker start #id VS docker start -i #id**
- Consola 1: **docker run -it ubuntu** ¿Qué paso?

Iniciar un contenedor en segundo plano

`docker run -d -it ubuntu`

Eliminar contenedores e imagenes

```
docker rm #idcontenedor  
docker rmi #imagen
```

```
docker rmi -f #imagen
```

Docker Hub

Probablemente uno de los éxitos de Docker más que la propia tecnología de contenedores sea Docker Hub que permite a los usuarios compartir las imágenes construidas, se podría decir que es el GitHub de los contenedores docker y quizá por ello el paralelismo en el nombre entre ambos. Docker Hub permite subir imágenes o usar las imágenes oficiales de postgresql, mysql, ubuntu, tomcat, ... y otra multitud de proyectos.

El archivo Dockerfile con el que construimos una imagen podemos hospedarlo en un repositorio de GitHub y que Docker Hub lo obtenga para construir la imagen. Docker Hub ofrece repositorios públicos en los que colocar las imágenes que cualquier otro usuario puede acceder y usar o repositorios privados con cierto coste según el número de repositorios privados, el primer repositorio privado es gratuito.



Conocer los tags y pulls Docker Hub

Crear una cuenta para
posteriormente subir
imágenes.

Ejecutar comandos en un contenedor

```
docker run -it --name miubuntu ubuntu bash  
docker exec miubuntu echo hola
```

```
docker pull python
```

```
docker images
```

```
docker run -it --name mipython python (Entra a la  
consola de desarrollo de Python)
```

Desde otra consola podemos entrar al bash

```
docker exec -it mipython bash
```

docker image y docker container

docker image (sin s)

docker image ls (docker images)

docker container

docker container ls -a (docker ps -a)

Logs y Kill

docker run -d ubuntu sh -c “while true; do date; done”
-d (background) sh (ejecutar un shell)

Consultar que esta haciendo el contenedor:

docker logs #idcontenedor

docker logs #idcontenedor --tail 10 (10 ultimas líneas)

docker kill #idcontenedor (Similar al docker stop)

Top y Stats

docker run --it ubuntu bash

Desde otra consola:

docker ps

docker top #idcontenedor

(proceso que más recursos consume)

docker stats #idcontenedor

(estadísticas generales de recursos que ocupa el contenedor)

Inspect

`docker inspect #idcontenedor`

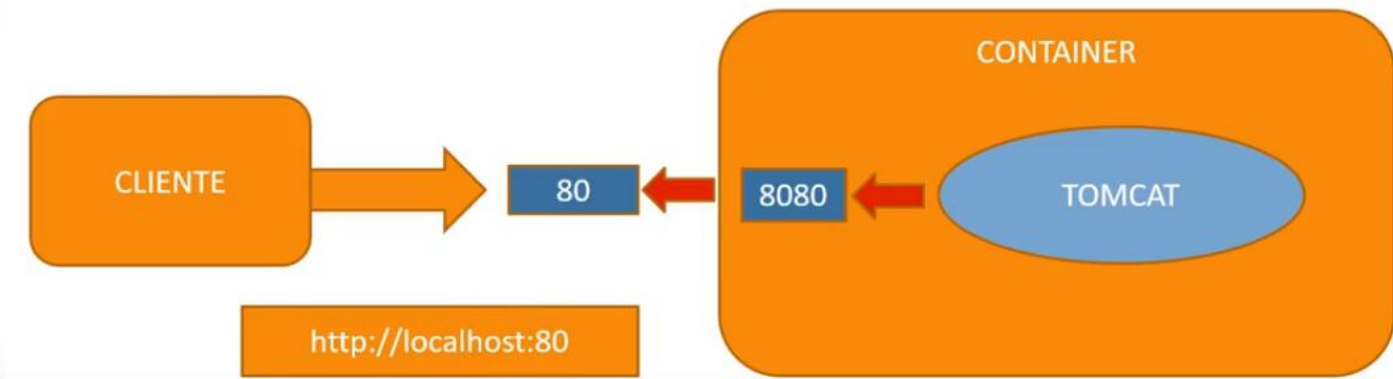
Enviar la salida a un archivo:

`docker inspect #idcontenedor >> contenedor.txt`

(muestra la información general del contenedor, aplica igual para imagenes)

Puertos en Docker

- ❑ Un contenedor puede tener aplicaciones que necesiten ser accedidas desde fuera del contenedor, por ejemplo Apache o Tomcat
- ❑ Por defecto los puertos de un contenedor son privados y no pueden ser accedidos
- ❑ Debemos hacerlos públicos y mapearlos con un puerto del host



Ejemplo: nginx

docker run -d -P nginx

Automáticamente mapea los puertos necesarios

```
PORTS  
0.0.0.0:32768->80/tcp
```

<http://localhost:32768>

docker run -d -p 8080:80 nginx

<http://localhost:8080>

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Redes en Docker

`docker network ls`

bridge (predefinida), redes privadas dentro de una maquina que se conectan a la red física.

host, todos los contenedores en esa red pueden ver a la maquina principal, pero no se pueden ver entre si (normalmente no se usa)
none, contenedores que no requieren de la red.

`docker inspect #idcontenedor`

```
"IPAdress": "172.17.0.3",  
  "IPAdress": "172.17.0.3",
```

docker network inspect

docker network inspect bridge >> networkbridge.txt

```
"Containers": {  
  "ce999c4971577696747507ccf7f2491ad41"  
    "Name": "gallant_poitras",  
    "EndpointID": "9a67934f2b9a578e9",  
    "MacAddress": "02:42:ac:11:00:02",  
    "IPv4Address": "172.17.0.2/16",  
    "IPv6Address": ""  
  },  
  "df55f4730180859046cef9b287b12e4df1d"  
    "Name": "nginx2",  
    "EndpointID": "56ed2af8295851fce",  
    "MacAddress": "02:42:ac:11:00:03",  
    "IPv4Address": "172.17.0.3/16",  
    "IPv6Address": ""  
}
```



TecGurus

Somos y formamos expertos en T.I.

Crear Redes en Docker

docker network create red1

Crea una red de tipo bridge llamada red1.

docker inspect red1 >> red1.txt

```
"Subnet": "172.18.0.0/16",  
"Gateway": "172.18.0.1"
```

docker network create --subnet=172.20.0.0/16 red2

docker inspect red2 >> red2.txt

Asociar Contenedores a una Red

```
docker network ls  
docker run -it --name ubuntu1 --network red1 ubuntu
```

```
docker run -d --name nginxred1 --network red1 nginx  
docker inspect red1 >> red1.txt
```

```
docker attach ubuntu1  
ping 178.18.0.X (ip del nginx) apt-get update  
apt-get install iputils-ping
```

```
root@0286d4e4b1d2:/# ping 172.18.0.3  
PING 172.18.0.3 (172.18.0.3) 56(84) bytes of data.  
64 bytes from 172.18.0.3: icmp_seq=1 ttl=64 time=0.116 ms  
64 bytes from 172.18.0.3: icmp_seq=2 ttl=64 time=0.150 ms
```

```
docker network connect red2 ubuntu1
```

Conecta en runtime a otra red el contenedor (a las 2 redes).

```
docker inspect ubuntu1 Checar IPAddress
```


Volumenes

- ❑ Los volúmenes son el mecanismo preferido para persistir la información y los datos en contenedores Docker.
- ❑ Docker administra los volúmenes completamente.
- ❑ Los volúmenes tienen varias ventajas sobre los puntos de montaje tradicionales
 - ❑ Los volúmenes son más fáciles de respaldar o migrar.
 - ❑ Se puede administrar volúmenes utilizando los comandos CLI de Docker o la API de Docker.
 - ❑ Los volúmenes funcionan tanto en contenedores de Linux como de Windows.
- ❑ Los volúmenes se pueden compartir de forma más segura entre varios contenedores.
- ❑ Se pueden almacenar volúmenes en hosts remotos o en entornos cloud, cifrar el contenido, etc.
- ❑ Los contenidos de un nuevo volumen pueden ser rellenos de forma previa por un contenedor.

Volumenes



- ☐ Con los volúmenes puedo
 - ☐ Usar un almacenamiento persistente para el contenedor
 - ☐ Compartir almacenamiento entre el HOST y un contenedor
 - ☐ Compartir almacenamiento entre distintos contenedores

Crear volúmenes en un contenedor

```
docker run -it -v c:/temp/datos:/datos --name ubuntu1 ubuntu bash
```

Crea un volumen

```
ls -l /datos
```

```
cd /datos
```

```
touch f1.txt
```

 (Crear un archivo dentro del contenedor)

Revisar la carpeta c:/temp/datos

```
docker run -it --name ubuntu2 --volumes-from ubuntu1 ubuntu bash
```

Comparte los volúmenes creados para el contenedor ubuntu1

Imágenes

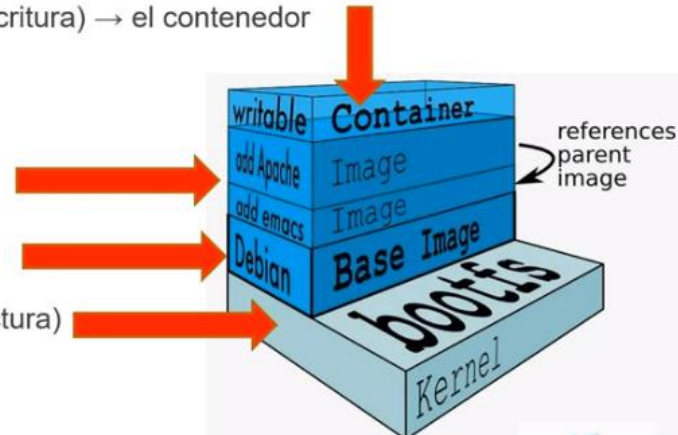
❑ Las imágenes en Docker están formadas por varias capas solo de lectura.

❑ Monta una capa por encima (lectura/escritura) → el contenedor

❑ Monta las n capas de imagen (lectura)

❑ Monta el sistema de ficheros de root: rootfs (lectura)

❑ Monta el sistema de ficheros de arranque: bootfs (lectura)



TecGurus

Somos y formamos expertos en T.I.

Personalizar un contenedor

docker run -it --name ubuntuwget ubuntu bash

Si trato de ejecutar wget <http://www.google.com> no esta disponible el comando

apt-get update
apt-get install wget

Ya debería funcionar wget <http://www.google.com>

Desde otra consola ejecutar:

docker diff ubuntuwget

Indica los cambios que se han realizado en el contenedor.

Crear una imagen a partir del contenedor

```
docker commit ubuntuwget mi_imagen_ubuntu_wget
```

Crea una imagen con ese nombre.

```
docker images
```

```
docker run --it mi_imagen_ubuntu_wget
```

Este contenedor se inicia y ya contiene wget.

Dockerfile

Revisar el dockerfile de hello-world

Revisar dockerfile de ubuntu (Mucho más complejo)

Un Dockerfile se compone de una serie de directivas, que construyen las capas que componen la imagen.

Crear un Dockerfile

- 1) Crear un directorio imagenpython
- 2) Crear un archivo con nombre Dockerfile
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
- 3) docker build -t imagen_python .
(Ejecutar en la carpeta donde esta el Dockerfile)
- 4) docker images
- 5) docker run -it imagen_python python
- 6) docker image history imagen_python

Directiva RUN

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping
```

```
docker build -t imagen_python:v1 .
```

```
docker images
```

imagen_python	v1
imagen_python	latest

docker run -it imagen_python:v1 bash

Directiva CMD

Es el comando que se ejecuta al terminar de crear el contenedor.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping
CMD echo "Welcome to this container" Basado en una shell
```

```
docker build -t imagen_python:v1 . v2
```

```
CMD ["echo","Welcome to this container"] exec (No necesito una shell)
CMD ["/bin/bash"]
```

```
docker run -it --rm image:v2 ls
```

Con CMD al momento de crear el contenedor se puede reemplazar el comando, en este ejemplo por ls

Directiva ENTRYPOINT

Es el comando que se ejecuta al terminar de crear el contenedor.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping
ENTRYPOINT ["/bin/bash"] exec (No necesito una shell)
```

```
docker build -t image:v2 .| No olvidar el punto
```

```
docker run -it --rm image:v2 df -h
```

df -h marcaria error, por que **ENTRYPOINT** ejecuta siempre el comando que le pongamos, no puedo agregar comandos finales, ya que los concatena.

Directiva WORKDIR

Cambia el directorio de trabajo y sobre ese se ejecutan los RUN.

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping
RUN mkdir /datos
WORKDIR /datos
RUN touch f1.txt
RUN mkdir /datos1
WORKDIR /datos1
RUN touch f2.txt
ENTRYPOINT ["/bin/bash"]
```


Directiva COPY

Copia archivos del HOST hacia el contenedor

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python
RUN echo 1.0 >> /etc/version && apt-get install -y git \
    && apt-get install -y iputils-ping

##WORKDIR##
RUN mkdir /datos
WORKDIR /datos
RUN touch f1.txt
RUN mkdir /datos1
WORKDIR /datos1
RUN touch f2.txt

##COPY##
COPY index.html .
COPY app.log /datos

##ENTRYPOINT##
ENTRYPOINT ["/bin/bash"]
```

Los archivos a copiar deben estar en la misma carpeta del Dockerfile

```
app.log
Dockerfile
index.html
```

```
docker build -t image:v4 .
```


Directiva ADD

Copia archivos del HOST hacia el contenedor, en caso de .tar los descomprime.

```
##WORKDIR##  
RUN mkdir /datos  
WORKDIR /datos  
RUN touch f1.txt  
RUN mkdir /datos1  
WORKDIR /datos1  
RUN touch f2.txt
```

```
##COPY##  
COPY index.html .  
COPY app.log /datos
```

```
##ADD##  
ADD docs docs  
ADD f* /datos/  
ADD f.tar .
```

```
##ENTRYPOINT##  
ENTRYPOINT ["/bin/bash"]
```

Copia la carpeta docs a docs (/datos1)
f* copia todos los archivos que su nombre empiece con f.
Descomprime f.tar en . (WORKDIR /datos1)

```
docker build -t image:v4 .
```

Docker Hub

¿Cómo subir las imágenes creadas?

docker login

Revisar el nombre de la imagen a subir (si es necesario cambiar el nombre)

docker image tag miweb tecgurus/miweb

docker image tag miweb tecgurus/miweb:v1

Subir la imagen al servidor

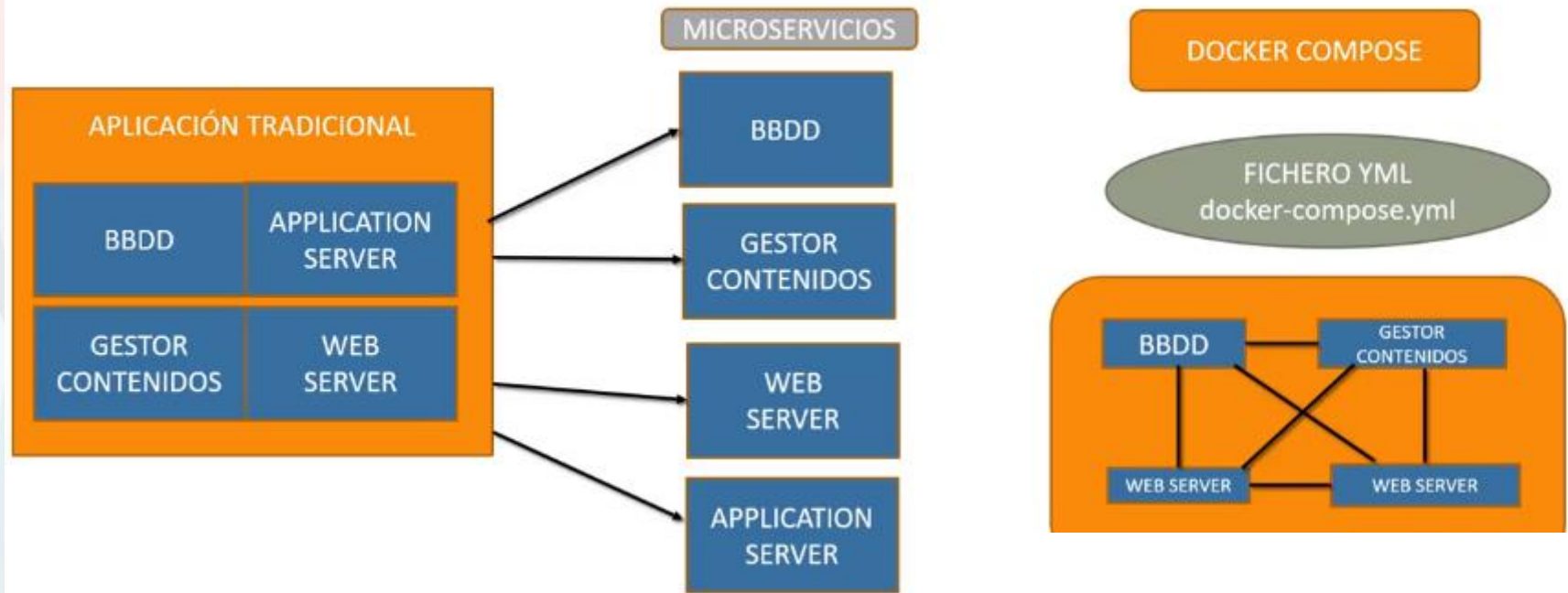
docker push tecgurus/miweb

docker push tecgurus/miweb:v1

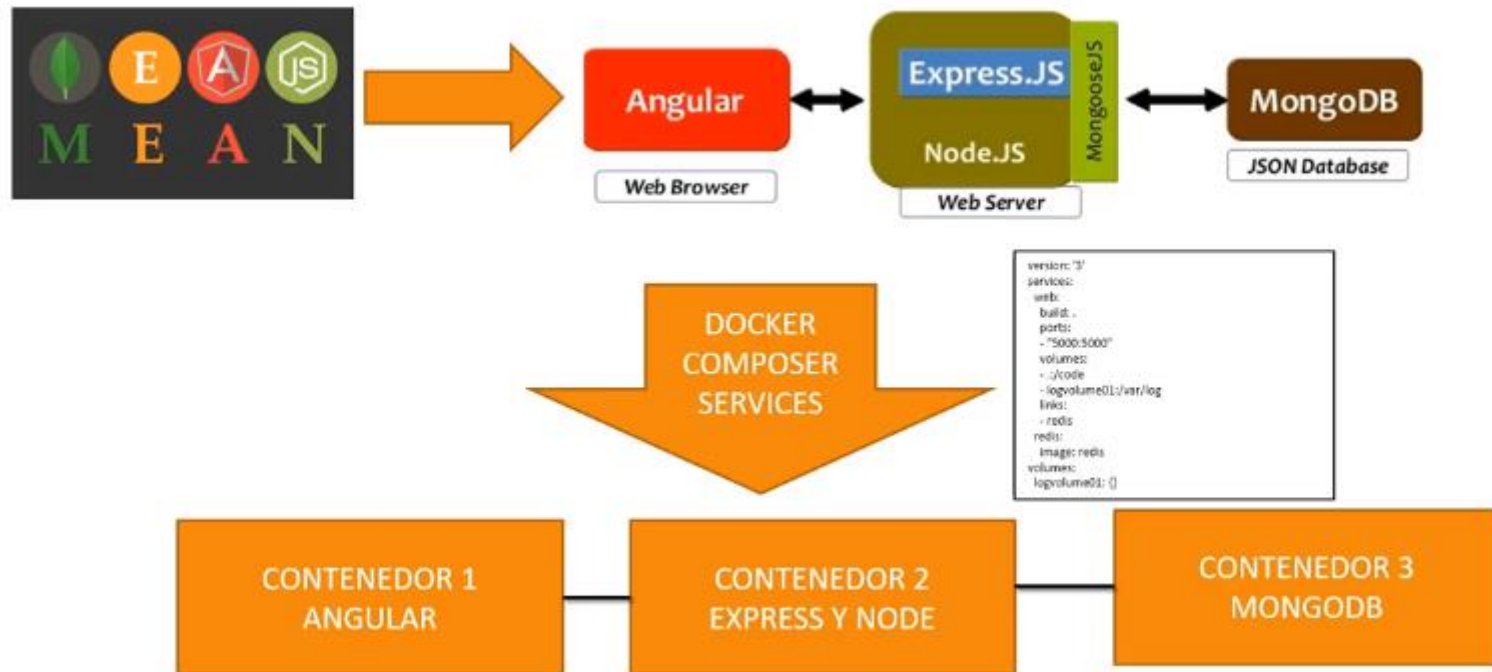


Docker Compose

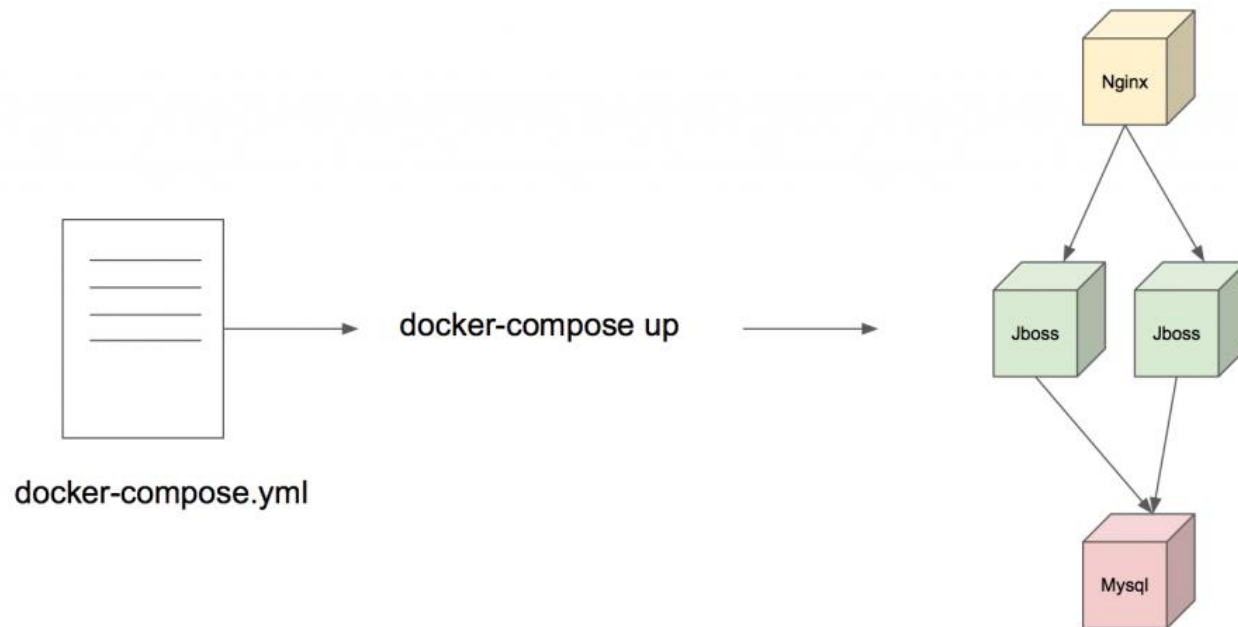
<https://docs.docker.com/compose/install/> (Linux)



Docker Compose



Docker Compose



docker-compose.yml

version: '3'

services:

wordpress:

image: wordpress

environment:

WORDPRESS_DB_HOST: dbserver:3306

WORDPRESS_DB_PASSWORD: mysqlpw

ports:

- 80:80

depends_on:

- dbserver

dbserver:

image: mysql:5.7

environment:

MYSQL_ROOT_PASSWORD: mysqlpw

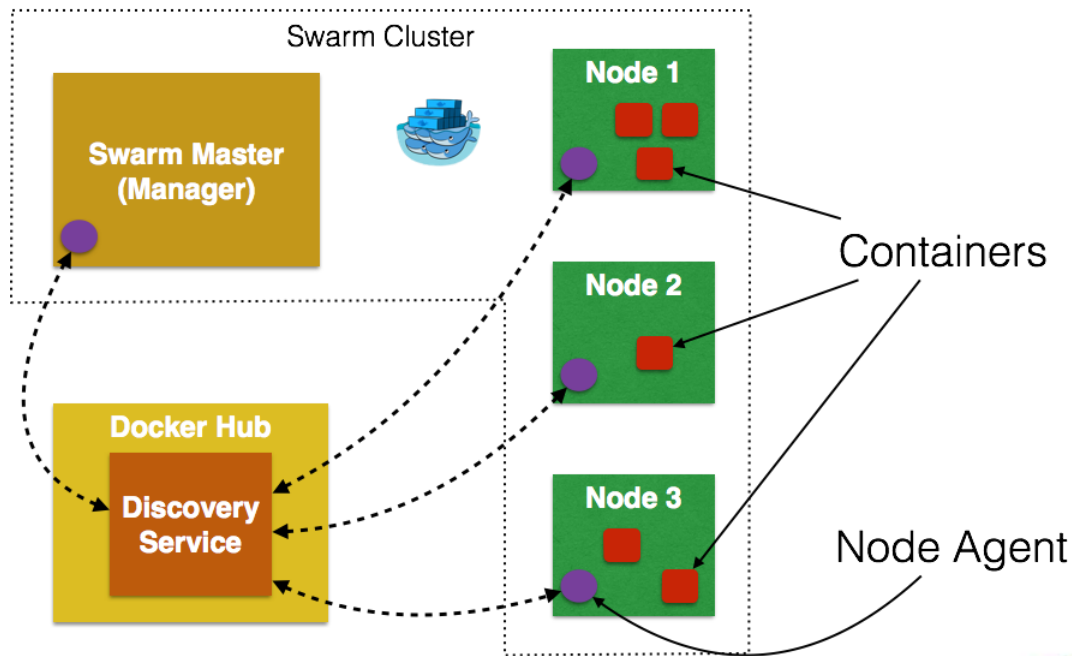
ports:

- 3307:3306

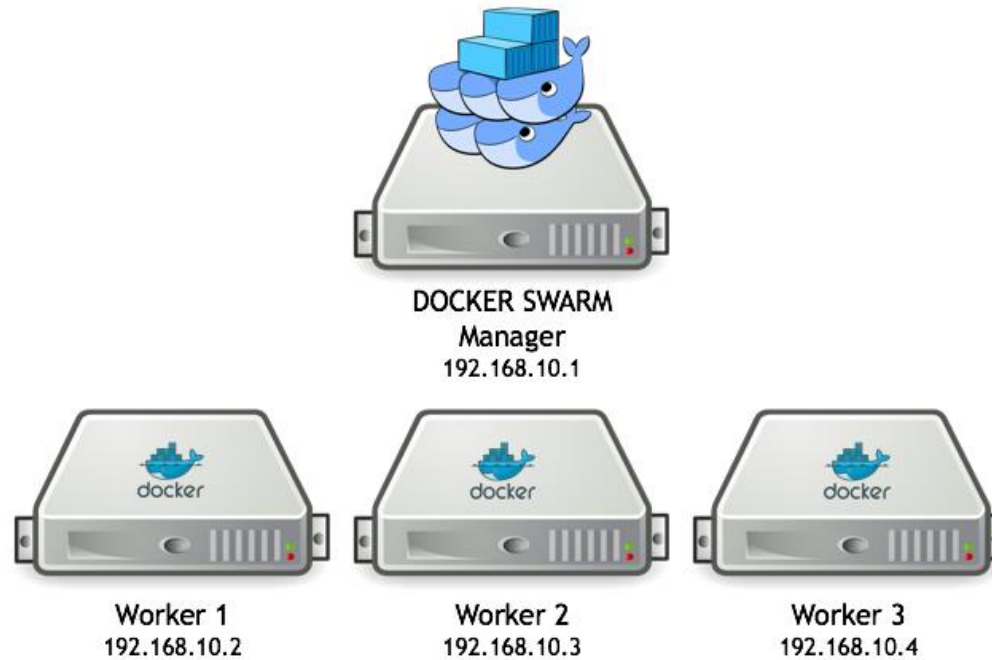
docker-compose up
docker-compose start
docker-compose stop

Docker Swarm

Docker Swarm es una herramienta nativa que permite construir un clúster de máquinas



Docker Swarm



Comandos

docker swarm init

–Inicializa un manager

docker swarm join --token SWMTKN-1-10wm8mn07nyalo26u6m0p6aq1qd3c3mpghtm3opvjqsmumb7uj-bhfa2124fncl9vdlrmc1x377n 192.168.65.3:2377

(docker swarm join-token worker)

docker info (provee información del cluster) **docker node ls**

docker node promote node3 (Se promueve como manager)

docker node demote node1 (Ya no será manager)

docker node ls (ahora lo tengo que ejecutar en node3 que es el manager)

docker swarm leave (un nodo abandona el cluster, pero no se elimina de la lista)

docker node rm nodoX (elimina el nodo del cluster)

Comandos

Crear Servicios

docker service create --replicas 1 --name helloworldswarm alpine ping www.tecgurus.net (Crea el servicio con 1 replica)

docker service ls (Se listan los servicios creados)

docker service inspect --pretty helloworldswarm (Información general del servicio)

docker service ps helloworldswarm (Se muestra la información del servicio)

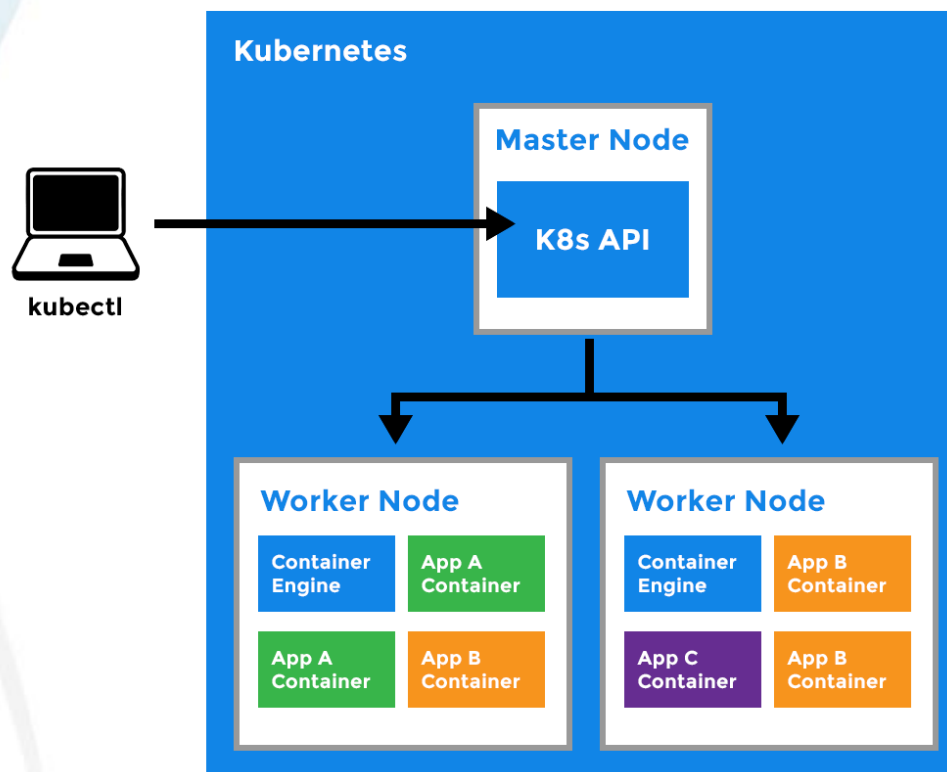
docker service logs helloworldswarm

docker service scale helloworldswarm=5

docker service rm helloworldswarm

Kubernetes

Kubernetes (referido en inglés comúnmente como “K8s”) es un sistema de código libre para la automatización del despliegue, ajuste de escala y manejo de aplicaciones en contenedores que fue originalmente diseñado por Google y donado a la Cloud Native Computing Foundation



Minikube

Instalación: <https://kubernetes.io/es/docs/tasks/tools/install-minikube/>

Iniciar minikube: minikube start

minikube start --vm-driver hyperv --hyperv-virtual-switch "Primary Virtual Switch"

```
Everything looks great. Please enjoy minikube!
```

```
[root@curso Descargas]# minikube status  
host: Running  
kubelet: Running  
apiserver: Running
```

Minikube Dashboard

minikube dashboard

The screenshot displays the Minikube Dashboard interface. The top navigation bar includes the Kubernetes logo and a search bar. The left sidebar contains a menu with options like Cluster, Namespaces, Nodes, Persistent Volumes, Roles, Storage Classes, Namespace (set to default), Overview, and Workloads. The main content area is divided into three sections: Namespaces, Nodes, and Roles.

Namespaces

Name	Labels	Status
✓ kube-public	-	Active
✓ default	-	Active
✓ kube-system	-	Active

Nodes

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)
✓ minikube	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/hostname: minikube node-role.kubernetes.io/master:	True	0.755 (37.75%)	0 (0.00%)	190 Mi (9.54%)

Roles

Name	Role Type	Namespace
kube-proxy	Role	kube-system

pod.- Grupo de contenedores

deployments.- instalación de un servicio

kubectl run nginx1 --image=nginx (deployment)
kubectl get deployments

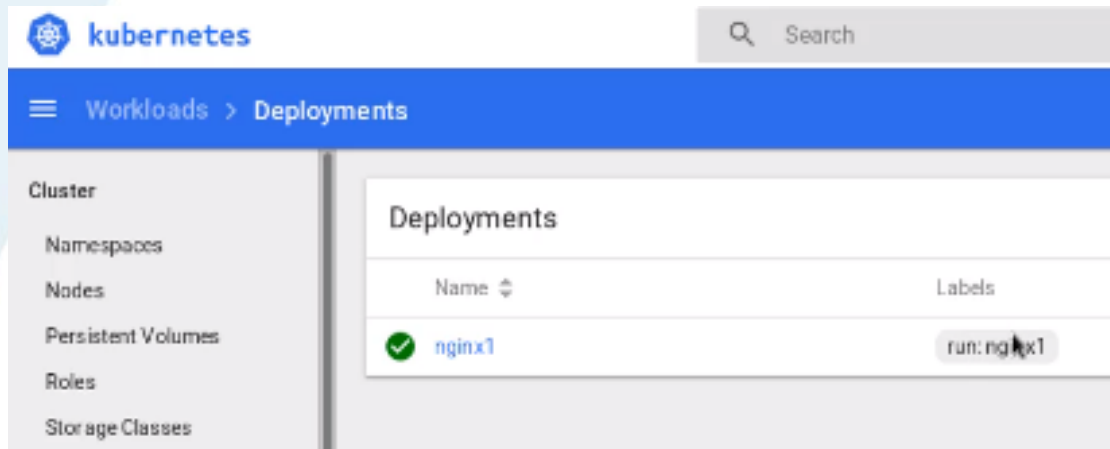
```
[root@curso ~]# kubectl get deployments
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
nginx1	1	1	1	1	52s


```
[root@curso ~]# kubectl get pods
```

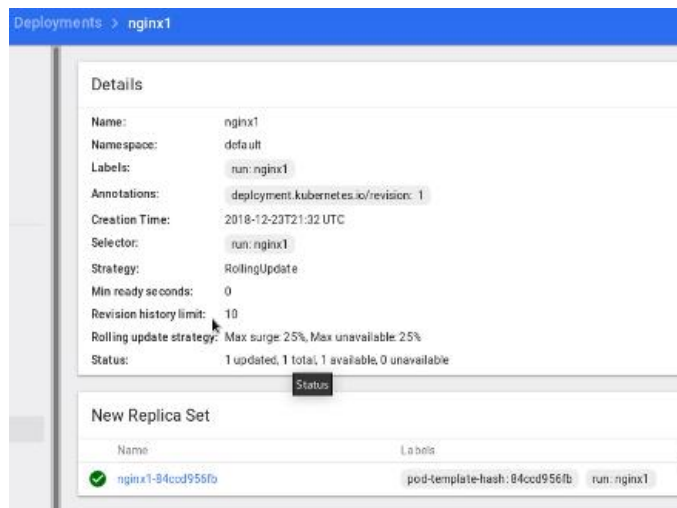
NAME	READY	STATUS	RESTARTS	AGE
nginx1-84ccd956fb-xm5jt	1/1	Running	0	80s

Deployments



A screenshot of the Kubernetes Dashboard. The top navigation bar shows the Kubernetes logo and a search bar. Below it, a blue header bar contains the breadcrumb 'Workloads > Deployments'. On the left, a sidebar lists navigation options: Cluster, Namespaces, Nodes, Persistent Volumes, Roles, and Storage Classes. The main content area is titled 'Deployments' and contains a table with two columns: 'Name' and 'Labels'. A single deployment is listed with the name 'nginx1' (marked with a green checkmark) and the label 'run: nginx1'.

Name	Labels
nginx1	run: nginx1



A screenshot showing the details of the 'nginx1' deployment. The breadcrumb is 'Deployments > nginx1'. The 'Details' section lists the following information:

- Name: nginx1
- Namespace: default
- Labels: run: nginx1
- Annotations: deployment.kubernetes.io/revision: 1
- Creation Time: 2018-12-23T21:32 UTC
- Selector: run: nginx1
- Strategy: RollingUpdate
- Min ready seconds: 0
- Revision history limit: 10
- Rolling update strategy: Max surge: 25%, Max unavailable: 25%
- Status: 1 updated, 1 total, 1 available, 0 unavailable

Below the details is a 'Status' tab and a 'New Replica Set' section. The 'New Replica Set' section shows a table with one entry:

Name	Labels
nginx1-84cd956fb	pod-template-hash: 84cd956fb run: nginx1

Acceder al Pod (Contenedores)

kubectl proxy
<http://localhost:8001>

kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
nginx1-84ccd956fb-xm5jt	1/1	Running	0	26m

[root@curso ~]#

<http://localhost:8001/api/v1/namespaces/default/pods/nginx1-84ccd956fb-xm5jt>

<http://localhost:8001/api/v1/namespaces/default/pods/nginx1-84ccd956fb-xm5jt/proxy>

```
{
  "volumeMounts": [
    {
      "name": "default-token-45299",
      "readOnly": true,
      "mountPath": "/var/run/secrets/kubernetes.io/serviceaccount"
    }
  ],
  "terminationMessagePath": "/dev/termination-log",
  "terminationMessagePolicy": "File",
  "imagePullPolicy": "Always"
},
{
  "restartPolicy": "Always",
  "terminationGracePeriodSeconds": 30,
  "dnsPolicy": "ClusterFirst",
  "serviceAccountName": "default",
  "serviceAccount": "default",
  "nodeName": "minikube",
  "securityContext": {
  },
  "schedulerName": "default-scheduler",
  "tolerations": [
    {
      "key": "node.kubernetes.io/not-ready",
      "operator": "Exists",
      "effect": "NoExecute",
      "tolerationSeconds": 300
    },
    {
      "key": "node.kubernetes.io/unreachable",
      "operator": "Exists",
      "effect": "NoExecute",
      "tolerationSeconds": 300
    }
  ],
  "priority": 0
},
{
  "status": {
    "phase": "Running",
    "conditions": [
      {
        "type": "Initialized",
        "status": "True",
        "lastProbeTime": null,
        "lastTransitionTime": "2018-12-29T21:32:56Z"
      },
      {
        "type": "Ready",
        "status": "True",
        "lastProbeTime": null
      }
    ]
  }
}
```

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.



Somos y formamos expertos en T.I.

kubectl

kubectl describe pods

```
Name:          nginx1-84ccd956fb-xm5jt
Namespace:     default
Priority:       0
PriorityClassName: <none>
Node:          minikube/10.0.2.15
Start Time:    Sun, 23 Dec 2018 22:32:56 +0100
Labels:        pod-template-hash=84ccd956fb
               run=nginx1
Annotations:   <none>
Status:        Running
IP:            172.17.0.5
Controlled By: ReplicaSet/nginx1-84ccd956fb
Containers:
```

kubectl logs nginx1-84ccd956fb-xm5jt

kubectl exec nginx1-84ccd956fb-xm5jt ls

kubectl exec -it nginx1-84ccd956fb-xm5jt ls bash

Servicios

(Pods cuando se vuelven publicos)

kubectl get deployments

kubectl get pods

```
NAME                                READY   STATUS    RESTARTS   AGE
nginx1-84ccd956fb-xm5jt            1/1     Running   0           26m
[root@curso ~]#
```

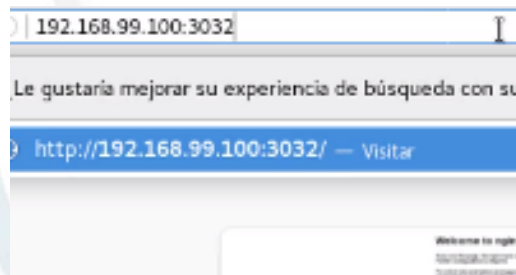
kubectl expose deployments/nginx1 --type=NodePort --port 80

```
[root@curso ~]# kubectl expose deployments/nginx1 --type="NodePort" --port 80
service/nginx1 exposed
```

kubectl get services

```
[root@curso ~]# kubectl get services
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
kubernetes    ClusterIP   10.96.0.1     <none>       443/TCP          22h
nginx1        NodePort    10.102.130.85 <none>       80:30322/TCP     23s
[root@curso ~]#
```

```
[root@curso ~]# minikube status
host: Running
kubelet: Running
apiserver: Running
kubectl: Correctly Configured: pointing to minikube-virtual-machine at 192.168.99.100
```



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and configured. Further configuration is required.

For online documentation and support please refer to <http://nginx.com>. Commercial support is available at nginx.com.

Thank you for using nginx.



Somos y formamos expertos en T.I.

Servicios

(Pods cuando se vuelven publicos)

Discovery and load balancing > Services

Name	Labels
nginx1	run: nginx1
kubernetes	component: apiserver provider: kubernetes

ancing > Services > nginx1

Details

Name:	nginx1	Connection	
Namespace:	default	Cluster IP:	10.102.130.85
Labels:	run: nginx1	Internal endpoints:	nginx1:80 TCP nginx1:30322 TCP
Creation Time:	2018-12-23T23:07 UTC		
Label selector:	run: nginx1		
Type:	NodePort		
Session Affinity:	None		

Endpoints

Host	Ports (Name, Port, Protocol)	Node
172.17.0.5	<unset>, 80, TCP	minikube

Escalar un servicio

```
[root@curso ~]# kubectl scale deployments/nginx1 --replicas=3
deployment.extensions/nginx1 scaled
```

```
[root@curso ~]# kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx1    3         3         3            3           112m
[root@curso ~]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx1-84ccd956fb-nk9cr             1/1     Running   0          19s
nginx1-84ccd956fb-nzctm             1/1     Running   0          19s
nginx1-84ccd956fb-xm5jt             1/1     Running   0          112m
[root@curso ~]#
```

```
[root@curso ~]# kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NO
DE
nginx1-84ccd956fb-nk9cr             1/1     Running   0          41s   172.17.0.7   minikube   <none>
nginx1-84ccd956fb-nzctm             1/1     Running   0          41s   172.17.0.6   minikube   <none>
nginx1-84ccd956fb-xm5jt             1/1     Running   0          112m  172.17.0.5   minikube   <none>
```


contacto@tecgurus.net
www.tecgurus.net