

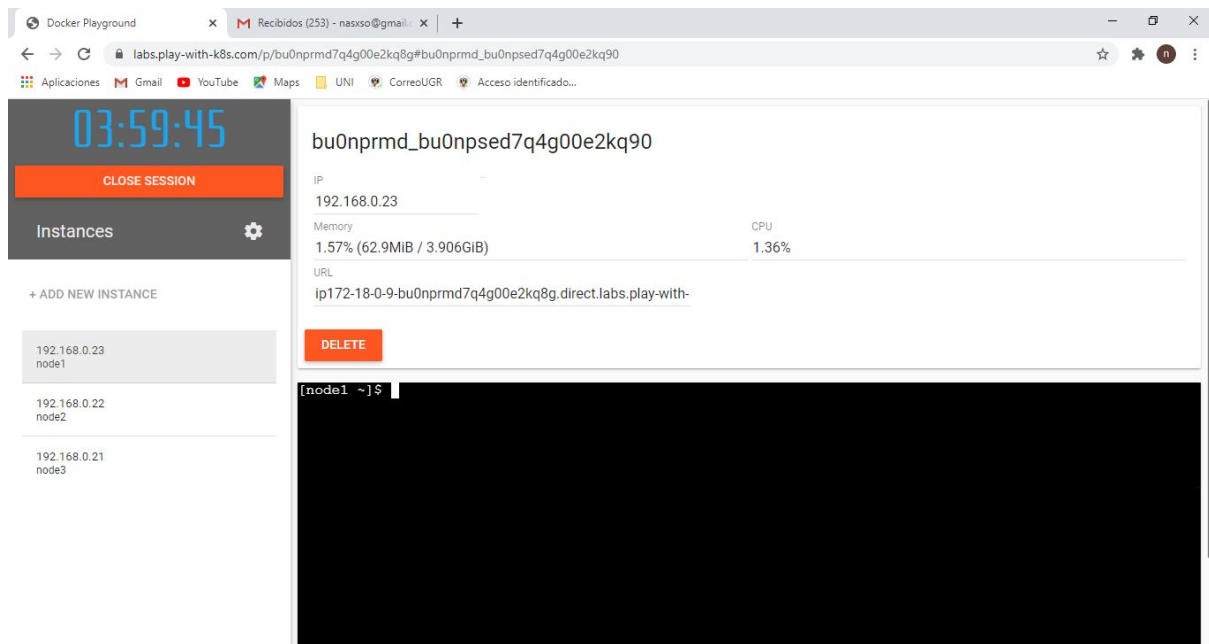
CPD_Practica_3. Docker Swarm

Combinando múltiples máquinas para la ejecución de contenedores Docker.

Objetivo: Crear un entorno basado en tres máquinas virtuales con Vagrant (+ VirtualBox) y evaluar diversas configuraciones de ejecución de contenedores Docker.

-(obligatorio): Realizar diversas capturas donde se muestren:

1. **-La creación de las máquinas virtuales con docker-machine o bien el acceso al laboratorio remoto.**



2. El inicio del manager de docker swarm.

The screenshot shows the Docker Playground interface. On the left, there's a sidebar with a timer at 03:56:32, a 'CLOSE SESSION' button, and an 'Instances' section. The 'Instances' section lists three nodes: node1 (192.168.0.23), node2 (192.168.0.22), and node3 (192.168.0.21). On the right, there's a terminal window showing the following commands and output:

```
[node1 ~]$ docker swarm init --advertise-addr 192.168.0.23
Swarm initialized: current node (7mw4uz89hi17jk7suvyaxxrqt) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3fbx5esezlbajzespl146cplopuwsnpak6ciy74ct2xpotpfjw-1gk36hr0xnkdzwuz4rdagyugo 192.168.0.23:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

[node1 ~]$ docker node ls
```

ID	ENGINE VERSION	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
7mw4uz89hi17jk7suvyaxxrqt	19.03.11	node1	Ready	Active	Leader
ya0sxi62ei6ffksczpgq8gk8q	19.03.11	node2	Ready	Active	
jawkuavx9r03mrtp0hndf51fq	19.03.11	node3	Ready	Active	

3. Ejecución del servicio web

The screenshot shows the Docker Playground interface. On the left, there's a sidebar with a timer at 03:45:46, a 'CLOSE SESSION' button, and an 'Instances' section. The 'Instances' section lists three nodes: node1 (192.168.0.23), node2 (192.168.0.22), and node3 (192.168.0.21). On the right, there's a terminal window showing the following commands and output:

```
[node1 ~]$ docker service create --name web --replicas 3 --mount type=bind,src=/etc/hostname,dst=/usr/share/nginx/html/index.html,readonly --publish published=8080,target=80 nginx
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service converged
[node1 ~]$
```

```
[node1 ~]$ docker service create --name web --replicas 3 --mount type=bind,src=/etc/hostname,dst=/usr/share/nginx/html/index.html,readonly --publish published=8080,target=80 nginx
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service converged
[node1 ~]$ curl http://192.168.0.23:8080
node3
[node1 ~]$
[node1 ~]$
[node1 ~]$ curl http://192.168.0.23:8080
node1
[node1 ~]$ curl http://192.168.0.23:8080
node3
[node1 ~]$ curl http://192.168.0.23:8080
node2
[node1 ~]$ curl http://192.168.0.23:8080
node1
[node1 ~]$ curl http://192.168.0.23:8080
node3
```

4. Cuando los 3 nodos están activos

03:56:32

CLOSE SESSION

Instances

+ ADD NEW INSTANCE

192.168.0.23
node1

192.168.0.22
node2

192.168.0.21
node3

Memory
2.23% (89.24MiB / 3.906GiB)

CPU
2.38%

URL
ip172-18-0-9-bu0nprmd7q4g00e2kq8g.direct.labs.play-with-

DELETE

```
[node1 ~]$ docker swarm init --advertise-addr 192.168.0.23
Swarm initialized: current node (7mw4uz89hi17jk7suvyaxxrqt) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3fbx5esezlbajzespl146cplpuwsnpak6ciy74ct2xpotpfjw-1gk36hr0xnkdzw
    uz4rdagyugo 192.168.0.23:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

[node1 ~]$ docker node ls
ID                                ENGINE VERSION  HOSTNAME           STATUS      AVAILABILITY  MANAGER STATUS
7mw4uz89hi17jk7suvyaxxrqt *      19.03.11        node1              Ready       Active         Leader
ya0sxi62ei6ffksczpgg8gk8q       19.03.11        node2              Ready       Active
jawkuavx9r03mrtp0hndf51fq       19.03.11        node3              Ready       Active
[node1 ~]$
```

03:36:08

CLOSE SESSION

Instances

+ ADD NEW INSTANCE

192.168.0.23
node1

192.168.0.22
node2

192.168.0.21
node3

Memory
26.95% (1.053GiB / 3.906GiB)

CPU
2.20%

URL
ip172-18-0-9-bu0nprmd7q4g00e2kq8g.direct.labs.play-with-

DELETE

```
[node1 ~]$ docker service ps web
ID                NAME      ERROR              IMAGE           NODE       DESIRED STATE  CURR
ENT STATE
wexcu0zs5ris     web.1     web.1              nginx:latest    node2      Running        Runn
ing 10 minutes ago
vj667tg4pesr     web.2     web.2              nginx:latest    node3      Running        Runn
ing 10 minutes ago
qos50nysu69w     web.3     web.3              nginx:latest    node1      Running        Runn
ing 10 minutes ago
[node1 ~]$
```

5. Cuando se cambia de escala a 2

The screenshot shows the Docker Playground interface. On the left, there's a sidebar with a clock showing 03:30:53, a 'CLOSE SESSION' button, and a list of instances: node1 (192.168.0.23), node2 (192.168.0.22), and node3 (192.168.0.21). The main panel displays the details of the selected instance, node1, including its IP, memory usage (19.92%), CPU usage (1.66%), and URL. Below this, a terminal window shows the execution of the command `docker service ps web`, which lists the current state of the 'web' service across three nodes. The output shows that the service is running on all three nodes (node1, node2, node3) with the 'nginx:latest' image. The terminal also shows the command `docker service scale web=2` being executed, which scales the service to 2 replicas. The output of this command shows that the service is now running on two nodes (node2 and node3) and is in the process of being scaled.

```
[node1 ~]$ docker service ps web
ID                NAME          IMAGE          PORTS          NODE          DESIRED STATE  CURR
ENT STATE        ERROR
wexcu0zs5ris     web.1         nginx:latest   -              node2         Running        Runn
ing 10 minutes ago
vj667tg4pesr     web.2         nginx:latest   -              node3         Running        Runn
ing 10 minutes ago
qos50nysu69w     web.3         nginx:latest   -              node1         Running        Runn
ing 10 minutes ago
[node1 ~]$ docker service scale web=2
web scaled to 2
overall progress: 2 out of 2 tasks
1/2: running
2/2: running
verify: Service converged
[node1 ~]$ docker service ps web
ID                NAME          IMAGE          PORTS          NODE          DESIRED STATE  CURR
ENT STATE        ERROR
wexcu0zs5ris     web.1         nginx:latest   -              node2         Running        Runn
ing 15 minutes ago
vj667tg4pesr     web.2         nginx:latest   -              node3         Running        Runn
ing 15 minutes ago
[node1 ~]$
```

```
[node1 ~]$ curl http://192.168.0.23:8080
node3
[node1 ~]$ curl http://192.168.0.23:8080
node2
[node1 ~]$ curl http://192.168.0.23:8080
node3
[node1 ~]$ curl http://192.168.0.23:8080
node2
[node1 ~]$ curl http://192.168.0.23:8080
node3
[node1 ~]$ curl http://192.168.0.23:8080
node2
[node1 ~]$ curl http://192.168.0.23:8080
node3
[node1 ~]$ curl http://192.168.0.23:8080
node2
[node1 ~]$ curl http://192.168.0.23:8080
node3
[node1 ~]$
```

6. Cuando apagamos un nodo activo y sólo ejecuta un nodo,

The screenshot shows the Docker Playground interface. On the left, there's a sidebar with a clock showing 03:22:17, a 'CLOSE SESSION' button, and a list of instances: node1 (192.168.0.23), node2 (192.168.0.22), and node3 (192.168.0.21). The main panel displays the details of the selected instance, node3, including its IP, memory usage (26.54%), CPU usage (1.45%), and URL. Below this, a terminal window shows the execution of the command `systemctl stop docker` on node3. The output of this command shows that the Docker service is being stopped on node3.

```
[node3 ~]$ systemctl stop docker
[node3 ~]$
```

7. y la activación automática del segundo nodo.

Como vemos antes el nodo uno no salía y al estar el nodo 3 en shutdown se ha arrancado el nodo1.

The screenshot shows the Docker Playground interface. On the left, there's a sidebar with a clock showing 03:21:08, a 'CLOSE SESSION' button, and a list of instances: node1 (192.168.0.23), node2 (192.168.0.22), and node3 (192.168.0.21). The main area displays resource usage: Memory at 26.65% (1.041GiB / 3.906GiB) and CPU at 12.15%. Below this is a 'DELETE' button. The terminal window shows the command `docker service ps web` and its output:

ID	NAME	IMAGE	PORTS	NODE	DESIRED STATE	CURR
wexcu0zs5ris	web.1	nginx:latest		node2	Running	Runn
fhlo8w8rhhh5	web.2	nginx:latest		node1	Running	Runn
vj667tg4pesr	_ web.2	nginx:latest		node3	Shutdown	Runn

8. (opcional): Capturas de diversas ejecuciones en la plataforma KataKoda

The screenshot shows the KataKoda platform interface. The main content area is titled '¿Qué es un contenedor?' and includes a 'Procesos' section. It explains that containers are normal Linux processes with additional configuration. It provides the command `docker run -d --name=db redis:alpine` and the command `ps aux | grep redis-server`. On the right, there's a terminal window showing the command prompt `PS` and the command `ps` being executed. The terminal output shows the command prompt `PS` and the command `ps` being executed.

swad.ugres | practica_3_CPD_v1.pdf | CPD - Google Drive | CPD_Practica3_DOCKER_ignaci | ¿Qué es un contenedor? | Con | + -

katacoda.com/courses/container-runtimes/what-is-a-container

Aplicaciones | Gmail | YouTube | Maps | UNI | CorreoUGR | Acceso identificado...

O'REILLY Katakoda

INFORMACIÓN DE CREACIÓN DE ESCENARIOS | DESCRIPCIÓN GENERAL Y SOLUCIONES DE KATACODA | PRUEBA O'REILLY | INICIAR SESIÓN >

¿Qué es un contenedor?

Paso 1 de 6 ▶

`docker run -d --name=db redis:alpine`

El contenedor Docker lanza un proceso llamado `redis-server`. Desde el host, podemos ver todos los procesos en ejecución, incluidos los iniciados por Docker.

`ps aux | grep redis-server` ✓

Docker puede ayudarnos a identificar información sobre el proceso, incluido el PID (ID de proceso) y PPID (ID de proceso principal) a través de `docker top db` ✓

¿Quién es el PPID? Utilice `ps aux | grep <ppid>` para encontrar el proceso principal. Es probable que sea Containerd.

El comando `pstree` enumerará todos los subprocesos. Vea el árbol de procesos de Docker usando `pstree -c -p -A $(pgrep dockerd)`

Terminal +

```
PS
$ docker ejecutar -d --name = db redis: alpine
No se puede encontrar la imagen 'redis: alpine' localmente
alpine: tirando de la biblioteca / redis
df20fa9351a1: Extracción completa
9b8c029ceab5: Extracción completa
e983a1eb737a: Extracción completa
de9de2cb6149: Pull complete
6a04900c891e: Tira completo
be5e938aa8ff: Extracción completa
Resumen: sha256: 4015d7a6a0901920a3adfae3a538bf8489325738153948f95ca2b637944bdf5
Estado: imagen más reciente descargada para redis: alpine
caaec1c4374266bd3f85a2b929f54ddb9a5dde40a1cc808687af53449431dd8d
$ ps aux | grep redis-server
999 1915 1.0 1.2 28952 12824? Ssl 10:32 0:00 servidor redis
root 1964 0.0 0.0 14220960 pts / 0 S + 10:32 0:00 grep --color = auto servidor redis
$ docker top db
UID PID PPID C STIME TTY
HORA CMD
999 1915 1900 0 10:32?
00:00:00 servidor redis
PS
```

swad.ugres | practica_3_CPD_v1.pdf | CPD - Google Drive | CPD_Practica3_DOCKER_ignaci | Introducción a Kubeadm | Kuti | + -

katacoda.com/courses/kubernetes/getting-started-with-kubeadm#

Aplicaciones | Gmail | YouTube | Maps | UNI | CorreoUGR | Acceso identificado...

O'REILLY Katakoda

INFORMACIÓN DE CREACIÓN DE ESCENARIOS | DESCRIPCIÓN GENERAL Y SOLUCIONES DE KATACODA | PRUEBA O'REILLY | INICIAR SESIÓN >

Introducción a Kubeadm

Paso 3 de 6 ▶

el cluster proporcionando la dirección IP del nodo principal.

`kubeadm join --discovery-token=unsafe-skip-ca-verification --token=102952.1a7dd4cc8d1f4cc5 172.17.0.44:6443` ✓

Este es el mismo comando que se proporciona después de inicializar el maestro.

La `--discovery-token=unsafe-skip-ca-verification` etiqueta se utiliza para omitir la verificación de Discovery Token. Como este token se genera dinámicamente, no pudimos incluirlo en los pasos. Cuando esté en producción, use el token proporcionado por `kubeadm init`.

SEGUIR

Anfitrión terminal 1 +

```
etcd-controlplane 1/1 En funcionamiento 0 48s
kube-apiserver-controlplane 1/1 Ejecutando 0 27s
kube-controller-manager-controlplane 1/1 Running 0 43s
kube-proxy-c6bkm 1/1 Ejecutando 0 95s
kube-planificador-controlplane 1/1 En ejecución 1 28s
weave-net-87phf 1/2 Corriendo 0 5s
controlplane $ kubeadm lista de tokens
TOKEN TTL EXPIRA USOS DESCRIPCIÓN
GRUPOS EXTRA
102952.1a7dd4cc8d1f4cc5 23h 2020-10-11T10: 34: 27Z autenticación, firmando el bootstrap predeterminado
token generado por 'kubeadm init'. system: bootstrappers: kubeadm: token de nodo predeterminado
Host de terminal 2
7.0.44: 6443
[Preflight] Ejecución de comprobaciones previas al vuelo
[Preflight] Leyendo la configuración del clúster ...
[Preflight] FYI: Puede ver este archivo de configuración con 'kubectl -n kube-system get cm kubeadm-config -ml'
[kubelet-start] Descargando la configuración para kubelet desde el ConfigMap "kubelet-config-1.14" en el espacio de nombres del sistema kube
[kubelet-start] Escribiendo la configuración de kubelet en el archivo "/var/lib/kubelet/config.yaml"
[kubelet-start] Escribiendo un archivo de entorno kubelet con banderas en el archivo "/var/lib/kubelet/kubea
[kubelet-start] Activando el servicio kubelet
[kubelet-start] Esperando que kubelet realice el TLS Bootstrap ...
```