



UNIVERSIDAD DE GRANADA

Centro de Procesamiento de Datos

Práctica 3 - Docker Swarm: Combinando múltiples máquinas para la ejecución de contenedores Docker

Arturo Alonso Carbonero

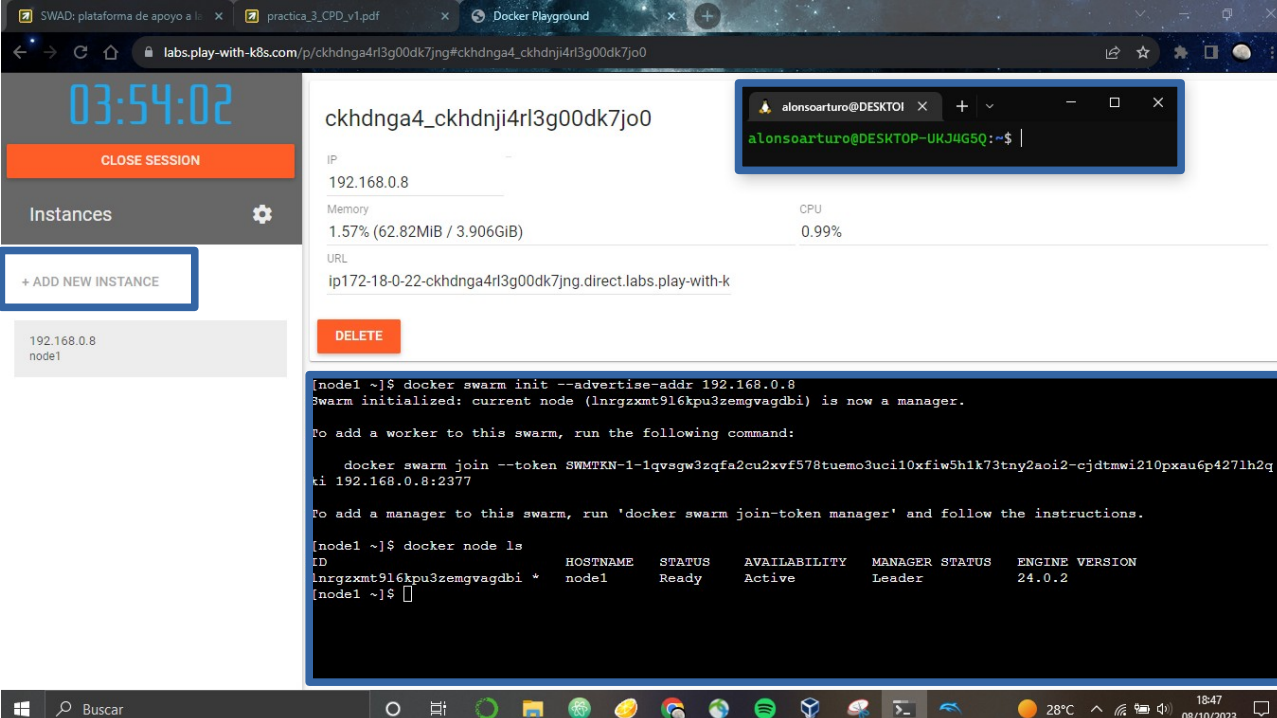
Apartado obligatorio – Docker Swarm

Una vez accedemos al laboratorio virtual, podemos crear diferentes instancias que se corresponden con máquinas independientes. Para ello, basta con hacer click en el apartado *ADD NEW INSTANCE* que aparece en el menú de la izquierda. Para la práctica, las capturas mostrarán al completo la pantalla, así como una pequeña terminal con mi nombre de usuario, para de esta forma asegurar su unicidad.

Antes de comenzar con la resolución del ejercicio, es necesario comprender que es un *swarm*. El modo *swarm* de *Docker* permite gestionar clústeres de demonios *Docker*, de forma que se puede simular, por ejemplo, un servidor con balanceo de carga.

Una vez hemos creado la primera máquina o nodo dentro del laboratorio, hacemos uso de la orden **docker swarm init** para inicializar el swarm. En este caso, lo inicializamos además utilizando la opción **--advertise-addr** para indicar la dirección IP a la que el resto de nodos deberán conectarse. De esta forma, hemos hecho que la máquina con nombre *node1* sea la máquina principal del *swarm*.

Tras esto, el resultado obtenido ofrece la orden que deben ejecutar los otros nodos que deseen ser añadidos al *swarm* como trabajadores. Por otra parte, haciendo uso de la opción **docker swarm ls**, podemos listar las máquinas pertenecientes al mismo, tal y como se ve en la siguiente imagen.



The screenshot displays the Docker Playground interface. On the left, a sidebar shows a clock at 03:54:02, a 'CLOSE SESSION' button, and an 'Instances' section with a '+ ADD NEW INSTANCE' button. Below this, a list of instances shows '192.168.0.8 node1'. The main panel displays details for the instance 'ckhdnga4_ckhdnji4rl3g00dk7jo0', including its IP (192.168.0.8), memory usage (1.57%), CPU usage (0.99%), and a URL. A 'DELETE' button is also present. A terminal window is open, showing the execution of the following commands:

```
[node1 ~]$ docker swarm init --advertise-addr 192.168.0.8
Swarm initialized: current node (lnrgzxt916kpu3zemgvagdbi) is now a manager.

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

    docker swarm join --token SWMTKN-1-lqvsgw3zqfa2cu2xvf578tuemo3uci10xfiw5h1k73tny2aoi2-cjdtmwi210pxau6p4271h2qi 192.168.0.8:2377

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

[node1 ~]$ docker node ls
ID                HOSTNAME        STATUS         AVAILABILITY   MANAGER STATUS   ENGINE VERSION
lnrgzxt916kpu3zemgvagdbi * node1           Ready          Active          Leader           24.0.2
[node1 ~]$
```

Creación del nodo *node1* y el *swarm*

A continuación, creamos otra máquina de la misma forma, obteniendo así el segundo nodo (*node2*). Dentro de esta instancia, hay que ejecutar el comando indicado por la orden anterior. Dicho comando es **docker swarm join** con la opción **--token**. El comando permite a un nodo unirse al *swarm* correspondiente al *token* a través de la dirección de la máquina principal del *swarm* por el puerto 2377.

The screenshot shows the Docker Playground interface. On the left, a sidebar displays a timer at 03:35:56, a 'CLOSE SESSION' button, and a list of instances: 'node1' (192.168.0.8) and 'node2' (192.168.0.7). The main panel shows details for 'ckhdnga4_ckhdtoa4rl3g00dk7jog' (IP: 192.168.0.7, Memory: 1.48%, CPU: 0.80%). A terminal window at the bottom shows the command `docker swarm join --token SWMTKN-1-1qvsgw3zqfa2cu2xvf578tuemo3uci10xfiw5hk73tny2aci2-cjdtmwi210pxau6p4271b2gki 192.168.0.8:2377` being executed, with the output 'this node joined a swarm as a worker.'

Máquina node2 añadida al swarm

Repetimos exactamente el mismo proceso con un tercer nodo, y comprobamos que se han añadido al *swarm* de forma correcta mediante la orden **docker swarm ls**.

The screenshot shows the Docker Playground interface with three nodes. The sidebar lists 'node1' (192.168.0.8), 'node2' (192.168.0.7), and 'node3' (192.168.0.6). The main panel shows details for 'ckhdnga4_ckhdnji4rl3g00dk7jo0' (IP: 192.168.0.8, Memory: 1.89%, CPU: 1.50%). A terminal window at the bottom shows the command `docker node ls` being executed, with the output:

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
lnrgzxm7916kpu3zemgvgadbi *	node1	Ready	Active	Leader	24.0.2
2pvkyygfv0fq20wpfrd27dfjn	node2	Ready	Active		24.0.2
l75c2h90ascm98sq7sbkijebr	node3	Ready	Active		24.0.2

Swarm con los tres nodos

Con esto disponemos de un *swarm* con tres máquinas diferentes. A continuación, mediante la orden **docker service create**, creamos un servicio web *Nginx* con tres réplicas para repartir la carga. Concretamente, lanzamos el comando con las siguientes opciones:

- **--name:** Establece el nombre del servicio.
- **--replicas:** Fija el número de réplicas.
- **--mount:** Añade un punto de montaje al servicio. En este caso, solo será de lectura.
- **--publish:** Permite configurar los puertos a emplear. En este caso, el 8080 sobre el 80.

The screenshot shows the Docker Playground interface in a web browser. On the left, there's a sidebar with a clock showing 03:27:38, a 'CLOSE SESSION' button, and a list of instances: 'node1' (192.168.0.8), 'node2' (192.168.0.7), and 'node3' (192.168.0.6). The main panel displays details for instance 'ckhdnga4_ckhdnji4rl3g00dk7jo0', including its IP (192.168.0.8), memory usage (7.60%), CPU usage (1.35%), and a URL. A terminal window is open, showing the command to create a service: `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`. The terminal output shows the service is running on all three nodes and has converged. A small terminal window in the top right shows the user 'alonsoarturo@DESKTOP-UKJ4G5Q' at a shell prompt.

Creación del servicio

Para comprobar que el resultado es correcto, podemos hacer uso de la orden **curl** desde cualquiera de las máquinas a la dirección del **swarm** a través del puerto 8080. El resultado es el mostrado en la siguiente imagen, donde se puede ver que la carga se reparte por turnos (*Round-Robin*), ya que esta es la configuración por defecto de *Nginx*.

The screenshot shows the Docker Playground interface. On the left, there's a sidebar with a clock showing 03:22:58, a 'CLOSE SESSION' button, and a list of instances: node1 (192.168.0.8), node2 (192.168.0.7), and node3 (192.168.0.6). The main panel displays details for instance 'ckhdnga4_ckhdnji4rl3g00dk7jo0', including its IP (192.168.0.8), memory usage (7.63%), CPU usage (1.45%), and URL. A terminal window is open, showing the execution of 'curl http://192.168.0.8:8080' from both node1 and node2, demonstrating load balancing.

Reparto de la carga entre los nodos

Podemos comprobar los nodos con el servicio mediante la orden **docker service ps** sobre un servicio concreto, en este caso “web”.

The screenshot shows the Docker Playground interface, similar to the previous one but with a different clock time (03:22:23). The terminal window now displays the output of the command 'docker service ps web', showing a table of services and their status across the nodes.

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
nj3cx8bmejfx	web.1	nginx:latest	node3	Running	Running 5 minutes ago		
93c2o3o863x6	web.2	nginx:latest	node1	Running	Running 5 minutes ago		
zliammqtfv2n	web.3	nginx:latest	node2	Running	Running 5 minutes ago		

Nodos con el servicio

Haciendo uso de la opción **scale** de la orden **docker service**, podemos indicar cuántos nodos queremos utilizar. En la siguiente imagen se muestra como, al escalar de 3 a 2 nodos, las peticiones pasan a repartirse entre dos de los nodos, que son los contenedores que ahora se lanzan. En este caso, son *node1* y *node3*. Podemos comprobar el resultado de igual forma que el apartado anterior.

The screenshot shows the Docker Playground interface. On the left, a sidebar lists three instances: *node1* (IP 192.168.0.8), *node2* (IP 192.168.0.7), and *node3* (IP 192.168.0.6). The main panel displays details for instance *ckhdnga4_ckhdnji4rl3g00dk7jo0*, including its IP (192.168.0.8), memory usage (7.63%), CPU usage (1.44%), and a URL. A terminal window is open, showing the command `docker service scale web=2` being executed on *node1*. The output indicates that the service has been scaled to 2 nodes. Below this, the command `curl http://192.168.0.8:8080` is run on *node1*, and the output shows that the service is running on *node3*. A table at the bottom lists the service's components:

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
aj3cx8bmejfx	web.1	nginx:latest	node3	Running	Running 6 minutes ago		
33c2o3o863x6	web.2	nginx:latest	node1	Running	Running 6 minutes ago		

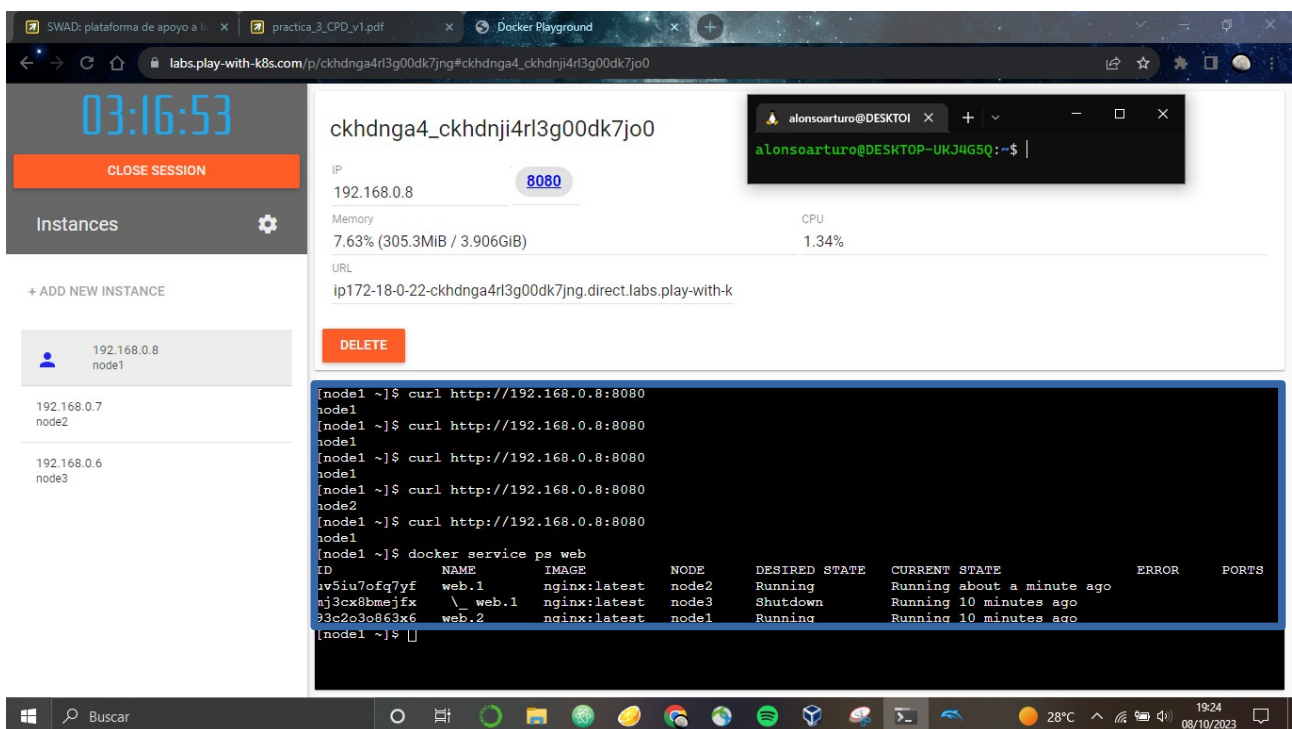
Escalado de 3 a 2 nodos

Además, si uno de los nodos falla, el sistema recupera el nodo que no se está utilizando a los pocos segundos de forma automática. Si tiramos el servicio en la máquina *node3*, mediante la orden **systemctl stop**, podemos comprobar en el nodo principal que, de forma automática, se recupera el segundo nodo, el cual se había dejado de utilizar.

The screenshot shows the Docker Playground interface. On the left, the same three instances are listed. The main panel displays details for instance *ckhdnga4_ckhe2ti4rl3g00dk7js0*, including its IP (192.168.0.6), memory usage (6.48%), CPU usage (8.01%), and a URL. A terminal window is open, showing the command `systemctl stop docker` being executed on *node3*. The output indicates that the service has been stopped.

Contenedor node3 sin servicio

Cabe destacar que, al listar los nodos, podemos ver como en el tercer nodo se indica que el servicio ha sido apagado y que este no está disponible.

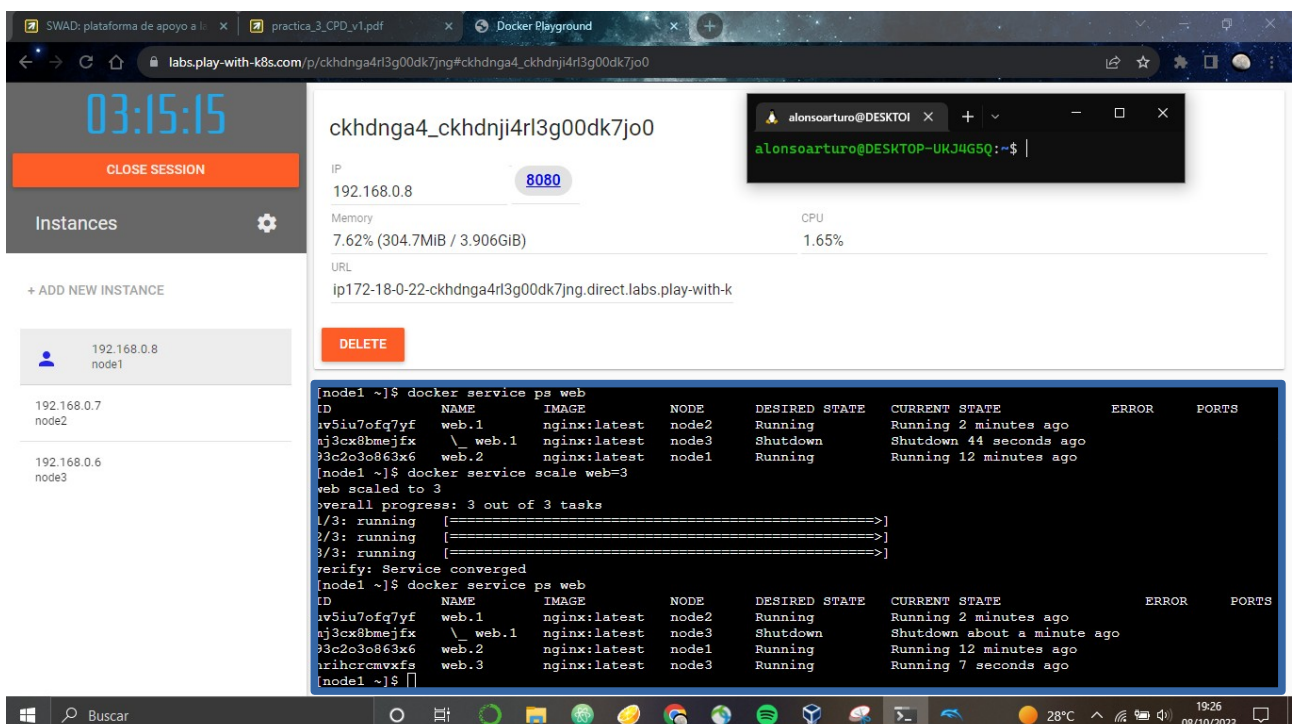


The screenshot shows the Docker Playground interface with three nodes listed on the left: node1 (192.168.0.8), node2 (192.168.0.7), and node3 (192.168.0.6). The main panel displays details for node1, including its IP, memory usage, CPU usage, and URL. A terminal window shows the command `curl http://192.168.0.8:8080` being executed on node1. Below the terminal, the command `docker service ps web` is shown, along with its output table.

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
lv5iu7ofq7yf	web.1	nginx:latest	node2	Running	Running about a minute ago		
nj3cx8bmejfx	web.1	nginx:latest	node3	Shutdown	Running 10 minutes ago		
93c2o3o863x6	web.2	nginx:latest	node1	Running	Running 10 minutes ago		

Contenedor node2 recuperado

Sin embargo, si el nodo que ha fallado vuelve a tener servicio, en este caso levantándolo mediante **systemctl start**, basta con re-escalar el número de nodos de nuevo a 3 y el sistema recuperará el nodo que había fallado de forma automática.



The screenshot shows the Docker Playground interface with three nodes listed on the left: node1 (192.168.0.8), node2 (192.168.0.7), and node3 (192.168.0.6). The main panel displays details for node1, including its IP, memory usage, CPU usage, and URL. A terminal window shows the command `docker service scale web=3` being executed on node1. Below the terminal, the command `docker service ps web` is shown, along with its output table.

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
lv5iu7ofq7yf	web.1	nginx:latest	node2	Running	Running 2 minutes ago		
nj3cx8bmejfx	web.1	nginx:latest	node3	Shutdown	Shutdown 44 seconds ago		
93c2o3o863x6	web.2	nginx:latest	node1	Running	Running 12 minutes ago		

Recuperación de todos los nodos