

Memoria Práctica 3.

Spark y Kubernetes

Computación de altas prestaciones

Juan González Domínguez
Colman López Alonso

Parte 1. Spark

Todas las cuestiones de esta parte se encuentran respondidas en profundidad en el notebook entregado.

```
# Map devuelve el mismo número de elementos que la entrada (K = N)
print(f"Quijote_count: {quijote.count()}\nChars_per_line: {charsPerLine.count()}")

Quijote_count: 2186
Chars_per_line: 2186

# Flatmap aumenta resultado por cada elemento de entrada (K >= N)
print(f"Quijote_count: {quijote.count()}\nAll_words: {allWords.count()}")

Quijote_count: 2186
All_words: 187018

# Filter filtra elementos por los que normalmente (K <= N, pero puede ser K = N o K = 0 si todos pasan o ninguno pasa)
print(f"All_words: {allWords.count()}\nAll_words_no_articles: {allWordsNoArticles.count()}")

All_words: 187018
All_words_no_articles: 178167

# Distinct borra elementos repetidos (K <= N o K = N si no hay)
print(f"All_words: {allWords.count()}\nAll_words_unique: {allWordsUnique.count()}")

All_words: 187018
All_words_unique: 22211

# Sample selecciona una muestra aleatoria (K <= N)
print(f"All_words: {allWords.count()}\nSample_words: {sampleWords.count()}")

All_words: 187018
Sample_words: 37251

# Union une dos RDDs (K = N1 + N2)
print(f"Sample_unique: {sampleUnique.count()}\nSample_words: {sampleWords.count()}\nWeirdSampling: {weirdSampling.count()}")

Sample_unique: 4430
Sample_words: 37251
WeirdSampling: 41681
```

SQL:

```
spark.sql("select * from reddit where not over_18").show()
```

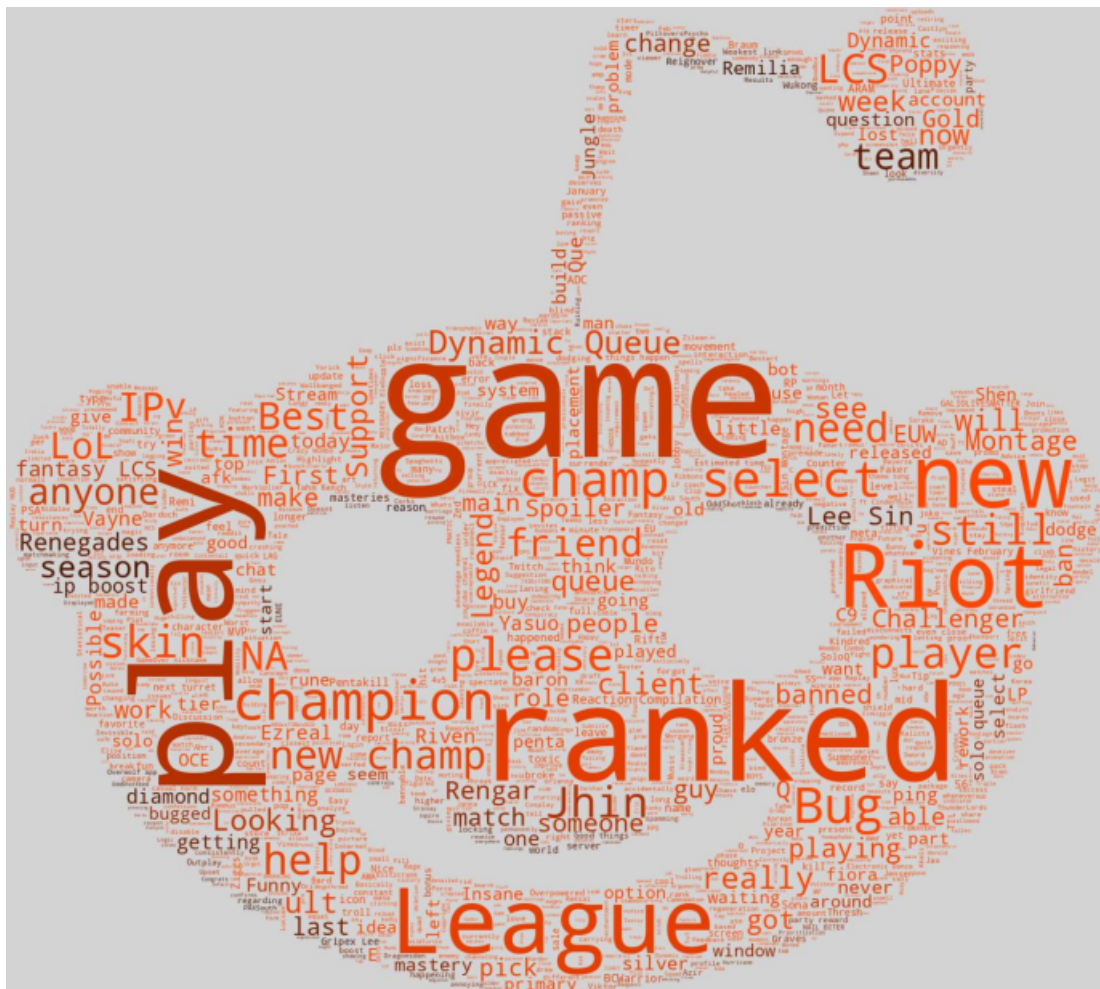
```
spark.sql("select created_utc * 2 from reddit").show()
```

```
spark.sql("select LN(created_utc * 2) from reddit").show()
```

```
spark.sql("select author, subreddit, count(*) from reddit where not over_18 group by author, subreddit").toPandas()
```

```
spark.sql("select author, subreddit, count(*) from reddit group by author, subreddit").toPandas()
```

```
spark.sql("Select author from reddit where length(selftext)>1000 group by author").toPandas()
```



Parte 2. Kubernetes

- Exercise 1: our own k8s cluster

Se han seguido las indicaciones del tutorial expuesto en el enunciado. No hay nada que entregar en este ejercicio.

- Exercise 2: defining master and worker images

En este ejercicio hay que crear las imágenes de **docker** adecuadas para *base*, *master* y para *worker*. El comando de **docker** a utilizar es **build**, de la siguiente manera:

~ > docker build -t image_name -f dockerfile .

~ > docker build -t base -f base.Dockerfile .

~ > docker build -t master -f master.Dockerfile .

~ > docker build -t image_name -f dockerfile .

```
> docker images
REPOSITORY          TAG          IMAGE ID          CREATED           SIZE
worker              latest       4cb5aa8fa100     3 minutes ago    1.21GB
master              latest       53c232c0d7f7     5 minutes ago    1.21GB
base                latest       783e2ff310bd     12 minutes ago   1.21GB
gcr.io/k8s-minikube/kicbase v0.0.42     dbc648475405     11 days ago      1.2GB
ubuntu              22.04       e4c58958181a     6 weeks ago      77.8MB
```

Tras esto, con las imágenes creadas, podemos crear una network para que los contenedores que lancemos estén en la misma red o podemos usar una de las que vienen creadas en docker por defecto.

Independientemente de si queremos crear una nueva red interna para los contenedores o no, el siguiente paso es crear los contenedores, con el siguiente comando:

~ > **docker run --net net_name --name cont_name -d -p port image**

--net para especificar la red, --name para especificar el nombre, -d para correrlo en background y -p para especificar los puertos, los comandos quedarían así.

~ > **docker run --net testing --name container-master -d -p 7077:7077 -p 8080:8080 master**


~ > **docker run --net testing --name container-worker -d -p 7076:7077 -p 8081:8081 worker**

```
docker ps - terminal
File Edit View Search Terminal Help
> docker run --net testing --name container-master -d -p 7077:7077 -p 8080:8080 master
b23c33ef4b08c837cb2d70d5a4d31bf050a2d69e240bb2ce039f53ad0be4e4be
> docker run --net testing --name container-worker -d -p 7076:7077 -p 8081:8081 worker
da86605f7864abcb30875ace8b14f5161d9dd81a877bdcc6c0aef84a52b21c5
docker
> docker ps
CONTAINER ID   IMAGE    COMMAND                  CREATED        STATUS        PORTS                                                                 NAMES
da86605f7864   worker   "/root/worker.sh"        4 seconds ago Up 3 seconds   0.0.0.0:8081->8081/tcp, 0.0.0.0:7076->7077/tcp   container-worker
b23c33ef4b08   master   "/root/master.sh"        22 seconds ago Up 21 seconds   0.0.0.0:7077->7077/tcp, 0.0.0.0:8080->8080/tcp   container-master
```

Una vez correctamente creados los contenedores, podemos acceder a la dirección <http://localhost:8080/> y acceder al dashboard del Spark Master. Podemos ver que el worker también se ha lanzado correctamente.

← → ↻ 🔍 http://127.0.0.1:8080/

Parrot OS Hack The Box OSINT Services Vuln DB Privacy and Security Learning Resources YouTube ChatGPT Gmail UAM Outlook Moodle

 **Spark Master at spark://spark-master:7077**

URL: spark://spark-master:7077
Alive Workers: 1
Cores in use: 4 Total, 0 Used
Memory in use: 10.6 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory
worker-20231118201400-172.18.0.3-35229	172.18.0.3:35229	ALIVE	4 (0 Used)	10.6 GiB (0.0 B Used)

Running Applications (0)

Completed Applications (0)

- Exercise 3: make everything work

Para continuar con el siguiente ejercicio, hay que completar los ficheros master.yaml, worker.yaml un fichero master-service.yaml. Hemos tenido que realizar algunos cambios en los ficheros de configuración yaml para el correcto funcionamiento en kubernetes.

Para probar todo el entorno, los pasos son los siguientes.

Iniciar el servidor con:

~ > minikube start

Ahora hay que realizar este comando para poder ejecutar las imágenes en el servidor de kubernetes.

~ > eval \$(minikube docker-env)

Tras esto, ejecutamos los mismos comandos de antes para subir las imágenes al kubernetes, esto se realiza con el comando de **docker build** con cada una de las imágenes, como hemos mostrado previamente.

Una vez subidas las imágenes, el siguiente paso es aplicar las configuraciones de los yaml, mediante los siguientes comandos.

Para el master:

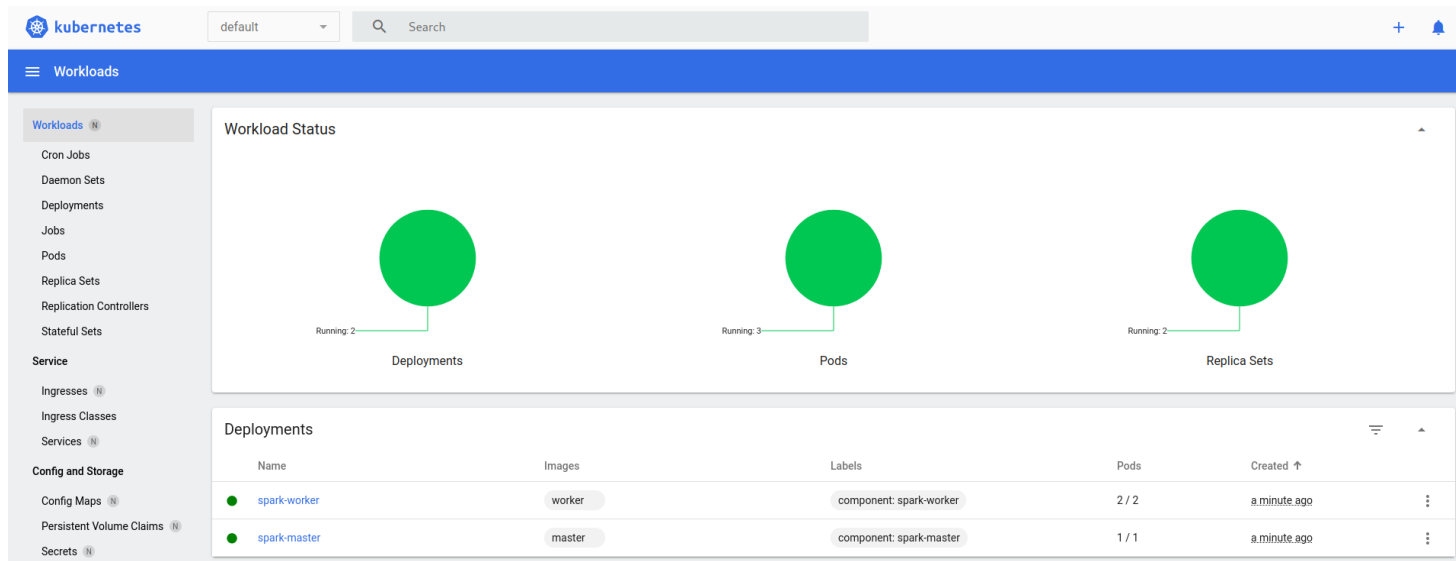
~ > kubectl apply -f master.yaml

Para el worker

~ > kubectl apply -f worker.yaml

Una vez ejecutados estos comandos se pueden ver ya los deployments en el dashboard de kubernetes (que se puede sacar con el comando **minikube dashboard**)

En las siguientes imágenes podemos ver los resultados obtenidos a través de la dashboard.



Deployments									
Name	Images	Labels	Pods	Created ↑					
spark-worker	worker	component: spark-worker	2 / 2	2 minutes ago					
spark-master	master	component: spark-master	1 / 1	2 minutes ago					

Pods									
Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑	
spark-worker-7c9db96ddd-249br	worker	component: spark-worker pod-template-hash: 7c9db96ddd	minikube	Running	0	~ 0.00m	140.36Mi	2 minutes ago	
spark-worker-7c9db96ddd-8zhgt	worker	component: spark-worker pod-template-hash: 7c9db96ddd	minikube	Running	0	~ 0.00m	137.57Mi	2 minutes ago	
spark-master-57759b7dbc-xdczl	master	component: spark-master pod-template-hash: 57759b7dbc	minikube	Running	0	~ 0.00m	281.61Mi	a minute ago	

Replica Sets									
Name	Images	Labels	Pods	Created ↑					
spark-worker-7c9db96ddd	worker	component: spark-worker pod-template-hash: 7c9db96ddd	2 / 2	2 minutes ago					
spark-master-57759b7dbc	master	component: spark-master pod-template-hash: 57759b7dbc	1 / 1	2 minutes ago					

Como podemos observar en la dashboard, están corriendo con éxito las dos *replicas* del worker definidas en el yaml y el master.

Trás desplegar tanto el master como el worker con éxito, el siguiente paso es desplegar el servicio, mediante el siguiente comando.

~ ➤ **kubectI apply -f master-service.yaml**

Después de la ejecución del comando, se añade el nuevo servicio, el cual podemos ver que aparece en la terminal con el comando.

~ ➤ **minikube service list**

Resultando el siguiente output:

minikube service list - terminal

File Edit View Search Terminal Help

> minikube service list

NAMESPACE	NAME	TARGET PORT	URL
default	kubernetes	No node port	
default	spark-master	spark-master/7077	http://192.168.49.2:30077
		spark-master-web/8080	http://192.168.49.2:30080
kube-system	kube-dns	No node port	
kube-system	metrics-server	No node port	
kubernetes-dashboard	dashboard-metrics-scraper	No node port	
kubernetes-dashboard	kubernetes-dashboard	No node port	

Al meternos en la url que indica (<http://192.168.49.2:30080>) para la web, podemos observar que el servicio está corriendo correctamente.

← → ↻ 🔍 http://192.168.49.2:30080/

Parrot OS Hack The Box OSINT Services Vuln DB Privacy and Security Learning Resources YouTube ChatGPT Gmail UAM Outlook Moodle

Spark Master at spark://spark-master:7077

URL: spark://spark-master:7077

Alive Workers: 2

Cores in use: 2 Total, 0 Used

Memory in use: 2.0 GiB Total, 0.0 B Used

Resources in use:

Applications: 0 Running, 0 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory	Resources
worker-202311191539-10.244.0.94-43855	10.244.0.94:43855	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-202311191539-10.244.0.95-35887	10.244.0.95:35887	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Con esto, daríamos por concluido el ejercicio 3.

- Exercise 4: why is this useful

1. Scale up and down the number of workers. Are the changes automatically detected by the Spark cluster?

Para la cuál hemos aumentado el número de replicas de lo workers y hemos aplicado los cambios (con el comando **kubectl apply** especificado previamente) como se muestra en la siguiente imagen.

The screenshot shows the Spark Master web interface at `http://192.168.49.2:30080/`. The interface displays the following information:

- URL: `spark://spark-master:7077`
- Alive Workers: 3
- Cores in use: 3 Total, 0 Used
- Memory in use: 3.0 GiB Total, 0.0 B Used
- Resources in use:
- Applications: 0 Running, 0 Completed
- Drivers: 0 Running, 0 Completed
- Status: ALIVE

The **Workers (3)** section is expanded, showing a table of 3 workers:

Worker Id	Address	State	Cores	Memory
<code>worker-20231119191539-10.244.0.94-43855</code>	<code>10.244.0.94:43855</code>	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)
<code>worker-20231119191539-10.244.0.95-35887</code>	<code>10.244.0.95:35887</code>	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)
<code>worker-20231119201722-10.244.0.96-42313</code>	<code>10.244.0.96:42313</code>	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)

The **Running Applications (0)** and **Completed Applications (0)** sections are also visible, each with a table header.

Podemos ver que cambiando el número de **replicas** a 3, el servicio detecta automáticamente el cambio y añade los workers necesarios para satisfacer el cambio.

```
Material > ! worker.yaml
1  kind: Deployment
2  apiVersion: apps/v1
3  metadata:
4    name: spark-worker
5    labels:
6      component: spark-worker
7  spec:
8    replicas: 3
9    selector:
10     matchLabels:
11       component: spark-worker
12     template:
```

← → 🔍 <http://192.168.49.2:30080/> Parrot OS Hack The Box OSINT Services Vuln DB Privacy and Security Learning Resources YouTube ChatGPT Gmail UAM Outlook Moodle

Spark Master at spark://spark-master:7077

URL: spark://spark-master:7077
Alive Workers: 1
Cores in use: 1 Total, 0 Used
Memory in use: 1024.0 MiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory
worker-20231119191539-10.244.0.94-43855	10.244.0.94:43855	DEAD	1 (0 Used)	1024.0 MiB (0.0 B Used)
worker-20231119191539-10.244.0.95-35887	10.244.0.95:35887	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)
worker-20231119201722-10.244.0.96-42313	10.244.0.96:42313	DEAD	1 (0 Used)	1024.0 MiB (0.0 B Used)

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time
----------------	------	-------	---------------------	------------------------	----------------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time
----------------	------	-------	---------------------	------------------------	----------------

Al igual que al bajar el número de **replicas** también se modifica a tiempo real para satisfacer el número especificado. Matando a dos workers, como se aprecia en la foto de arriba.

```
Material > ! worker.yaml
1  kind: Deployment
2  apiVersion: apps/v1
3  metadata:
4    name: spark-worker
5    labels:
6      component: spark-worker
7  spec:
8    replicas: 1
9    selector:
10     matchLabels:
11       component: spark-worker
12    template:
```

2. Delete the Apache Spark (without deleting minikube).

Para esto lo que habría que hacer es borrar la aplicación de spark:

```
kubectl delete sparkapplication <nombre>
```

Luego borrar todos los recursos que haya abierto:

```
kubectl delete pods
```

```
kubectl delete deployments
```

```
kubectl delete services
```

Y con eso ya estaría, podríamos comprobar con get que no hay recursos activos

```
kubectl get pods
```

```
kubectl get deployments
```

```
kubectl get services
```

```
kubectl get pv (para ver si hay algun valor de memoria persistente)
```

3. Deploy two separate Spark clusters on the same k8s infrastructure. They must be totally independent.

Para crear otro spark cluster paralelo al que teníamos y totalmente independiente se puede realizar con el comando.

```
~ > minikube start -p nombre_cluster
```

En nuestro caso.

```
~ > minikube start -p cluster2
```

Listándolos, con el comando `minikube profile list`, obtenemos lo siguiente:

```
> minikube profile list
```

Profile	VM Driver	Runtime	IP	Port	Version	Status	Nodes	Active
cluster2	docker	docker	192.168.58.2	8443	v1.28.3	Running	1	
minikube	docker	docker	192.168.49.2	8443	v1.28.3	Running	1	*

Viendo los perfiles vemos que **minikube** corresponde al que hemos estado trabajando previamente y **cluster2** al que acabamos de crear.

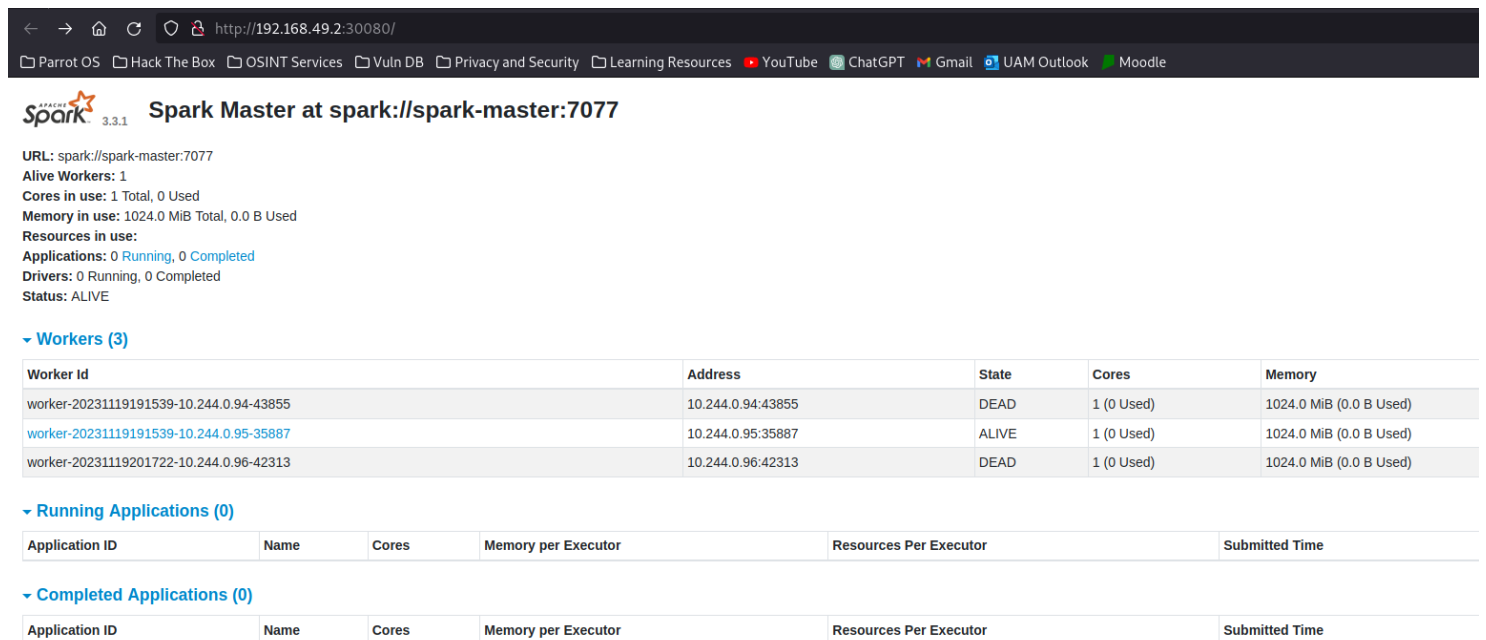
Tras esto, entramos al nuevo perfil **cluster2** (mediante el comando **minikube profile**) y desde ahí podemos desplegar de nuevo los deployments y servicios necesarios para el nuevo cluster de spark. Con los comandos especificados previamente. Como se muestra en la imagen.

```
> kubectl apply -f master.yaml
deployment.apps/spark-master created
> kubectl apply -f worker.yaml
deployment.apps/spark-worker created
> kubectl apply -f master-service.yaml
service/spark-master created
> minikube service list
```

NAMESPACE	NAME	TARGET PORT	URL
default	kubernetes	No node port	
default	spark-master	spark-master/7077	http://192.168.58.2:30077
		spark-master-web/8080	http://192.168.58.2:30080
kube-system	kube-dns	No node port	

Podemos ver que esta vez, ha creado una nueva IP (<http://192.168.58.2:30080>) y al acceder vemos que está totalmente disponible, creada con otro número de workers para poder diferenciarla correctamente de la anterior, en las siguientes imágenes se comparan los dos servicios, uno de cada cluster. Fijarse en la **IP** de cada uno y en el **número de workers**.

Servicio cluster minikube



The screenshot shows the Spark Master UI for a minikube cluster. The URL bar indicates the address is <http://192.168.49.2:30080/>. The page title is "Spark Master at spark://spark-master:7077". The status is "ALIVE". The "Workers" section shows 3 workers, with one in a "DEAD" state and two in an "ALIVE" state. The "Running Applications" section shows 0 applications.

URL: spark://spark-master:7077
Alive Workers: 1
Cores in use: 1 Total, 0 Used
Memory in use: 1024.0 MiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory
worker-20231119191539-10.244.0.94:43855	10.244.0.94:43855	DEAD	1 (0 Used)	1024.0 MiB (0.0 B Used)
worker-20231119191539-10.244.0.95:35887	10.244.0.95:35887	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)
worker-20231119201722-10.244.0.96:42313	10.244.0.96:42313	DEAD	1 (0 Used)	1024.0 MiB (0.0 B Used)

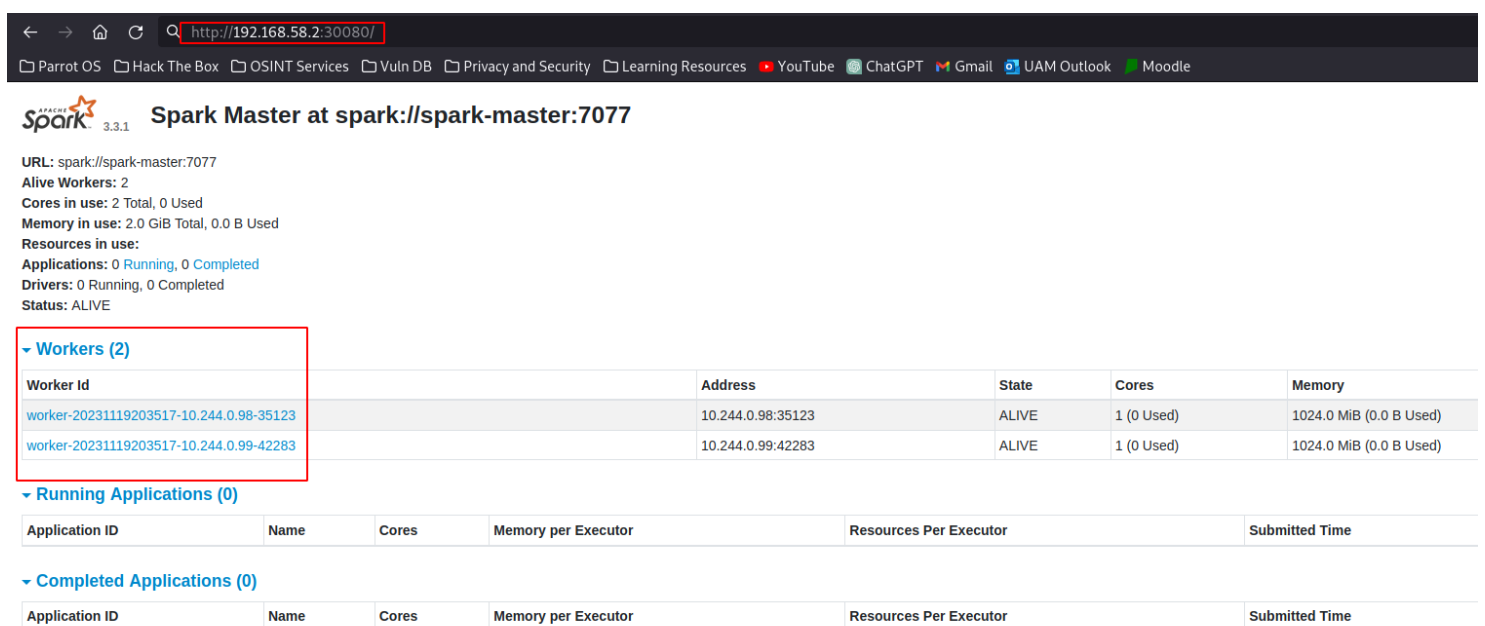
Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time
----------------	------	-------	---------------------	------------------------	----------------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time
----------------	------	-------	---------------------	------------------------	----------------

Servicio cluster cluster2



The screenshot shows the Spark Master UI for a cluster2. The URL bar indicates the address is <http://192.168.58.2:30080/>. The page title is "Spark Master at spark://spark-master:7077". The status is "ALIVE". The "Workers" section shows 2 workers, both in an "ALIVE" state. The "Running Applications" section shows 0 applications.

URL: spark://spark-master:7077
Alive Workers: 2
Cores in use: 2 Total, 0 Used
Memory in use: 2.0 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (2)

Worker Id	Address	State	Cores	Memory
worker-20231119203517-10.244.0.98:35123	10.244.0.98:35123	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)
worker-20231119203517-10.244.0.99:42283	10.244.0.99:42283	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time
----------------	------	-------	---------------------	------------------------	----------------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time
----------------	------	-------	---------------------	------------------------	----------------

