# Memoria Practica 3

Participante: Pablo Sanchez Fernandez del Pozo

Participante: Carlos Riveira Ramos

Grupo: 146 Pareja: 3

# Parte 1

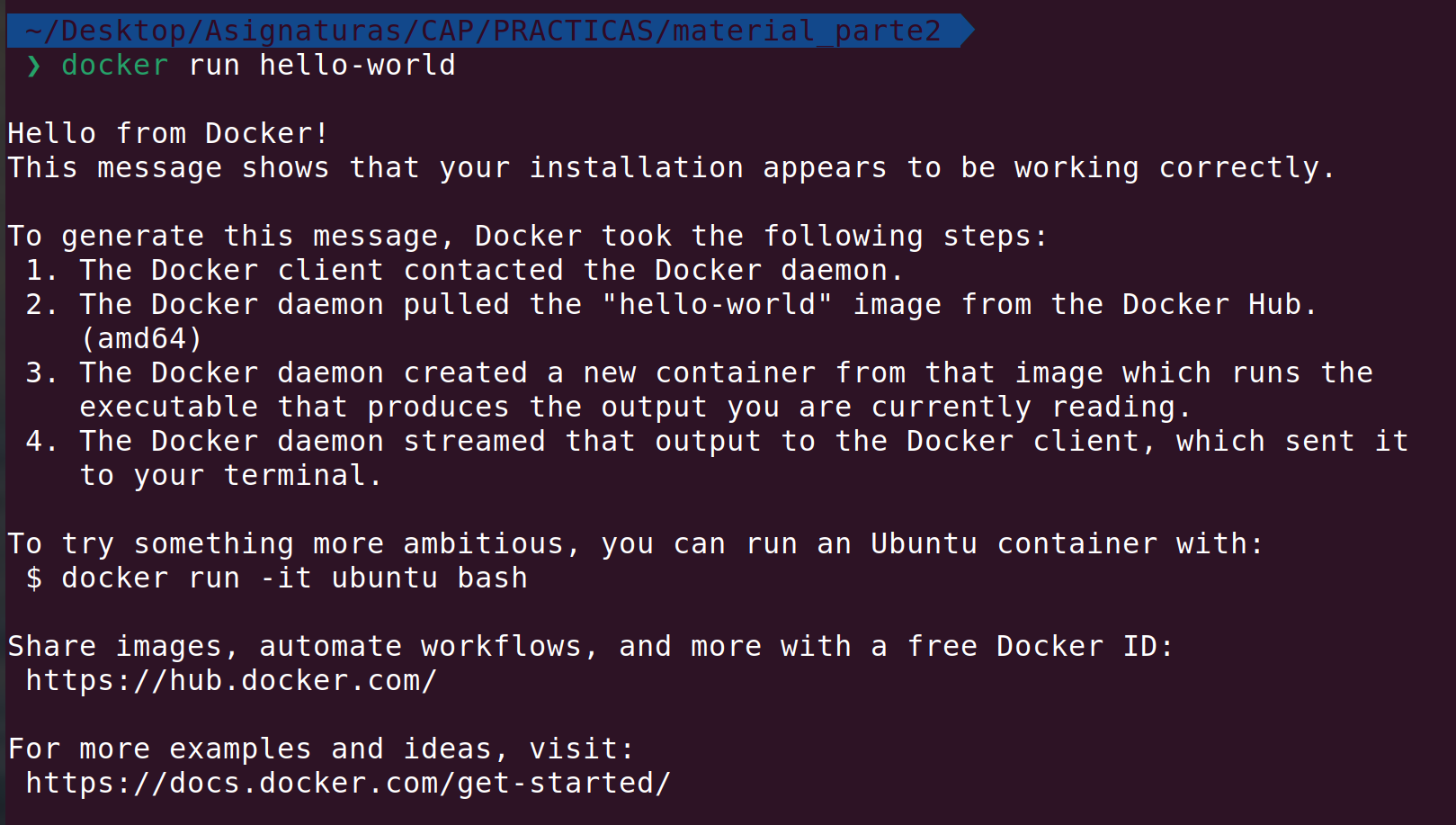
La parte 1 está implementada al completo dentro del notebook de Google Collab. 👍

# Parte 2

Seguimos el tutorial de instalación. Para ello, primeramente requerimos de un container o

virtual machine manager. En nuestro caso instalamos docker.

Para comprobar, podemos ejecutar ‘docker run hello-world’:



Una vez tenemos docker instalado, instalamos de minikube y arrancamos el servicio con:

*minikube start*

Agregamos al usuario a el grupo de docker:

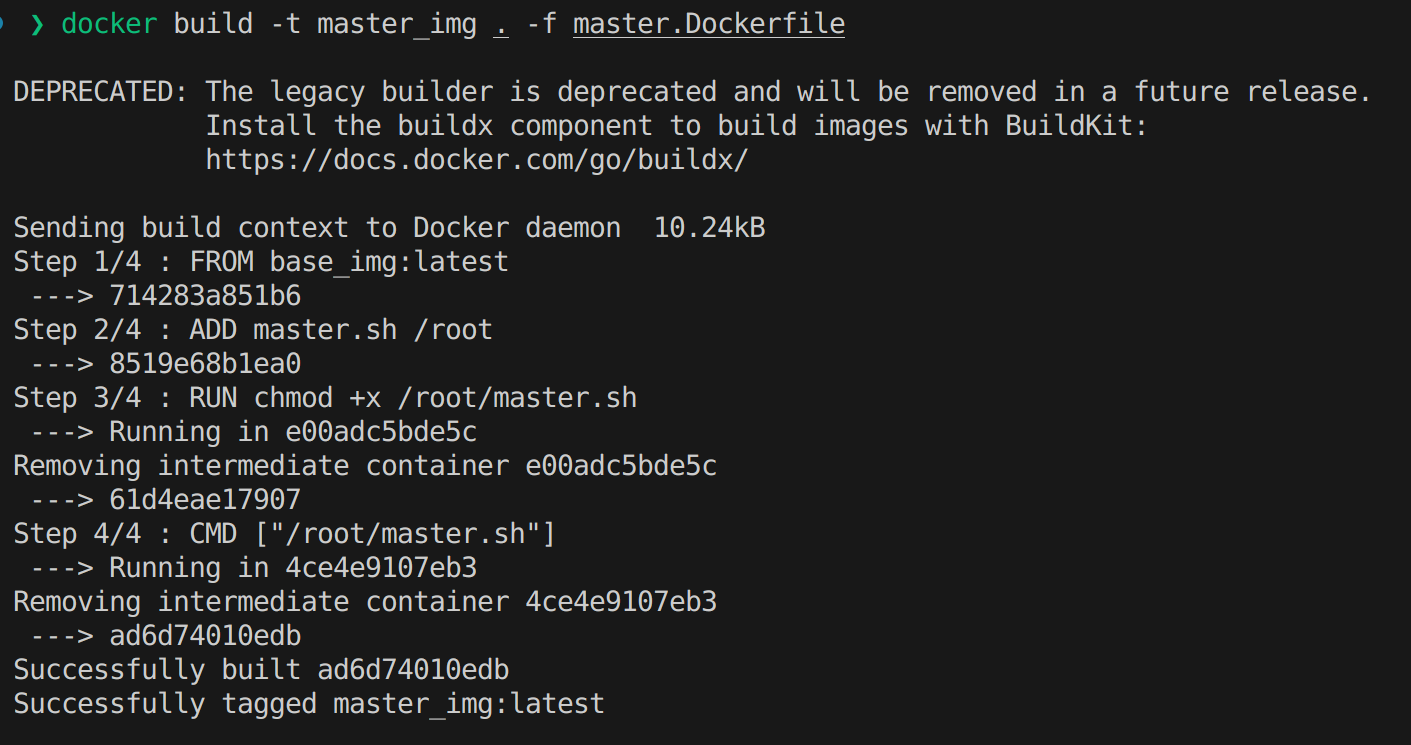
*sudo usermod -aG docker $USER && newgrp docker*

Generamos todas las imagenes:

*docker build -t base\_img . -f base.Dockerfile*

*docker build -t master\_img . -f master.Dockerfile*

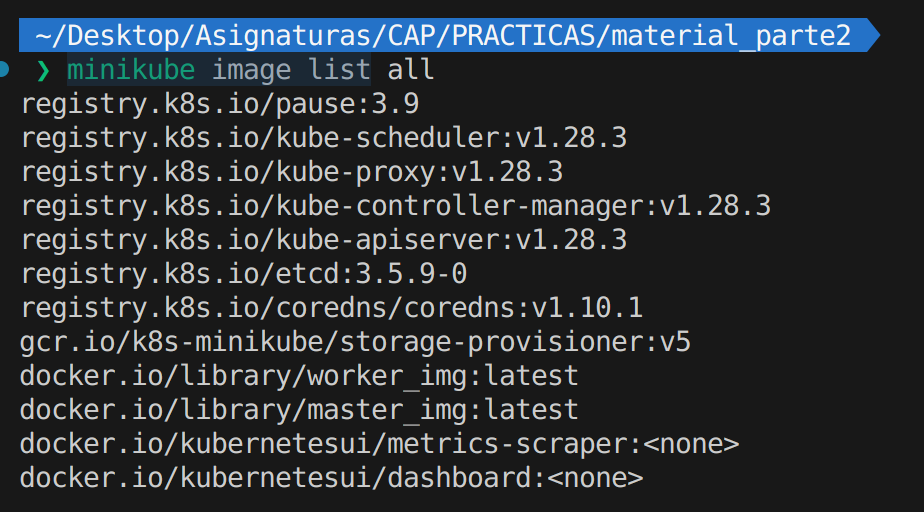
*docker build -t worker\_img . -f worker.Dockerfile*



Cargamos las imagenes de docker a minikube:

*minikube image load master\_img*

*minikube image load worker\_img*

**

Creamos un namespace para el despliegue, se llamará ***spark***:

*kubectl create namespace spark*

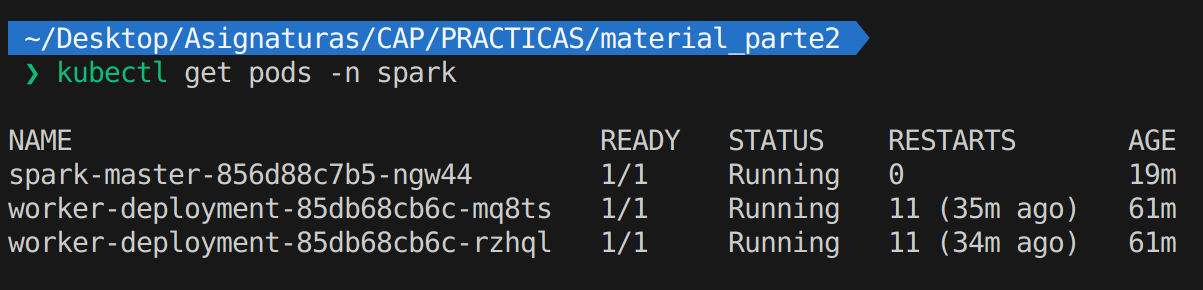
Y creamos las instancias de master, worker y el master-sevice:

*kubectl apply -f master-deployment.yaml -n spark*

*kubectl apply -f worker-deployment.yaml -n spark*

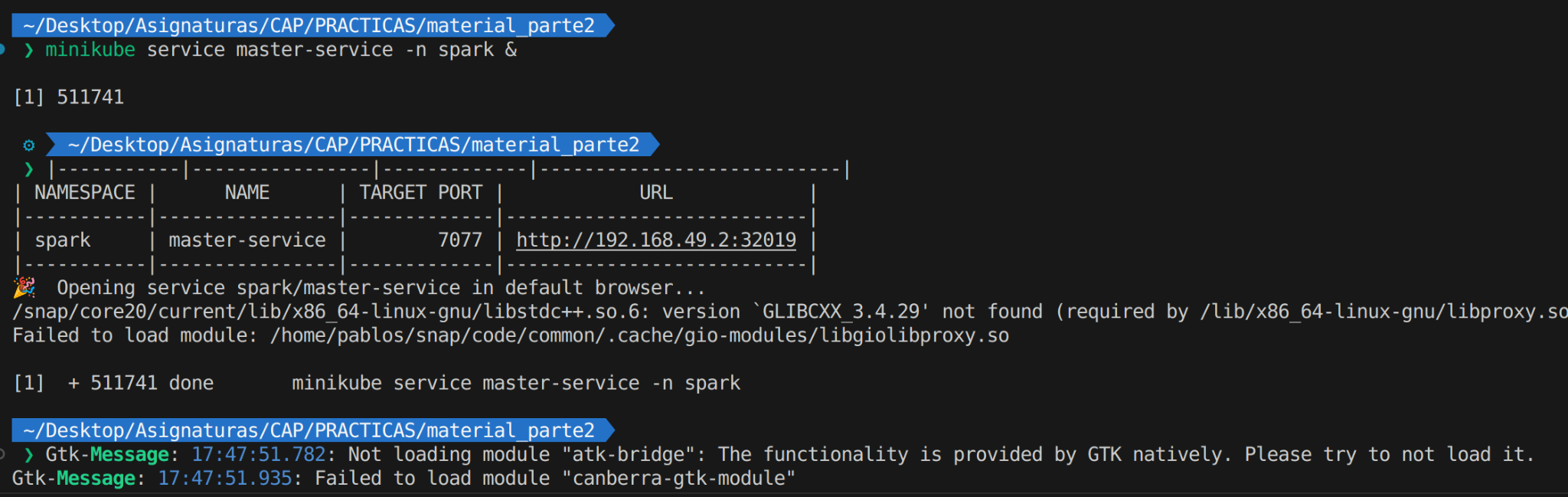
*kubectl apply -f master-service.yaml -n spark*

*kubectl describe pods -n spark*

**

Exponemos el servicio con:

*minikube service master-service -n spark &*

**

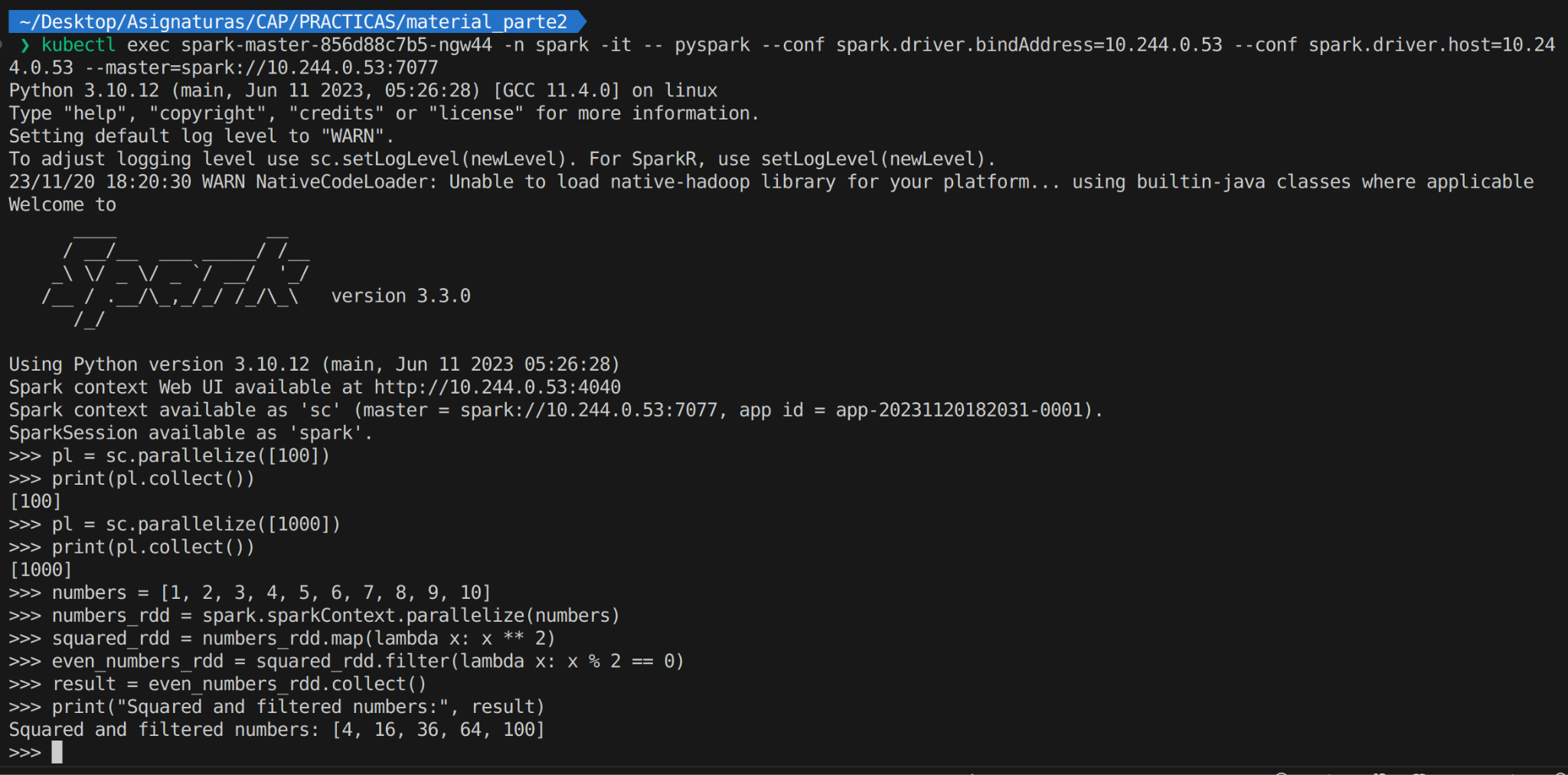
Obtenemos información de nuestro *spark-master*:

*kubectl describe pod -n spark*

Lanzamos la consola de **pyspark:**

*kubectl exec spark-master-856d88c7b5-ngw44 -n spark -it -- pyspark --conf spark.driver.bindAddress=10.244.0.53 --conf spark.driver.host=10.244.0.53 --master=spark://10.244.0.53:7077*

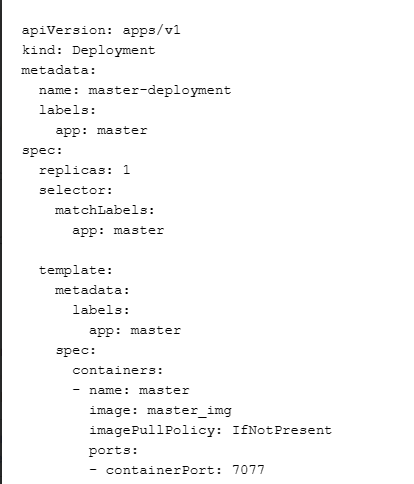
Para comprobar el buen funcionamiento de los trabajadores, en nuestro ejemplo paralelizamos dos transformaciones: elevamos al cuadrado una lista de números y luego filtramos y nos quedamos con los pares:



## Exercise 3: make everything work

Ficheros YAML:

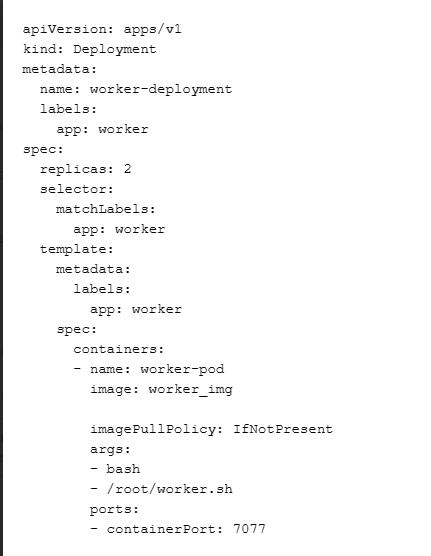
master-deployment.yaml:



Este fichero define un recurso de Kubernete de tipo Deployment para desplegar un nodo maestro, con 1 sola replica, en Apache Spark. Cada una de las etiquetas define una cosa distinta. Paso a su descripción:

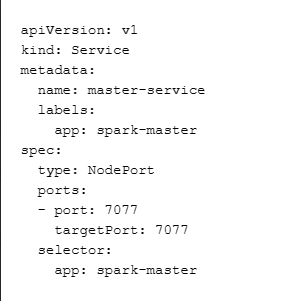
* apiVersion: especifica la versión de Kubernetes utilizada.
* kind: especifica el tipo de recurso.
* metadata: contiene la información de metadatos para el Deployment
* name: especifica el nombre del deployment.
* labels: define etiquetas para el deployment.
* spec: describe la especificación del deployment.
* replicas: establece el número deseado de instancias. En este caso un pod maestro.
* selector: especifica un selector de etiquetas para los pods controlados por el deployment.
* matchLabels: define el conjunto de etiquetas que deben coincidir para que un pod pueda ser controlado por el deployment.
* template: define la plantilla del pod para el deployment.
* containers: especifica los contenedores dentro del pod
* image: especifica la imagen del docker para el contenedor.
* imagePullPolicy: establece la política para la descarga de la imagen del contenedor. En nuestro caso, significa que la imagen solo se descarga si no está presente.
* ports: especifica los puertos del contenedor.

worker-deployment.yaml:



Este archivo define un recurso Kubernetes de tipo Deployment, que despliega nodos trabajadores, dos réplicas, de Apache Spark. Utiliza una imagen de Docker y un script de Bash al iniciar.

master-service.yaml:



Este archivo, crea un servicio, “master-service” para el nodo maestro de Spark. Mediante el atributo “NodePort”, se permite el acceso desde fuera del cluster a través de un puerto en cada nodo del mismo. Este servicio, dirige el tráfico a los pods con la etiqueta app: spark-master.

## Exercise 4: why this is useful

**Provide instructions for the following:**

**1. Scale up and down the number of workers. Are the changes automatically detected by the Spark cluster? To check it, you may need to expose port 8080 of the master.**

Para poder aumentar o disminuir la cantidad de “workers”. Bastaría con añadir dentro del archivo “worker-deployment.yaml” el atributo “replicas:”. Otra forma de hacerlo sería desde el campo “instances” en el servicio Spark.

Si se utiliza la primera opción, el controlador de despliegue monitoriza automáticamente el estado del conjunto de pods y compara la cantidad actual de pods con el valor del campo “replicas”. Crea o elimina pods para poder ajustarse al mismo.

En caso de querer utilizar la opción del servicio de Spark, existe una opción de detección automática para ajustar el número de forma dinámica dependiendo de la carga de trabajo. Para hacerlo, hay que activar la dicha asignación: spark.dynamicAllocation.enabled.

En caso de realizarse de forma manual y alterar a mano el valor del campo “instances” se debe recargar el servicio para que los cambios surjan efecto.

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

Para realizar esta tarea, debemos:

1. Eliminar la aplicación Spark Mediante el comando:

kubectl delete sparkapplication <nombre\_spark>

1. Eliminar el operador de Spark:

kubectl delete -f https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/releases/download/v<version>/crds/sparkoperator.k8s.io\_sparkapplications\_crd.yaml

1. Eliminar las configuraciones de Spark:

kubectl delete configmap <nombre\_configmap>

kubectl delete secret <nombre\_secret>

1. Comprobar que no quedan pods desplegados:

kubectl get pods -l spark-role=master

kubectl get pods -l spark-role=worker

En cuyo caso, los eliminamos manualmente:

kubectl delete pod <nombre\_pod>

Siguiendo estos pasos, no debería verse afectada la instalación de minikube.

**3. Deploy two separate Spark clusters on the same k8s infrastructure. They must be totally independent. What changes should be done to the YAML files.**

Con el objetivo de poder tener dos clusters de Spark separados. Debemos:

* Crear un nuevo namespace, por ejemplo:

*kubectl create namespace spark2*

En este nuevo namespace podemos usar los mismos .yaml que usamos en el otro namespace sin importarnos la colisión entre puertos, ya que cada namespace se le asigna una IP diferente, funcionan en este caso como clusters independientes pero con la misma infraestructura.

Si queremos automatizar la creación de los pods en cada namespace, podemos añadir este campo a cada copia de los services:

***…***

***metadata:***

***name: spark-master***

***namespace: spark2***

***…***

Aquí dejamos algunos kuber-memes!!! 😀



