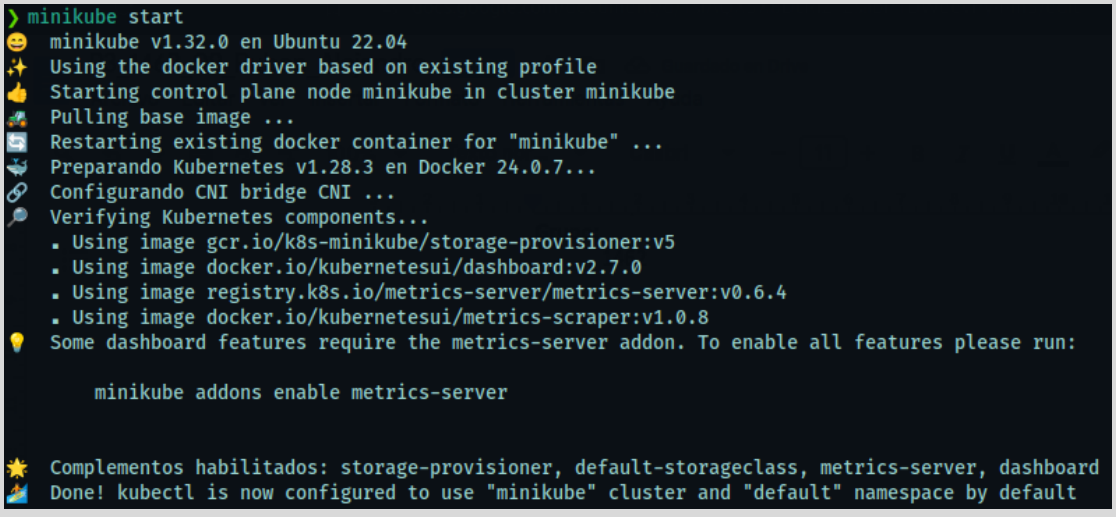
ASIGNATURA Computación de altas prestaciones

Tarea 3
Programación de GPU

Autores:
Rubén Fernández Alimán
Alejandro Monterrubio Navarro
Grupo:
1462

Ejercicio 1

Siguiendo el tutorial instalamos lo necesario para esta práctica. Ejecutamos minikube con docker:



Imágen 1. minikube start

Ejercicio 2

Para esto tenemos que crear los 3 dockerfiles. En este caso para las imágenes de master y worker hemos dado permisos al .sh correspondiente.

```
FROM ubuntu:22.04
# 1. Install dependencies
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y default-jdk default-jre curl

# 2. define spark and hadoop versions
```

```
ENV SPARK_VERSION=3.3.0
ENV HADOOP_VERSION=3.3.4

# 3. Download and extract spark
RUN mkdir -p /opt && \
  cd /opt && \
  curl http://archive.apache.org/dist/spark/spark-${SPARK_VERSION}/spark-${SPARK_VERSION}-bin-hadoop3.tgz | \
    tar -zx && \
  ln -s spark-${SPARK_VERSION}-bin-hadoop3 spark

# 4. Download and extract hadoop
RUN mkdir -p /opt && \
  cd /opt && \
  curl http://archive.apache.org/dist/hadoop/common/hadoop-${HADOOP_VERSION}/hadoop-${HADOOP_VERSION}.tar.gz | \
    tar -zx hadoop-${HADOOP_VERSION}/lib/native && \
  ln -s hadoop-${HADOOP_VERSION} hadoop

# 5. Add spark and hadoop to PATH
ENV PATH $PATH:/opt/spark/bin
```

Código 1. base.dockerfile

```
FROM base:latest

ADD master.sh /root

# chmod +x master.h
RUN chmod +x /root/master.sh
CMD ["/root/master.sh"]
```

Código 2. master.dockerfile

```
FROM base:latest

ADD worker.sh /root

# chmod +x master.h
RUN chmod +x /root/worker.sh
CMD ["/root/worker.sh"]
```

Código 3. worker.dockerfile

Y construimos las imágenes, para esto nosotros hemos ejecutado los siguientes dos comandos:

- docker build -t nombre . -f nombredocker.Dockerfile
- minikube image load nombre

Por ejemplo para la imágen de base:

```
❯ docker build -t base . -f base.Dockerfile
[+] Building 189.4s (8/8) FINISHED                                                                 docker:default
 => [internal] load .dockerignore                                                                            0.0s
 => => transferring context: 2B                                                                              0.0s
 => [internal] load build definition from base.Dockerfile                                                    0.0s
 => => transferring dockerfile: 890B                                                                         0.0s
 => [internal] load metadata for docker.io/library/ubuntu:22.04                                              1.2s
 => [1/4] FROM docker.io/library/ubuntu:22.04@sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f   3.4s
 => => resolve docker.io/library/ubuntu:22.04@sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f   0.0s
 => => sha256:2b7412e6465c3c7fc5bb21d3e6f1917c167358449fecac8176c6e496e5c1f05f 1.13kB / 1.13kB               0.0s
 => => sha256:c9cf959fd83770dfdefd8fb42cfef0761432af36a764c077aed54bbc5bb25368 424B / 424B                   0.0s
 => => sha256:e4c58958181a5925816faa528ce959e487632f4cfd192f8132f71b32df2744b4 2.30kB / 2.30kB               0.0s
 => => sha256:aece8493d3972efa43bfd4ee3cdba659c0f787f8f59c82fb3e48c87cbb22a12e 29.54MB / 29.54MB             1.6s
 => => extracting sha256:aece8493d3972efa43bfd4ee3cdba659c0f787f8f59c82fb3e48c87cbb22a12e                    1.5s
 => [2/4] RUN apt-get update && apt-get install -y default-jdk default-jre curl                             52.6s
 => [3/4] RUN mkdir -p /opt &&     cd /opt &&     curl http://archive.apache.org/dist/spark/spark-3.3.0/spark-3.3.0-bin-hadoop3.tgz |         tar -zx &&     ln -  36.2s
 => [4/4] RUN mkdir -p /opt &&     cd /opt &&     curl http://archive.apache.org/dist/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz |         tar -zx hadoop-3.3  92.4s
 => exporting to image                                                                                       3.5s
 => => exporting layers                                                                                      3.5s
 => => writing image sha256:7ebea26da22d83bdc2e5877e7c2f0d4fe550425950962ebd0e9f81e3ad652663                 0.0s
 => => naming to docker.io/library/base                                                                      0.0s
```

Imágen 2. build de la imagen base

```
❯ minikube image load base
❯ minikube image load master
❯ minikube image load worker
```

Imágen 3. cargar las imágenes en minikube

Ejercicio 3

Una vez las imágenes están cargadas necesitamos crear los ficheros yaml necesarios para obtener los dos contenedores, para esto necesitamos un master.yaml un worker.yaml y un service.yaml que permita exponer el cluster al host.

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: spark-master
  namespace: spark
spec:
  replicas: 1
  selector:
    matchLabels:
      component: spark-master
  template:
    metadata:
      labels:
        component: spark-master
    spec:
      containers:
        - name: spark-master
          image: master
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 7070
```

Código 4. master.yaml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: spark-worker
  namespace: spark
spec:
  replicas: 2
  selector:
    matchLabels:
      component: spark-worker
  template:
    metadata:
      labels:
        component: spark-worker
    spec:
      containers:
```

```
      - name: spark-worker
        image: worker
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 7077
```

Código 5. worker.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: spark-master
  namespace: spark
spec:
  type: NodePort
  ports:
  - port: 7077
    targetPort: 7077
    nodePort: 30077
  selector:
    app: AppSpark
```

Código 6. service.yaml

Ahora una vez creados los yaml, tenemos que crear el namespace de spark y aplicar estos yaml.

```
❯ kubectl create namespace spark
namespace/spark created
❯ kubectl apply -f master.yaml
deployment.apps/spark-master created
❯ kubectl apply -f service.yaml
service/spark-master created
❯ kubectl apply -f worker.yaml
deployment.apps/spark-worker created
```

Imágen 6. Crear namespace y aplicar yaml

Y comprobamos que están creados estos contenedores mediante el siguiente comando:

```
❯ kubectl get pods -n spark -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP
spark-master-9c8f984d8-dfg7h    1/1     Running   0          36s   10.244.0.154
spark-worker-59c948dcb5-264ms   1/1     Running   0          21s   10.244.0.156
spark-worker-59c948dcb5-m5nd9   1/1     Running   0          21s   10.244.0.155
```

Imágen 7. Pods junto con su ip, en correcto funcionamiento

Para ver si el service se ha creado correctamente ejecutamos el siguiente comando:

```
❯ kubectl get service -n spark
NAME           TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)
spark-master   NodePort   10.110.191.207   <none>        7077:30077/TCP
```

Imágen 8. servicio en el puerto solicitado

Ejercicio 4

- Scale up and down the number of workers. Are the changes automatically detected by the Spark cluster?
  Para realizar esto basta con ejecutar el siguiente comando:

```
❯ kubectl get pods -n spark -o wide
NAME                            READY   STATUS    RESTARTS   AGE     IP             NO
spark-master-6b99b549b6-9pxxs   1/1     Running   0          9m9s    10.244.0.124   mi
spark-worker-7c757bbb8c-s4cjn   1/1     Running   0          8m55s   10.244.0.125   mi
spark-worker-7c757bbb8c-vxnbm   1/1     Running   0          8m55s   10.244.0.126   mi
❯ kubectl scale --replicas=4 deployment spark-worker -n spark
deployment.apps/spark-worker scaled
❯ kubectl get pods -n spark -o wide
NAME                            READY   STATUS    RESTARTS   AGE     IP             NO
spark-master-6b99b549b6-9pxxs   1/1     Running   0          9m44s   10.244.0.124   mi
spark-worker-7c757bbb8c-7m7pz   1/1     Running   0          2s      10.244.0.127   mi
spark-worker-7c757bbb8c-b7gkf   1/1     Running   0          2s      10.244.0.128   mi
spark-worker-7c757bbb8c-s4cjn   1/1     Running   0          9m30s   10.244.0.125   mi
spark-worker-7c757bbb8c-vxnbm   1/1     Running   0          9m30s   10.244.0.126   mi
❯ kubectl scale --replicas=2 deployment spark-worker -n spark
deployment.apps/spark-worker scaled
❯ kubectl get pods -n spark -o wide
NAME                            READY   STATUS        RESTARTS   AGE     IP
spark-master-6b99b549b6-9pxxs   1/1     Running       0          9m58s   10.244.0.124
spark-worker-7c757bbb8c-7m7pz   1/1     Terminating   0          16s     10.244.0.127
spark-worker-7c757bbb8c-b7gkf   1/1     Terminating   0          16s     10.244.0.128
spark-worker-7c757bbb8c-s4cjn   1/1     Running       0          9m44s   10.244.0.125
spark-worker-7c757bbb8c-vxnbm   1/1     Running       0          9m44s   10.244.0.126
```

Imágen 9. Escalado de réplicas de los worker

Como se observa ampliamos las réplicas a 4 y los cambios son detectados automáticamente, creándose 2 réplicas más totalmente funcionales. Después volvemos a escalar pero ahora a 2 réplicas otra vez, y vemos como el status de dos workers se establece a “Terminating” para en los segundos posteriores ser eliminados.

- Delete the Apache Spark (without deleting minikube).

Para esto basta con ejecutar el siguiente comando:

```
❯ kubectl delete all --all -n spark
pod "spark-master-6b99b549b6-9pxxs" deleted
pod "spark-worker-7c757bbb8c-s4cjn" deleted
pod "spark-worker-7c757bbb8c-vxnbm" deleted
service "spark-master" deleted
deployment.apps "spark-master" deleted
deployment.apps "spark-worker" deleted
❯ kubectl get pods -n spark
No resources found in spark namespace.
```

Imágen 10. Eliminación de todos los contenedores dentro de spark

También basta con ejecutar el comando “kubectl delete namespace spark”.

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

Para realizar esto basta con crear un nuevo namespace, luego modificar los yaml cambiando el namespace y los puertos del servicio NodePort.