Entrenament d'un Model amb Docker i Kubernetes

0. Crear github repository

Repositori: https://github.com/NIU1638962/GIXPD_Practica_2

```
#sudo -i
#mkdir /home/adminp/GIXPD_Practica_2
#setfacl -R -m u:adminp:rwx /home/adminp/GIXPD_Practica_2
#logout
#cd GIXPD_Practica_2
#echo “# GIXPD_Practica_2” >> README.md
#git init
#git config --global --add safe.directory /home/adminp/GIXPD_Practica_2
#git config --global user.name “Joel Tapia Salvador (1638963)”
#git config --global user.email 91463596+NIU1638962@users.noreply.github.com
#git config --global pull.rebase false
#git config --global pull.ff only
#git config --global core.editor “gedit”
#git add README.md
#git commit -m “first commit”
#git branch -M main
#git remote add origin https://github.com/NIU1638962/GIXPD_Practica_2
#git push -u origin main
```

1. Configuració i Instal·lació de Dependències

En primer lloc, necessitarem instal·lar les dependències.

1. Docker

```
#sudo -i
#apt-get update
#apt-get install ca-certificates curl gnupg
#install -m 0755 -d /etc/apt/keyrings
#curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
#chmod a+r /etc/apt/keyrings/docker.gpg
#echo \
  "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
#sudo apt-get update
#apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

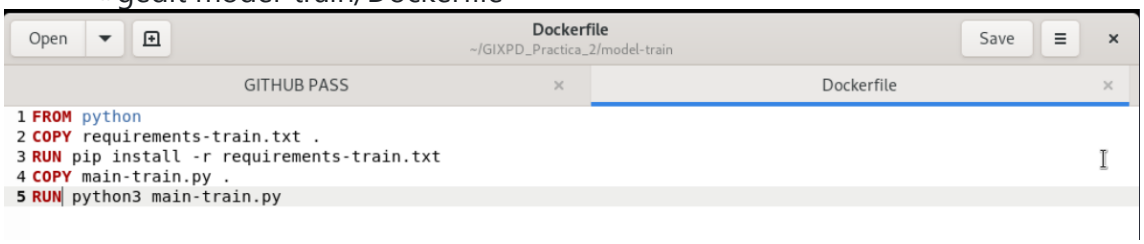
#docker run hello-world
#groupadd docker
#logout
#usermod -aG docker $USER
#newgrp docker
#newgrp docker

2. Kubectl
curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256
#echo "$(cat kubectl.sha256) kubectl" | sha256sum –check
#sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
#kubectl version --client --output=yaml
#sudo apt-get install bash-completion
#_init_completion
#kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null
#sudo chmod a+r /etc/bash_completion.d/kubectl

3.Minikube
#curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64.deb
#sudo dpkg -i minikube_latest_amd64.deb

2. Crear Aplicació i Imatges Docker

Per a aquest exercici, necessitarem crear 2 imatges Docker.
1. Una aplicació de feina o "script" que bàsicament entreni un model i el desa en undisc. La imatge hauria d'estar etiquetada com **model-train:default**.
#mkdir model-train
eval $(minikube docker-env)
#mv main-train.py model-train/
#mv requirements-train.txt model-train/
#> model-train/Dockerfile
#gedit model-train/Dockerfile



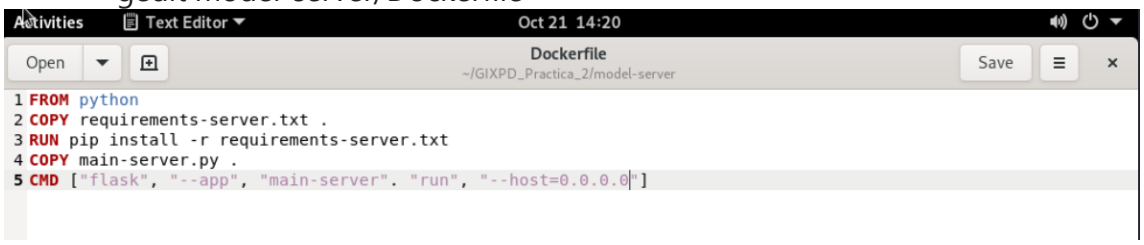
#docker build model-train --tag model-train:default
#docker run \

```
--volume /tmp:/model \
--env MODEL_PATH=/model/model_1.npy \
model-train:deafult
```

```
adminp@debian:~/GIXPD_Practica_2$ docker run --volume /tmp:/model --env MODEL_PA
TH=/model/model_1.npy model-train:default
Model trained successfully
Model Score: 0.8086921460343566
```

2. Una aplicació de servei amb Flask que farà el següent. A **/** carregarà una pàgina HTML senzilla que expliqui com funciona el servei. A **/model** carregarà el model i rebrà els paràmetres del model per a retornarla sortida del model com a JSON. La imatge hauria d'estar etiquetada com **model-server:default**.

```
#mkdir model-server
#mv maint-server.py model-server/
#mv requirements-server.txt model-server/
#> model-server/Dockerfile
#gedit model-server/Dockerfile
```



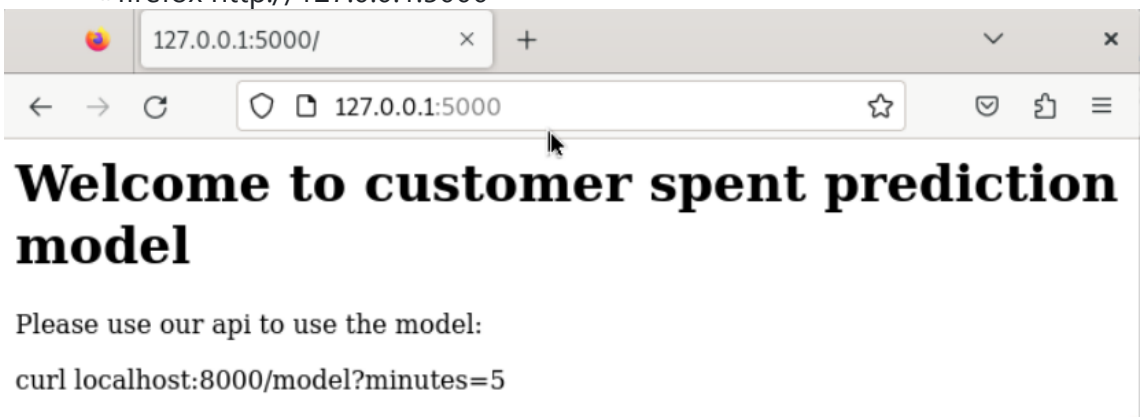

```
#docker build model-server --tag model-server:default
#docker run \
        --volume /tmp:/model \
        --env MODEL_PATH=/model/model_1.npy \
        model-server:deafult
```

```
adminp@debian:~/GIXPD_Practica_2$ docker run --volume /tmp:/model --env MODEL_PA
TH=/model/model_1.npy model-server:default
 * Serving Flask app 'main-server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

```
#docker run \
        --volume /tmp:/model \
        --env MODEL_PATH=/model/model_1.npy \
        --publish 5000:5000 \
        model-server:deafult
```

```
^Cadminp@debian:~/GIXPD_Practica_2$ docker run --volume /tmp:/model --env MODEL_
PATH=/model/model_1.npy --publish 5000:5000 model-server:default
 * Serving Flask app 'main-server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

#firefox http://127.0.0.1:5000



#curl localhost:5000/model?minutes=5

```
adminp@debian:~/GIXPD_Practica_2$ curl localhost:5000/model?minutes=5
{"spent":22.8796259181172}
```

3. Desplegar l'Aplicació a Kubernetes

1. Feina: Entrenarà el model i el desarà en un volum.

Necessita tenir una variable d'entorn MODEL_PATH en la qual s'emmagatzemarà el model. Cal que tingui sol·licituds i límits de recursos definits.

#> Job.yaml

#gedit Job.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: model-train
  namespace: default
spec:
  template:
    spec:
      containers:
        - name: model-train
          image: model-train:default
          imagePullPolicy: IfNotPresent
          env:
            - name: MODEL_PATH
              valueFrom:
                configMapKeyRef:
                  name: model-path
                  key: MODEL_PATH
          volumeMounts:
            - mountPath: /model
              name: model-mount
          resources:
            requests:
              memory: 64Mi
              cpu: 200m
            limits:
              memory: 100Mi
              cpu: 500m
      volumes:
        - name: model-mount
          hostPath:
            path: /tmp
            type: Directory
      restartPolicy: OnFailure
```

2. Desplegament: Aplicació que servirà el model entrenat. Necessita tenir 3 rèpliques. Necessita tenir una variable d'entorn MODEL_PATH des de la qual es llegirà el model. Cal que tingui sol·licituds i límits de recursos definits. Cal que tingui probes de salut i preparación.

#> Deployment.yaml

#gedit Deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: model-server
  namespace: default
spec:
  selector:
    matchLabels:
      environment: "dev"
  replicas: 3
  template:
    metadata:
      labels:
        environment: "dev"
    spec:
      containers:
      - name: model-server
        image: model-server:default
        imagePullPolicy: IfNotPresent
        env:
          - name: MODEL_PATH
            valueFrom:
              configMapKeyRef:
                name: model-path
                key: MODEL_PATH
        volumeMounts:
          - mountPath: /model
            name: model-mount
        resources:
          requests:
            memory: 64Mi
            cpu: 200m
          limits:
            memory: 100Mi
            cpu: 500m
```


```yaml
          - name: MODEL_PATH
            valueFrom:
              configMapKeyRef:
                name: model-path
                key: MODEL_PATH
        volumeMounts:
          - mountPath: /model
            name: model-mount
        resources:
          requests:
            memory: 64Mi
            cpu: 200m
          limits:
            memory: 100Mi
            cpu: 500m
```


```yaml
        ports:
          - containerPort: 5000
        livenessProbe:
          httpGet:
            path: /healthz
            port: 5000
          initialDelaySeconds: 5
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /readiness
            port: 5000
          initialDelaySeconds: 10
          periodSeconds: 5
      volumes:
        - name: model-mount
          hostPath:
            path: /tmp
            type: Directory
```

3. Servei: Es crearà per servir les 3 rèpliques diferents del desplegament.

#> Service.yaml

#gedit Service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: model-server
  namespace: default
spec:
  selector:
    enviroment: "dev"
  ports:
  - name: http-service
    appProtocol: http
    protocol: TCP
    port: 5000
    targetPort: 5000
  type: ClusterIP
```

4. ConfigMap: Emmagatzemarà un valor tant per a la Feina com per al Desplegament persaber on desar o llegir el model.

#> ConfigMap.yaml

#gedit ConfigMap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: model-path
data:
  MODEL_PATH: "/model/model_1.npy"
```

#minikube image load model-train:default

#minikube image load model-server:default

#minikube starta

#kubectl apply -f ConfigMap.yaml

#kubectl apply -f Job.yaml

#kubectl apply -f Deployment.yaml

#kubectl apply -f Server.yaml

Comandes útils per veure Pods de Kubernetes.

kubectl get pods

#kubectl logs <pod_name>

#kubectl describe pod <pod_name>