



PRÀCTICA 2

GESTIÓ D'INFRASTRUCTURES PER AL
PROCESSAMENT DE DADES

Pol Termes 1671849 / Guille Martínez 1673113

ÍNDEX

Objectiu	2
Sessió 1	2
Instal·lació de Docker	2
Postinstal·lació de Docker	3
Instal·lació de Kubectl	3
Instal·lació Kubectl Autocompletion	4
Creació 1a Imatge	4
Sessió 2	5
Creació 2a Imatge	5
Carregar imatges.....	7
Crear el ConfigMap de Kubernetes	8
Crear el Job de Kubernetes	9
Sessió 3	10
Crear el Desplegament de Kubernetes	10
Crear el Servei de Kubernetes.....	11
Model2	12
Problemes i errors.....	13

Objectiu

L'objectiu d'aquesta pràctica és entrenar un model de dades i exposar-lo a través d'una API utilitzant Flask. Aquesta implementació inicial permet fer prediccions i gestionar el model de manera senzilla. Posteriorment, es pretén replicar aquest procés en un entorn Kubernetes, amb l'objectiu de desplegar l'aplicació de manera més escalable i robusta. L'ús de Kubernetes permet gestionar múltiples instàncies del model, assegurant una millor disponibilitat i facilitant el seu manteniment en producció.

Es durà a terme a aquest repositori Github: https://github.com/Pallofa04/GIXPD_Practica2

Sessió 1

Instal·lació de Docker

Per tal d'instal·lar Docker s'ha seguit els passos de la web de dockerdocs: <https://docs.docker.com/engine/install/ubuntu/>

On s'ha hagut de realitzar les següents comandes:

Add Docker's official GPG key:

```
sudo apt-get update
```

```
sudo apt-get install ca-certificates curl
```

```
sudo install -m 0755 -d /etc/apt/keyrings
```

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
```

```
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Add the repository to Apt sources:

```
echo \
```

```
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]  
https://download.docker.com/linux/ubuntu \
```

```
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
```

```
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt-get Update
```

Seguidament per finalitzar la instal·lació s'han instal·lat els paquets de Docker:

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Finalment per verificar la correcta instal·lació es carrega l'imatge *hello-world*:

sudo docker run hello-world

En el nostra cas, no s'ha afegit el *sudo* ja que ja s'està en mode administrador (*sudo -i*).

```
root@adminp:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c19c31eb5944: Pull complete
Digest: sha256:91fb4b041da273d5a3273b6d587d62d518300a6ad268b28628f74997b93171b2
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Postinstal·lació de Docker

Docker per motius de seguretat corre el seu daemon amb permisos de root. Per tal no haver d'entra en mode administrador o escriu *sudo* davant de cada comanda s'agrega el nostre usuari a un grup anomenat docker. Quan l'usuari pertany a aquest grup ja se li otorguen permisos per poder intereutar amb el demon de docker sense necessitat d'utilitzar *sudo*.

Primer es crea el grup docker i després s'afegeix l'usuari al grup. Perquè es guardin els canvis es reinicia la màquina.

sudo groupadd docker

sudo usermod -aG docker \$USER

Per comprovar-ho es carrega l'imatge *hello-world* sense *sudo*.

docker run hello-world

```
adminp@adminp:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

Instal·lació de Kubectl

Per instal·lar Kubectl, primerament s'ha d'instal·lar el seu binari, en l'arquitectura x86-64 es realitza la següent comanda:

curl -LO [https://dl.k8s.io/release/\\${curl -L -s https://dl.k8s.io/release/stable.txt}/bin/linux/amd64/kubectl](https://dl.k8s.io/release/${curl -L -s https://dl.k8s.io/release/stable.txt}/bin/linux/amd64/kubectl)

Per validar el binari de Kubectl, es descarrega el fitxer checksum corresponent i es comprova el binari amb l'arxiu checksum per assegurar-ne la integritat:

```
curl -LO https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubect1.sha256
echo "$(cat kubect1.sha256) kubect1" | sha256sum --check
```

```
adminp@adminp:~$ echo "$(cat kubect1.sha256) kubect1" | sha256sum --check
kubect1: OK
```

Finalment s'instal·la Kubectl i es comprova si la versió està al dia:

```
sudo install -o root -g root -m 0755 kubect1 /usr/local/bin/kubect
```

```
kubect1 version --client
```

```
adminp@adminp:~$ sudo install -o root -g root -m 0755 kubect1 /usr/local/bin/kubect1
adminp@adminp:~$ kubect1 version --client
Client Version: v1.31.1
Kustomize Version: v5.4.2
```

Instal·lació Kubectl Autocompletion

Per facilitar l'ús de comandes a la terminal mitjançant la suggerència automàtica de comandes s'instal·la l'autocompletació, el nostre cas serà per Bash.

Primer de tot s'ha d'instal·lar mitjançant:

```
sudo apt-get install bash-completion
```

Seguidament s'ha d'habilitar i per a què l'script de bash-completion s'executi automàticament cada vegada que s'obri una terminal:

```
echo 'source <(kubect1 completion bash)' >> ~/.bashrc
```

I finalment es recarrega la terminal perquè es guardin els canvis.

```
source ~/.bashrc
```

Creació 1a Imatge

La primera imatge Docker és bàsicament un script el qual entreni un model i el desi en un disc. Haurà de ser anomenat *model-train:default*.

```

GNU nano 7.2 Dockerfile *
# Dockerfile: Use an official Python runtime as a parent image
FROM python:3.6-slim
# Set the working directory to /app
WORKDIR /app
# Copy the current directory contents into the container at /app
ADD . /app
# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements-train.txt
# Define environment variable
ENV MODEL_PATH=/app/model.npy
# Run main-train.py when the container launches
CMD ["python", "main-train.py"]

```

Construir la imatge:

docker build -f Dockerfile.train -t model-train:default .

```

adminp@adminp: ~/Practica2/GIXPD_Practica2/Model$ docker build -f Dockerfile.train -t model-train:default .
[+] Building 14.4s (9/9) FINISHED
=> [internal] load build definition from Dockerfile.train                                docker:default
=> [internal] load build definition from Dockerfile.train                                0.0s
=> => transferring dockerfile: 529B                                                    0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim                      0.9s
=> [internal] load .dockerignore                                                         0.0s
=> => transferring context: 2B                                                         0.0s

```

Executar la imatge i entrenar model:

docker run model-train:default

```

adminp@adminp: ~/Practica2/GIXPD_Practica2/Train$ docker run model-train:default
Model trained successfully
Model Score: 0.8086921460343659

```

Sessió 2

Creació 2a Imatge

Dockerfile:

```

# Dockerfile: Use an official Python runtime as a parent image
FROM python:3.9-slim
# Set the working directory to /app
WORKDIR /app
# Copy the current directory contents into the container at /app
ADD . /app
# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements-server.txt
# Define environment variable
ENV MODEL_PATH=/app/model.npy
ENV FLASK_APP=main-server.py
# Run main-server.py when the container launches
CMD ["flask", "--app", "main-server.py", "run", "--host=0.0.0.0"]

```

Construir la imatge:

docker build -f Dockerfile.server -t model-server:default .

```

adminp@adminp: ~/Practica2/GIXPD_Practica2/Model$ docker build -f Dockerfile.server -t model-server:default .
[+] Building 13.3s (9/9) FINISHED
=> [internal] load build definition from Dockerfile.server                                docker:default
=> [internal] load build definition from Dockerfile.server                                0.0s
=> => transferring dockerfile: 564B                                                    0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim                      0.8s
=> [internal] load .dockerignore                                                         0.0s
=> => transferring context: 2B                                                         0.0s

```

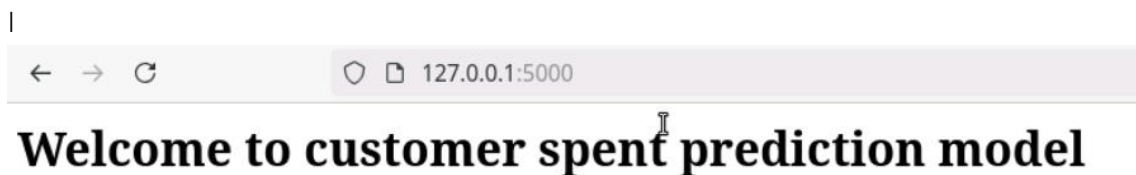
Executar:

```
docker run -e MODEL_PATH=/app/model.npy -v  
/home/adminp/Practica2/GIXPD_Practica2/Model:/app -p 5000:5000 model-server:default
```

- **-e MODEL_PATH=/app/model.npy:** Defineix la variable d'entorn MODEL_PATH que apunta al fitxer que s'emmagatzemarà dins del contenidor.
- **-f Dockerfile.server:** Indica el nom del Dockerfile que es vol utilitzar.
- **-v \$(pwd):/app:** Muntatge del directori app del teu sistema host al directori /app dins del contenidor. Això permet que qualsevol fitxer creat o modificat a /app sigui visible a app en el teu sistema host.
- **-p 5000:5000:** Exposar el port 5000 del contenidor al port 5000 del teu host, la qual cosa permet accedir a l'aplicació Flask des del navegador.

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Model$ docker run -e MODEL_PATH=/app/model.npy -v /home/adminp/Practica2/GIXPD_Practica2/Model:/app -p 5000:5000 model-server:default
* Serving Flask app 'main-server.py'
* 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
```

Si es va a l'adreça web <http://127.0.0.1:5000> es pot visualitzar:



Please use our api to use the model:

```
curl localhost:8000/model?minutes=5
```

Es veu l'accés a la web:

```
* Running on http://172.17.0.2:5000
Press CTRL+C to quit
172.17.0.1 - - [20/Oct/2024 11:15:06] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [20/Oct/2024 11:15:06] "GET /favicon.ico HTTP/1.1" 404 -
```

A la web es mostra com utilitzar la API on es farà una petició a la ruta /model on el paràmetre minutes es defineix com a 5. El servidor Flask agafarà aquest valor i el passarà al model de predicció. En el nostre cas, seria aquesta línia de comandes ja que s'ha utilitzat el port 5000.

```
curl localhost:5000/model?minutes=5
```

Mentre el contenidor està en execució i el servidor Flask iniciat, a una altra terminal s'executa aquesta línia de comandes.

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Server$ curl localhost:5000/model?minutes=5
<!doctype html>
<html lang=en>
<title>500 Internal Server Error</title>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.</p>
```

```
docker run -e MODEL_PATH=/app/model.npy -v  
/home/adminp/Practica2/GIXPD_Practica2/Model:/app -p 5000:5000 model-server:default
```

I es torna a executar el contenidor `model-server:default` i el `curl`. Com que s'ha realitzat correctament ens surt el següent missatge:

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Model$ curl localhost:5000/model?minutes=5  
{"spent":22.879625918124134}
```

Carregar imatges

Primer de tot s'ha d'instal·lar el paquet *Minikube*.

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64  
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64
```

I realitzem *minikube start* per arrencar el paquet.

Primer, s'han de guardar les imatges en arxius `.tar`:

```
docker save -o model-train.tar model-train:default  
docker save -o model-server.tar model-server:default
```

Seguidament les carreguem:

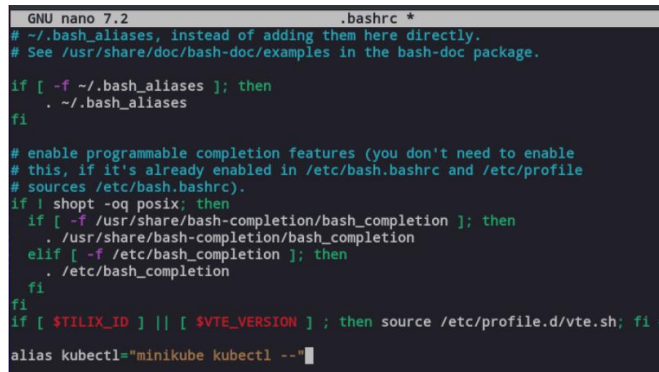
```
minikube image load model-train.tar  
minikube image load model-server.tar
```

Per veure les imatges carregades:

```
minikube image ls
```

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Model$ minikube image ls  
registry.k8s.io/pause:3.10  
registry.k8s.io/kube-scheduler:v1.31.0  
registry.k8s.io/kube-proxy:v1.31.0  
registry.k8s.io/kube-controller-manager:v1.31.0  
registry.k8s.io/kube-apiserver:v1.31.0  
registry.k8s.io/etcd:3.5.15-0  
registry.k8s.io/coredns/coredns:v1.11.1  
gcr.io/k8s-minikube/storage-provisioner:v5  
docker.io/library/model-train:default  
docker.io/library/model-server:default
```


Abans de tot s'afegeix *alias kubectl com minikube kubectl* – a l'arxiu `.bashrc`. Es realitza per no haver d'escriure *minikube kubectl* – cada vegada.



```

GNU nano 7.2 .bashrc *
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
  . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
if [ $TILIX_ID ] || [ $VTE_VERSION ]; then source /etc/profile.d/vte.sh; fi #
alias kubectl="minikube kubectl --"

```

I s'apliquen els canvis amb `source .bashrc`.

Un cop instal·lat *Minikube*, per tal que l'entorn de *Kubernetes* funcioni correctament, cal configurar diversos recursos essencials:

- **ConfigMap:** Emmagatzema informació de configuració que poden utilitzar els *pods*, com ara variables d'entorn o camins d'accés.
- **Job:** Permet executar tasques puntuals o treballs que han de completar-se un cop. En el nostre cas entrenar un model.
- **Deployment:** S'encarrega de gestionar la creació i l'escalat de rèpliques d'un *pod*, assegurant que sempre hi hagi un nombre desitjat de còpies en execució.
- **Service:** Proporciona una manera estable de comunicar-se amb els *pods*, assignant-los una adreça IP fixa i gestionant el balanç de càrrega.

Aquests recursos han de ser configurats per assegurar el correcte funcionament de l'aplicació a l'entorn *Kubernetes*. Després de ser configurats s'han de aplicar amb `kubectl apply -f <nom.yaml>`.

Crear el ConfigMap de Kubernetes

Aquest *ConfigMap* de *Kubernetes* es diu *my-config* i emmagatzema una variable de configuració anomenada `MODEL_PATH`. Aquesta variable apunta a la ruta `/app/model.npy`, que és la ubicació on es troba el model que l'aplicació utilitza.

El *ConfigMap* permet que aquesta configuració es defineixi i s'utilitzi dins dels contenidors del clúster de *Kubernetes* sense haver de codificar la ruta directament en l'aplicació. Això facilita la gestió de la configuració i permet canviar la ruta del model fàcilment si cal.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  MODEL_PATH: "/app/model.npy"
```

Crear el Job de Kubernetes

Aquest *Job* de *Kubernetes*, anomenat *model-train-job*, llança un contenidor que entrena un model utilitzant la imatge *model-train:default*. Executa el fitxer *main-train.py* amb les variables d'entorn definides al *ConfigMap* *my-config*. El contenidor té límits de recursos de CPU i memòria i accedeix a un volum muntat a */app* per emmagatzemar el model entrenat. El *Job* es reintenta fins a 4 vegades si falla i només s'executa una vegada.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: model-train-job
spec:
  template:
    metadata:
      name: model-train-job-pod
    spec:
      containers:
        - name: model-train
          image: model-train:default
          command: ["python"]
          args: ["main-train.py"]
          envFrom:
            - configMapRef:
                name: my-config
          resources:
            limits:
              cpu: "500m"
              memory: "128Mi"
            requests:
              cpu: "250m"
              memory: "64Mi"
          volumeMounts:
            - name: model-data
              mountPath: "/app"
      volumes:
        - name: model-data
          hostPath:
            path: "/tmp"
      restartPolicy: Never
      backoffLimit: 4
```

Sessió 3

Crear el Desplegament de Kubernetes

Aquest *Deployment* de *Kubernetes*, anomenat *model-server-deployment*, crea tres rèpliques del contenidor *model-server* usant la imatge *model-server:default*, exposant el port 5000. Configura la variable d'entorn *MODEL_PATH* amb el valor definit al *ConfigMap* *my-config*. Es defineixen límits

de CPU i memòria, i el volum `model-data` es munta a `/app` per emmagatzemar dades. També inclou probes de liveness i readiness per monitoritzar l'estat del servidor.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: model-server-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: model-server
  template:
    metadata:
      labels:
        app: model-server
    spec:
      containers:
        - name: model-server
          image: model-server:default
          ports:
            - containerPort: 5000
          env:
            - name: MODEL_PATH
              valueFrom:
                configMapKeyRef:
                  name: my-config
                  key: MODEL_PATH
          resources:
            requests:
              memory: "64Mi"
              cpu: "250m"
            limits:
              memory: "128Mi"
              cpu: "500m"
          volumeMounts:
```

```
    volumeMounts:
      - name: model-data
        mountPath: "/app"
    livenessProbe:
      httpGet:
        path: /healthz
        port: 80
      initialDelaySeconds: 5
      periodSeconds: 10
    readinessProbe:
      httpGet:
        path: /readiness
        port: 80
      initialDelaySeconds: 10
      periodSeconds: 5
    volumes:
      - name: model-data
        hostPath:
          path: "/tmp"
```

Crear el Servei de Kubernetes

Aquest *Service* de *Kubernetes*, anomenat *model-server-service*, exposa el *Deployment* *model-server* al clúster. Selecciona els contenidors amb l'etiqueta *app: model-server* i redirigeix el trànsit al port 5000 dels contenidors. Al ser de tipus *NodePort*, permet accedir al servei des de fora del clúster a través de qualsevol node, facilitant la connexió amb el servidor de model.

```
apiVersion: v1
kind: Service
metadata:
  name: model-server-service
spec:
  selector:
    app: model-server
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 5000
  type: NodePort
```

Model2

```
import os
import numpy as np
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeRegressor

# Ruta del model
MODEL_PATH = os.environ["MODEL_PATH"]

# Dades de mostra
np.random.seed(2)
x = np.random.normal(3, 1, 100).reshape(-1, 1)
y = np.random.normal(150, 40, 100) / x.ravel()

# Divisió en conjunts d'entrenament i test
train_x = x[:80]
train_y = y[:80]

test_x = x[80:]
test_y = y[80:]

# Creació i entrenament del model amb un arbre de decisió regressiu
model = DecisionTreeRegressor(max_depth=4)
model.fit(train_x, train_y)
print("Model entrenat amb èxit")

# Avaluació del model
r2 = r2_score(test_y, model.predict(test_x))
print("Model Score:", r2)

# Guardar el model
np.save(MODEL_PATH, model)
```

Aquest codi és per crear i avaluar un model de regressió amb un arbre de decisió utilitzant dades generades de manera aleatòria. La gran diferència d'aquest model amb l'anterior és que intenta aprendre una relació entre les dades mitjançant un arbre de decisió mentre que l'anterior ho feia servir una regressió polinòmica.

S'ha copiat tots els arxius i s'han enganxat a un nou directori anomenat Model2. S'ha canviar tots els noms afegint 2 i els seus respectius paràmetres dins dels arxius.

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Model2$ ls
app2                Dockerfile.server2  main-server2.py      model-server-deployment2.yaml  model-train-job2.yaml  requirements-server2.txt
configmap2.yaml     Dockerfile.train2   main-train2.py       model-server-service2.yaml     __pycache__            requirements-train2.txt
```

A l'hora de crear i executar les imatges també s'han modificat els paràmetres per tal de que les línies de comandes s'executessin de manera desitjada.

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Model2$ docker build -f Dockerfile.train2 -t model-train2:default .
[+] Building 30.0s (9/9) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile.train2      1.3s
=> => transferring dockerfile: 534B                             0.2s
=> [internal] load metadata for docker.io/library/python:3.9-slim 1.7s
=> [internal] load .dockerignore                               0.0s
=> => transferring context: 2B                                    0.0s
=> CACHED [1/4] FROM docker.io/library/python:3.9-slim@sha256:7a9cd42706c174cdc578880ab9ae3b6551323a7ddbc2a89ad6e3b20a28fbfbc 0.0s
=> [internal] load build context                                0.1s
=> => transferring context: 5.90kB                               0.0s
=> [2/4] WORKDIR /app2                                         0.2s
=> [3/4] ADD . /app2                                           0.0s
=> [4/4] RUN pip install --trusted-host pypi.python.org -r requirements-train2.txt 28.6s
=> exporting to image                                          3.0s
=> => exporting layers                                          3.0s
=> => writing image sha256:c9a857c47da50a611c3ca0c60d6aca47577fa629f2ac7f4a301c0f44187e716a 0.0s
=> => naming to docker.io/library/model-train2:default          0.0s
adminp@adminp:~/Practica2/GIXPD_Practica2/Model2$ docker run model-train2:default
Model entrenat amb èxit
Model Score: 0.7895650468933675
```

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Model2$ docker build -f Dockerfile.server -t model-server:default .
[+] Building 13.4s (9/9) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile.server      0.1s
=> => transferring dockerfile: 593B                             0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim 0.5s
=> [internal] load .dockerignore                               0.0s
=> => transferring context: 2B                                    0.0s
=> [1/4] FROM docker.io/library/python:3.9-slim@sha256:7a9cd42706c174cdc578880ab9ae3b6551323a7ddbc 0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 374B                                  0.0s
=> CACHED [2/4] WORKDIR /app                                    0.0s
=> CACHED [3/4] ADD . /app                                      0.0s
=> [4/4] RUN pip install --trusted-host pypi.python.org -r requirements-server.txt 11.7s
=> exporting to image                                          0.9s
=> => exporting layers                                          0.9s
=> => writing image sha256:089e8445c4f62a9c564549ab8f3eab5d3860c02508c12f6bb26612bcd66c3f48 0.0s
=> => naming to docker.io/library/model-server:default          0.0s
adminp@adminp:~/Practica2/GIXPD_Practica2/Model2$ docker run -v /home/adminp/Practica2/GIXPD_Practica2/Model2:/app -p 5000:5000 model-server:default
* Serving Flask app 'main-server.py'
* 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
```

S'ha intentat dur a terme l'entorn kubernetes però sorgeix el mateix problema. A causa de que no podem carregar les imatges no es pot verificar si les configuracions són correctes.

Problemes i errors

En aquesta apartat, s'explicaran els problemes i errors destacables sorgits durant la pràctica. Pel que fa la sessió 1 no es va tenir cap problema rellevant mentre que la sessió 2 i 3 sí que hi van haver.

Durant la sessió 2 es van presentar 3 problemes, 2 relacionats amb l'execució de l'imatge *model-server:default* i 1 relacionat amb l'estructura dels directoris.

El primer va ser causat per una configuració al *Dockerfile*. Al *Dockerfile* se l'indicava un entorn de python el qual no era compatible amb Flask, era massa antic (Python3.6-slim) i per tant no es reconeixia el paràmetre `--app`. Es va substituir per Python3.9-slim.

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Server$ docker run -e MODEL_PATH=/app/model.npy -v /home/adminp/Practica2/GIXPD_Practica2/Server/app:/app -p 5000:5000 model-server:default
Usage: flask [OPTIONS] COMMAND [ARGS]...
Try 'flask --help' for help.

Error: no such option: --app
```

Una vegada corregit l'anterior error, ens vam adonar que l'adreça de muntatge era incorrecta ja que s'estava afegint `/app` a l'adreça del host. Un cop canviat ja es va executar correctament.

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Server$ docker run -e MODEL_PATH=/app/model.npy -v /home/adminp/Practica2/GIXPD_Practica2/Server/app:/app -p 5000:5000 model-server:default
Usage: flask run [OPTIONS]
Try 'flask run --help' for help.

Error: Could not import 'main-server'.
```

El tercer problema que va requerir més temps de resolució. Estava relacionat amb la ubicació del fitxer del model, *model.npy*, generat amb *model-train:default*. Inicialment, s'havia configurat el servidor i l'entrenament del model en *Dockerfiles* independents, cadascun amb el seu directori de treball `/app`, però aquests directoris no estaven enllaçades. Quan es va provar d'executar *model-server:default*, aquest intentava accedir a *model.npy*, però com que aquest es trobava en el directori `/app` de *model-train:default*, no era accessible des de l'entorn del servidor.

```
[2024-10-20 12:13:22,253] ERROR in app: Exception on /model [GET]
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/site-packages/flask/app.py", line 1473, in wsgi_app
    response = self.full_dispatch_request()
  File "/usr/local/lib/python3.9/site-packages/flask/app.py", line 882, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/usr/local/lib/python3.9/site-packages/flask/app.py", line 880, in full_dispatch_request
    rv = self.dispatch_request()
  File "/usr/local/lib/python3.9/site-packages/flask/app.py", line 865, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args) # type: ignore[no-any-return]
  File "/app/main-server.py", line 24, in model
    model = np.polyid(np.load(MODEL_PATH))
  File "/usr/local/lib/python3.9/site-packages/numpy/lib/_npymath_impl.py", line 455, in load
    fid = stack.enter_context(open(os.fspath(file), "rb"))
FileNotFoundError: [Errno 2] No such file or directory: '/app/model.npy'
172.17.0.1 - - [20/Oct/2024 12:13:22] "GET /model?minutes=5 HTTP/1.1" 500 -
```

Per resoldre aquest problema, es va establir una configuració de volum compartit que permetés que el directori on es guarda el model entrenat fos accessible tant per a *model-server:default* com per a *model-train:default*, assegurant així que el fitxer *model.npy* estigués disponible en l'entorn del servidor per a fer prediccions. Aquesta solució va requerir una revisió de la distribució de directoris i volums per assegurar la compartició correcta de dades entre els contenidors. De tenir 2 directoris, *Train* i *Server* amb els seus respectius `/app`, es va passar a tenir un únic directori anomenat *Model* amb un únic `/app` i els *dockerfiles* *Dockerfile.train* i

Dockerfile.server. Degut a aquest canvi, es va haver de tornar a executar *model-train:default* tot indicant el dockerfile desitjat amb el paràmetre *-f*.

Finalment a la sessió 3 es va trobar un problema amb el repositori Github i la carrèga de les imatges.

Durant la sessió, es va trobar un problema en fer *git push* al repositori de GitHub després de desar imatges en format *.tar*, ja que superaven el límit de mida de pujada. Per solucionar-ho, es va crear un fitxer *.gitignore* amb la línia **.tar* per evitar que Git rastregi aquests arxius. També es va esborrar la caché dels *.tar* amb *git rm --cached *.tar* per prevenir futurs conflictes, i així el *git push* es va completar amb èxit.

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Model$ nano .gitignore
adminp@adminp:~/Practica2/GIXPD_Practica2/Model$ git rm --cached *.tar
fatal: pathspec 'model-server.tar' did not match any files
adminp@adminp:~/Practica2/GIXPD_Practica2/Model$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  .gitignore

nothing added to commit but untracked files present (use "git add" to track)
adminp@adminp:~/Practica2/GIXPD_Practica2/Model$ git add .gitignore
adminp@adminp:~/Practica2/GIXPD_Practica2/Model$ git commit -m "Ignorar arxius .tar"
```

El problema amb les imatges no es va resoldre. Les imatges carregades en format *.tar* no contenien l'arxiu *model.npy*, essencial per al model. Per solucionar-ho, es va intentar carregar les imatges sense comprimir, però va fallar amb un error de *killed*, possiblement per falta de recursos. Així, malgrat les configuracions fetes a l'entorn de *Kubernetes*, no es va poder comprovar el correcte funcionament del model.

```
adminp@adminp:~/Practica2/GIXPD_Practica2/Model$ minikube image load model-train:default
Killed
adminp@adminp:~/Practica2/GIXPD_Practica2/Model$ minikube image load model-train:default
Killed
```