

# MLOPS assignment

Diabetes application

Dominic Ho 3MCT  
13-12-2022

## Inhoud

Chosen dataset.....	2
AI Model.....	2
About .....	2
FastAPI .....	3
Environment.....	3
Schema.....	4
Routes .....	5
Database with Docker .....	6
Connect database.....	7
Database tables .....	8
Repository and queries .....	9
Docker deployment.....	10
Github Container Registry .....	11
Push image to Github .....	11
Kubernetes.....	12
Create deployment, add service, port-forward and scale it .....	12
Secret .....	14

## Chosen dataset

<https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset>

## AI Model

I will keep it simple and use a Logistic regression classifier, with hyperparameter tuning I will have around 77 – 82 accuracy.

## About

My fictional company is making an application that speeds up diabetes diagnoses.

A normal blood sugar/glucose test will take a few hours (taking the blood sample and comparing changes after different tests)

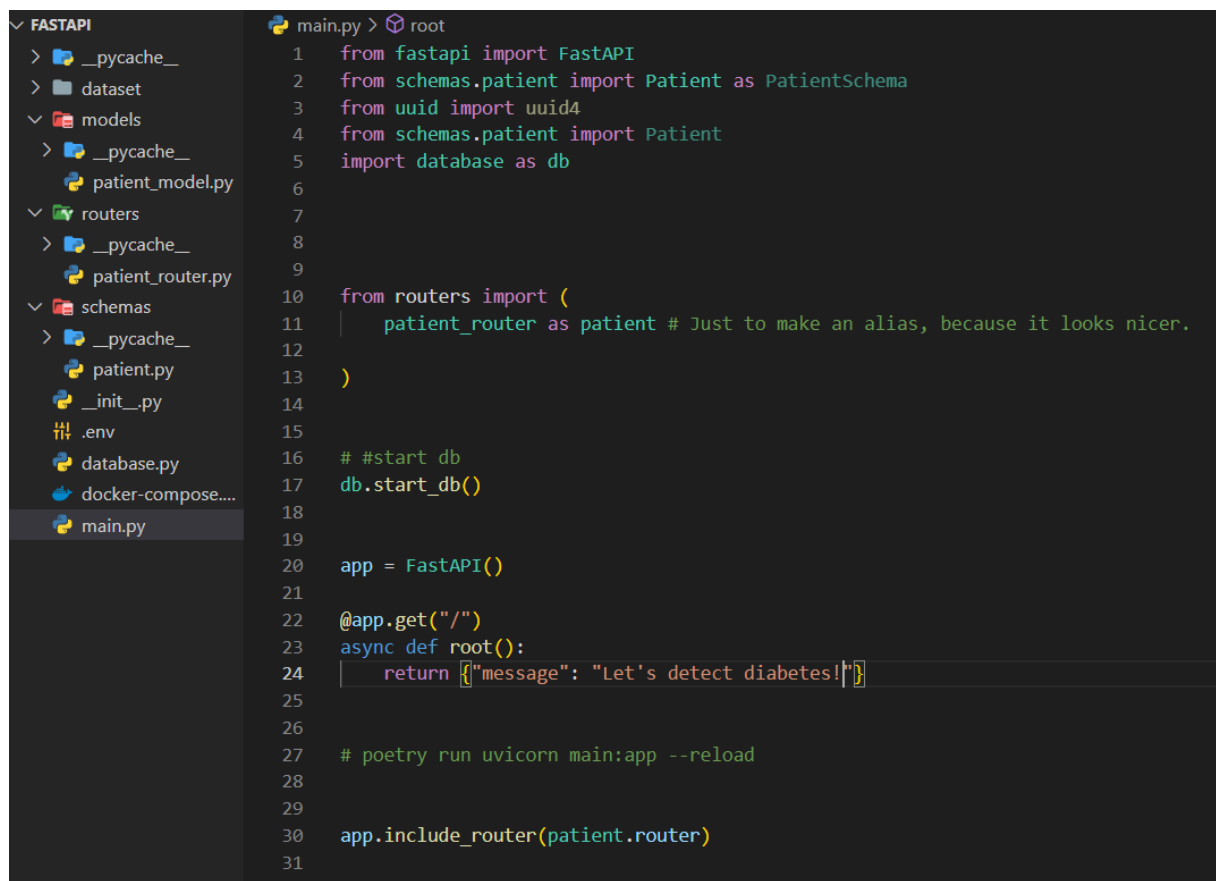
After every test, a blood sample must be sent to the lab which then can take a few days to get the test results back. We want to speed up diagnosis in the lab with our AI model.

(if possible we could also skip the lab part with a mobile application that nurses or doctors can use directly, but I don't know what happens with the samples in the lab. I assume the blood gets analyzed with (expensive) machines )

# FastAPI

## Environment

I made a new Fastapi **environment** with Poetry



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure for 'FASTAPI' with folders like '\_\_pycache\_\_', 'dataset', 'models', 'routers', and 'schemas'. The 'main.py' file is selected. The code editor shows the following Python code:

```
main.py > root
1  from fastapi import FastAPI
2  from schemas.patient import Patient as PatientSchema
3  from uuid import uuid4
4  from schemas.patient import Patient
5  import database as db
6
7
8
9
10 from routers import (
11     |   patient_router as patient # Just to make an alias, because it looks nicer.
12 )
13
14
15
16 # #start db
17 db.start_db()
18
19
20 app = FastAPI()
21
22 @app.get("/")
23 async def root():
24     return [{"message": "Let's detect diabetes!"}]
25
26
27 # poetry run uvicorn main:app --reload
28
29
30 app.include_router(patient.router)
31
32
```

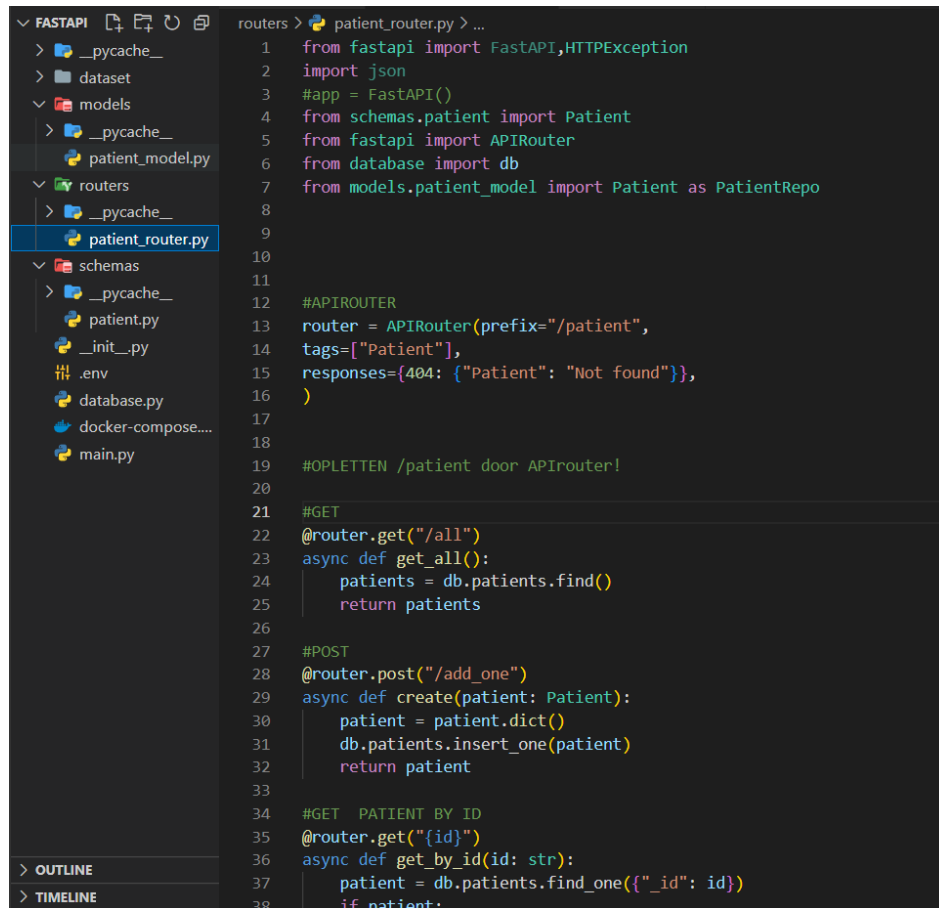
## Schema

Added a schema **Patient**

```
schemas > patient.py > Patient > name
1  # patient.py
2  from typing import Optional
3  from pydantic import BaseModel
4
5
6  class Patient(BaseModel):
7      # uuid: Optional[str]
8      id: str
9      name: str
10     pregnancies: int
11     glucose: int
12     bloodpressure: int
13     skinthickness: int
14     insulin: int
15     bmi: float
16     dpf: float
17     age: int
18
19
20
21  class Config:
22      orm_mode = True
23
24
25  def sayHello(self):
26      print("Hello, I am " + self.name)
27
28
```

## Routes

Added **routes** with **API rerouting**

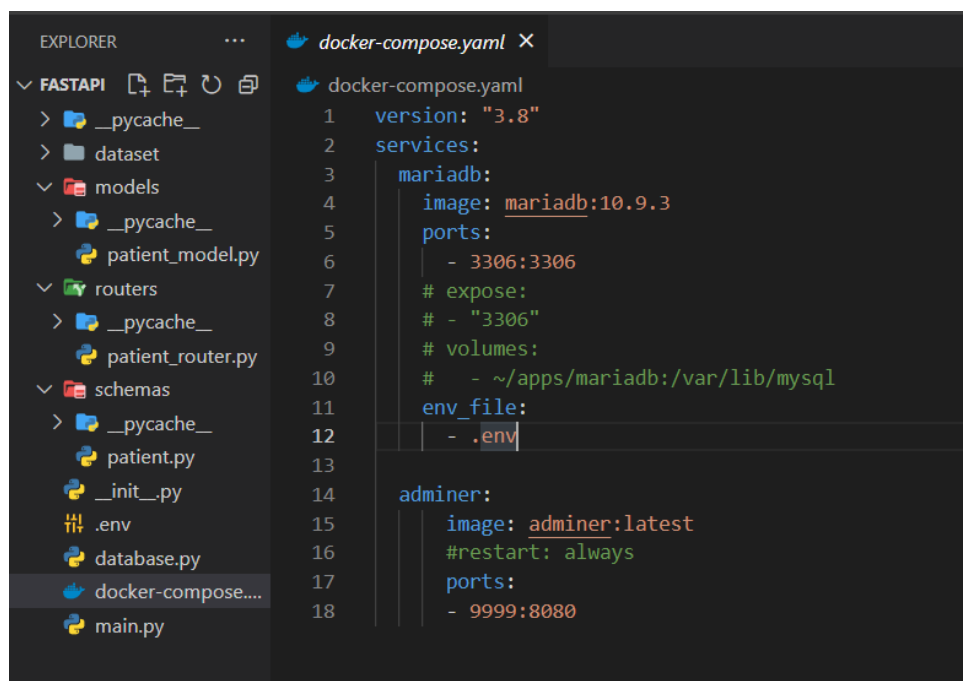


```
1 from fastapi import FastAPI, HTTPException
2 import json
3 #app = FastAPI()
4 from schemas.patient import Patient
5 from fastapi import APIRouter
6 from database import db
7 from models.patient_model import Patient as PatientRepo
8
9
10
11
12 #APIROUTER
13 router = APIRouter(prefix="/patient",
14 tags=["Patient"],
15 responses={404: {"Patient": "Not found"}},
16 )
17
18
19 #OPLETTEN /patient door APIrouter!
20
21
22 #GET
23 @router.get("/all")
24 async def get_all():
25     patients = db.patients.find()
26     return patients
27
28 #POST
29 @router.post("/add_one")
30 async def create(patient: Patient):
31     patient = patient.dict()
32     db.patients.insert_one(patient)
33     return patient
34
35 #GET PATIENT BY ID
36 @router.get("/{id}")
37 async def get_by_id(id: str):
38     patient = db.patients.find_one({"_id": id})
39     if patient:
```

## Database with Docker

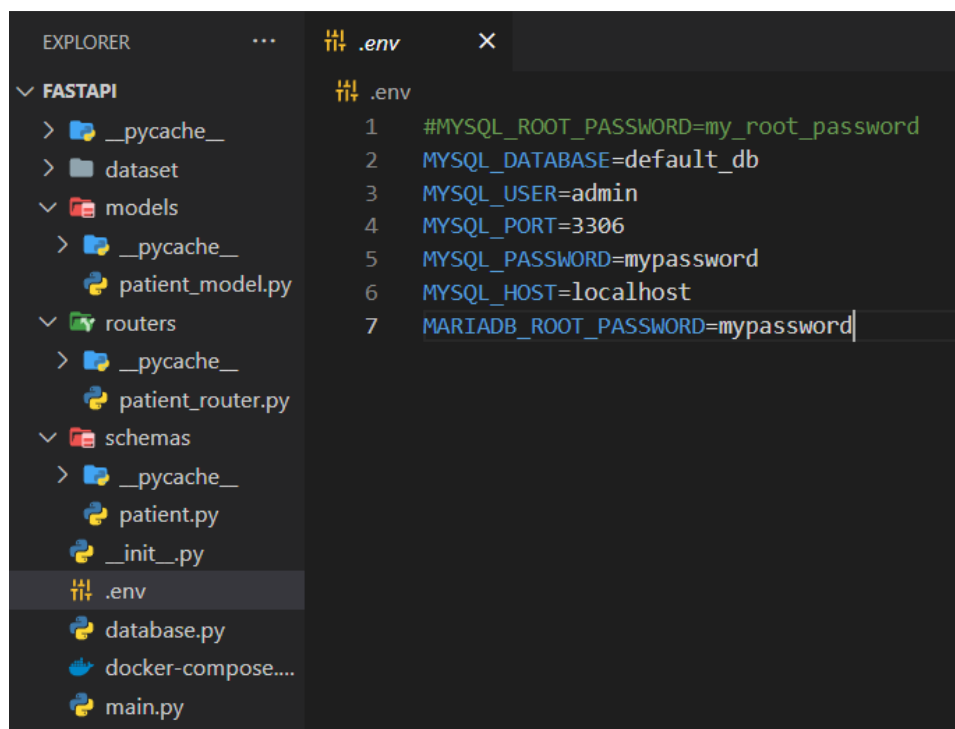
Setting up **database** with a **docker-compose.yml** file

Login to database with **.env** file



The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows a project structure for 'FASTAPI' with folders like '\_\_pycache\_\_', 'dataset', 'models', 'routers', 'schemas', and files like 'patient\_model.py', 'patient\_router.py', 'patient.py', 'init.py', '.env', 'database.py', 'docker-compose....', and 'main.py'. The Editor displays the 'docker-compose.yml' file with the following content:

```
1 version: "3.8"
2 services:
3   mariadb:
4     image: mariadb:10.9.3
5     ports:
6       - 3306:3306
7     # expose:
8     #   - "3306"
9     # volumes:
10    #   - ~/apps/mariadb:/var/lib/mysql
11    env_file:
12      - .env
13
14  adminer:
15    image: adminer:latest
16    #restart: always
17    ports:
18      - 9999:8080
```

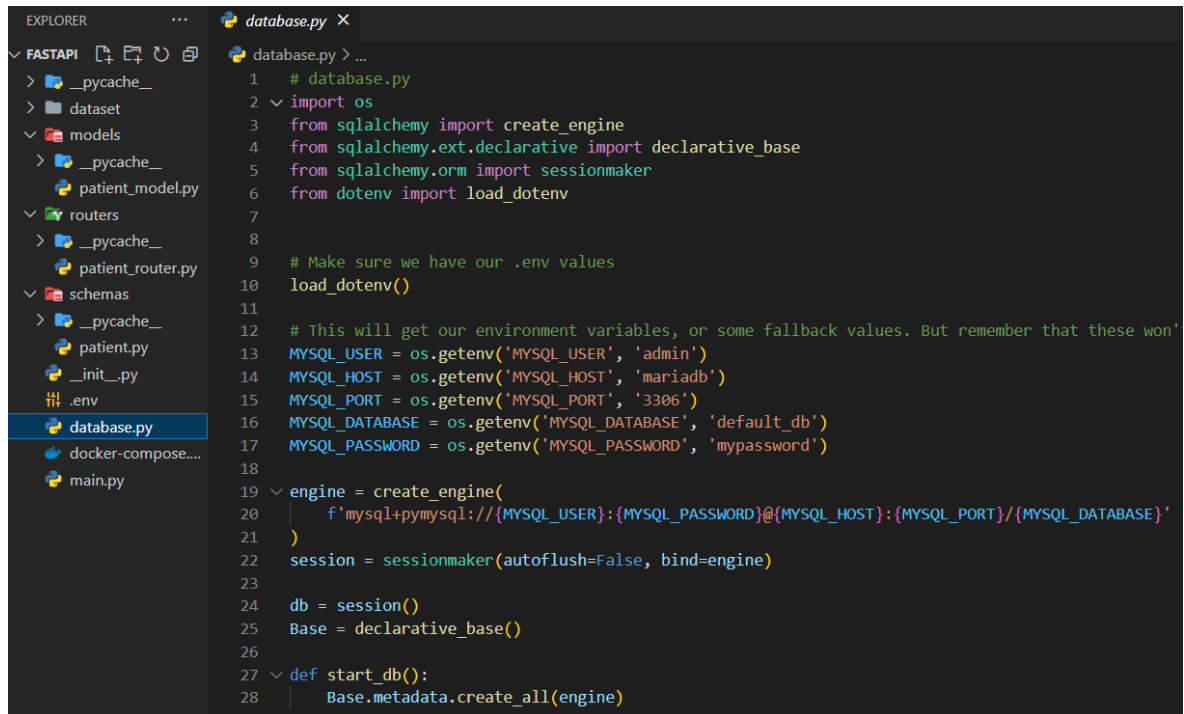


The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows the same project structure as the previous screenshot, with the '.env' file selected. The Editor displays the '.env' file with the following content:

```
1 #MYSQL_ROOT_PASSWORD=my_root_password
2 MYSQL_DATABASE=default_db
3 MYSQL_USER=admin
4 MYSQL_PORT=3306
5 MYSQL_PASSWORD=mypassword
6 MYSQL_HOST=localhost
7 MARIADB_ROOT_PASSWORD=mypassword
```

## Connect database

Connect to database with **database.py**



```
1  # database.py
2  import os
3  from sqlalchemy import create_engine
4  from sqlalchemy.ext.declarative import declarative_base
5  from sqlalchemy.orm import sessionmaker
6  from dotenv import load_dotenv
7
8
9  # Make sure we have our .env values
10 load_dotenv()
11
12 # This will get our environment variables, or some fallback values. But remember that these won't
13 MYSQL_USER = os.getenv('MYSQL_USER', 'admin')
14 MYSQL_HOST = os.getenv('MYSQL_HOST', 'mariadb')
15 MYSQL_PORT = os.getenv('MYSQL_PORT', '3306')
16 MYSQL_DATABASE = os.getenv('MYSQL_DATABASE', 'default_db')
17 MYSQL_PASSWORD = os.getenv('MYSQL_PASSWORD', 'mypassword')
18
19 engine = create_engine(
20     f'mysql+pymysql://{MYSQL_USER}:{MYSQL_PASSWORD}@{MYSQL_HOST}:{MYSQL_PORT}/{MYSQL_DATABASE}'
21 )
22 session = sessionmaker(autoflush=False, bind=engine)
23
24 db = session()
25 Base = declarative_base()
26
27 def start_db():
28     Base.metadata.create_all(engine)
```



## Database tables

Create database tables with **patient\_model**

```
class Patient(Base):
    __tablename__ = 'patients'

    uuid = Column(String(150), primary_key=True, default=generate_uuid, name="uuid")
    id = Column('id', String(20))
    name = Column('name', String(20))
    pregnancies = Column('pregnancies', Integer)
    glucose = Column('glucose', Integer)
    bloodpressure = Column('bloodpressure', Integer)
    skinthickness = Column('skinthickness', Integer)
    insulin = Column('insulin', Integer)
    bmi = Column('bmi', Integer)
    dpf = Column('dpf', Integer)
    age = Column('age', Integer)
    # outcome = Column('outcome', Integer)

    #init method

    def __init__(self, *, uuid: str = generate_uuid(), id: str = "", name: str = "", pregn
        print("uuid:", uuid)
        print(len(uuid))

        self.uuid = uuid
        self.id = id
        self.name = name
        self.pregnancies = pregnancies
        self.glucose = glucose
        self.bloodpressure = bloodpressure
        self.skinthickness = skinthickness
        self.insulin = insulin
        self.bmi = bmi
        self.dpf = dpf
        self.age = age
```

Register Patient Schema as an ORM-ready object

```
class Bird(BaseModel):
    # uuid: Optional[str]
    id: str
    name: str
    short: str
    image: str
    recon: list
    food: dict
    see: str

    class Config:
        orm_mode = True

    def sayHello(self):
        print("Hello, I am " + self.name)
```

## Repository and queries

Make a repository and add queries from repository

GET http://127.0.0.1:8000/patient/get\_all\_from\_repo

Send 200 OK 3.99 ms 357 B

JSON Auth Query Headers 1 Docs

Preview Headers 4 Cookies Timeline

```
1 [
2 {
3   "uuid": "09ffd707-a8c7-4a09-967b-303235002461",
4   "pregnancies": 5,
5   "bloodpressure": 1,
6   "insulin": 1,
7   "dpf": 100,
8   "glucose": 1,
9   "id": "jan",
10  "name": "dominic",
11  "skinthickness": 1,
12  "bmi": 30,
13  "age": 30
14 },
15 {
16   "uuid": "184c4126-f75a-4b9f-9a0b-a1c807c32dae",
17   "pregnancies": 5,
18   "bloodpressure": 1,
19   "insulin": 1,
20   "dpf": 100,
21   "glucose": 1,
22   "id": "ben",
23   "name": "ben",
24   "skinthickness": 1,
25   "bmi": 30,
26   "age": 30
27 }
28 ]
```

```
<Patient gert>
<class 'models.patient_model.Patient'> has been added to the database!
INFO: 127.0.0.1:54485 - "POST /patient/add_from_repo HTTP/1.1" 200 OK
INFO: 127.0.0.1:54487 - "GET /patient/get_all_from_repo HTTP/1.1" 200 OK
```

## Docker deployment

Docker deployment with **Dockerfile**, we also add a **service** to our docker-compose.yaml

```
Dockerfile > ...
1 FROM python:3.10-slim-bullseye as python-base
2
3 # https://python-poetry.org/docs#ci-recommendations
4 ENV POETRY_VERSION=1.2.0
5 ENV POETRY_HOME=/opt/poetry
6 ENV POETRY_VENV=/opt/poetry-venv
7
8 # Tell Poetry where to place its cache and virtual environment
9 ENV POETRY_CACHE_DIR=/opt/.cache
10
11 # Create stage for Poetry installation
12 FROM python-base as poetry-base
13
14 # Creating a virtual environment just for poetry and install it with pip
15 RUN python3 -m venv $POETRY_VENV \
16     && $POETRY_VENV/bin/pip install -U pip setuptools \
17     && $POETRY_VENV/bin/pip install poetry==${POETRY_VERSION}
18
19 # Create a new stage from the base python image
20 FROM python-base as example-app
21
```

```
api:
  image: api:latest
  build:
    context: .
    dockerfile: Dockerfile
  env_file:
    - .env
  ports:
    - 5000:5000
  depends_on:
    - mariadb
```

## Github Container Registry


Generate a token from Github and login with Github account

```
C:\Users\domin>set PAT=[REDACTED]

C:\Users\domin>echo %PAT%
[REDACTED]

C:\Users\domin>echo %PAT% | docker login ghcr.io --username HoDominic --password-stdin
Login Succeeded
```

## Push image to Github

 **assignment-image** Private

Published 44 seconds ago by HoDominic

↓ 0

## Kubernetes

To scale out our API accross multiple machines, and get them updated in a perfect manner, we use **Kubernetes**

Create deployment, add service, port-forward and scale it

```
PS C:\Users\domin\OneDrive\Bureaublad\MCT3\ML_OPS\Project_mlops\fastapi-env\fastapi> kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-app	3/3	3	3	70d
nginx-app-assignment	3/3	3	3	87m
nginx-deployment	2/2	2	2	67d

```
PS C:\Users\domin\OneDrive\Bureaublad\MCT3\ML_OPS\Project_mlops\fastapi-env\fastapi> kubectl port-forward service/nginx-http-assignment 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

```
PS C:\Users\domin\OneDrive\Bureaublad\MCT3\ML_OPS\Project_mlops\fastapi-env\fastapi> kubectl get service
```


NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	76d
my-service	ClusterIP	10.99.182.31	<none>	80/TCP	67d
nginx-http	ClusterIP	10.101.210.26	<none>	80/TCP	70d
nginx-http-assignment	ClusterIP	10.105.236.60	<none>	80/TCP	85m
vue-dockerservice	ClusterIP	10.101.120.89	<none>	80/TCP	11d

```
PS C:\Users\domin\OneDrive\Bureaublad\MCT3\ML_OPS\Project_mlops\fastapi-env\fastapi> kubectl scale deployment nginx-app-assignment --replicas=10
deployment.apps/nginx-app-assignment scaled
PS C:\Users\domin\OneDrive\Bureaublad\MCT3\ML_OPS\Project_mlops\fastapi-env\fastapi>
```

We need a **cluster-role-binding.yaml** and **service-account.yaml** and a **token** to access the **Kubernetes dashboard**

```
kubernetes > service-account.yaml
```

```
1  apiVersion: v1
2  kind: ServiceAccount
3  metadata:
4    name: admin-user
5    namespace: kubernetes-dashboard
```



```
kubernetes >  cluster-role-binding.yaml
1  apiVersion: rbac.authorization.k8s.io/v1
2  kind: ClusterRoleBinding
3  metadata:
4    name: admin
5  roleRef:
6    apiGroup: rbac.authorization.k8s.io
7    kind: ClusterRole
8    name: cluster-admin
9  subjects:
10 - kind: ServiceAccount
11   name: admin
12   namespace: kubernetes-dashboard
```

## Secret

Create secret from Docker Registry and deploy API with Kubernetes

```
kubernetes > deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: fastapi-deployment
5    namespace: assignment-fastapi-gui
6  spec:
7    replicas: 3
8    selector:
9      matchLabels:
10       app: nginx-app-assignment
11    template:
12      metadata:
13        labels:
14          app: nginx-app-assignment
15      spec:
16        containers:
17          - name: nginx-app-assignment
18            image: ghcr.io/hodominic/assignment-image
19            ports:
20              - containerPort: 80
21          imagePullSecrets:
22            - name: regcredassignment
```

## Deployments

Name	Namespace	Images	Labels
 fastapi-deployment	assignment-fastapi-gui	ghcr.io/hodominic/assignment-image	-
 nginx-app-assignment	default	nginx	app: nginx-app-assignment