**RETO 2: DESARROLLO DE UN API REST CON FASTAPI Y DESPLIEGUE EN KUBERNETES**

1. **Modelo de Datos y CRUD:**

* ID: Identificador único (Autoincremental)
* Nombre del Pasajero: Texto (max. 100 caracteres)
* Origen: Texto (max. 100 caracteres)
* Destino: Texto (max. 100 caracteres)
* Fecha de Vuelo: Fecha y Hora
* Estado de la Reserva: Texto (max. 20 caracteres)

from datetime import datetime

from fastapi import FastAPI, HTTPException, Depends

from sqlalchemy import create\_engine, Column, Integer, String, DateTime

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker, Session

from pydantic import BaseModel

# Configuración de la base de datos

SQLALCHEMY\_DATABASE\_URL = "sqlite:///./test.db"

engine = create\_engine(SQLALCHEMY\_DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative\_base()

# Definición del modelo

class Book(Base):

    \_\_tablename\_\_ = "reservavuelo"

    id = Column(Integer, primary\_key=True, index=True)

    nombre\_pasajero = Column(String(100))

    origen = Column(String(100))

    destino = Column(String(100))

    fecha = Column(DateTime)

    estado = Column(String(20))

# Crea la tabla si no existe

Base.metadata.create\_all(bind=engine)

# Definición de modelos Pydantic para entrada y salida

class BookCreate(BaseModel):

    nombre\_pasajero: str

    origen: str

    destino: str

    fecha: datetime

    estado: str

class BookUpdate(BaseModel):

    nombre\_pasajero: str

    origen: str

    destino: str

    fecha: datetime

    estado: str

class BookOut(BaseModel):

    nombre\_pasajero: str

    origen: str

    destino: str

    fecha: datetime

    estado: str

# Función para obtener una sesión de la base de datos

def get\_db():

    db = SessionLocal()

    return db

# Inicializa la aplicación FastAPI

app = FastAPI()

# Ruta para obtener todas las reseñas

@app.get("/reservas/", response\_model=list[BookOut])

async def read\_booking(skip: int = 0, limit: int = 10, db: Session = Depends(get\_db)):

    books = db.query(Book).offset(skip).limit(limit).all()

    return books

# Ruta para crear una reseña

@app.post("/reservas/", response\_model=BookOut)

async def create\_booking(book: BookCreate, db: Session = Depends(get\_db)):

    db\_book = Book(\*\*book.dict())

    db.add(db\_book)

    db.commit()

    db.refresh(db\_book)

    return db\_book

# Ruta para obtener una reseña por ID

@app.get("/reservas/{book\_id}", response\_model=BookOut)

async def read\_book(book\_id: int, db: Session = Depends(get\_db)):

    db\_book = db.query(Book).filter(Book.id == book\_id).first()

    if db\_book is None:

        raise HTTPException(status\_code=404, detail="book not found")

    return db\_book

# Ruta para actualizar una reseña por ID

@app.put("/reservas/{book\_id}", response\_model=BookOut)

async def update\_book(book\_id: int, book: BookUpdate, db: Session = Depends(get\_db)):

    db\_book = db.query(Book).filter(Book.id == book\_id).first()

    if db\_book is None:

        raise HTTPException(status\_code=404, detail="book not found")

    db\_book.nombre\_pasajero = book.nombre\_pasajero

    db\_book.origen = book.origen

    db\_book.destino = book.destino

    db\_book.fecha = book.fecha

    db\_book.estado = book.estado

    db.commit()

    return db\_book

# Ruta para eliminar una reseña por ID

@app.delete("/books/{book\_id}")

async def delete\_book(book\_id: int, db: Session = Depends(get\_db)):

    db\_book = db.query(Book).filter(Book.id == book\_id).first()

    if db\_book is None:

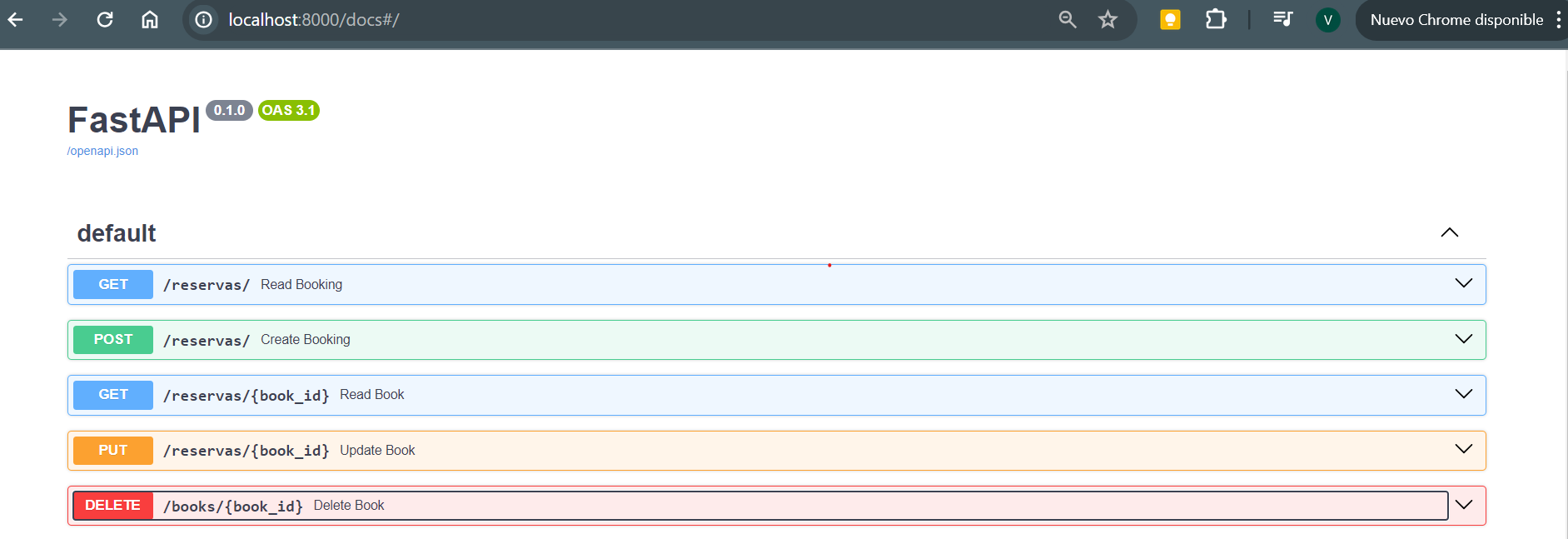
        raise HTTPException(status\_code=404, detail="book not found")

    db.delete(db\_book)

    db.commit()

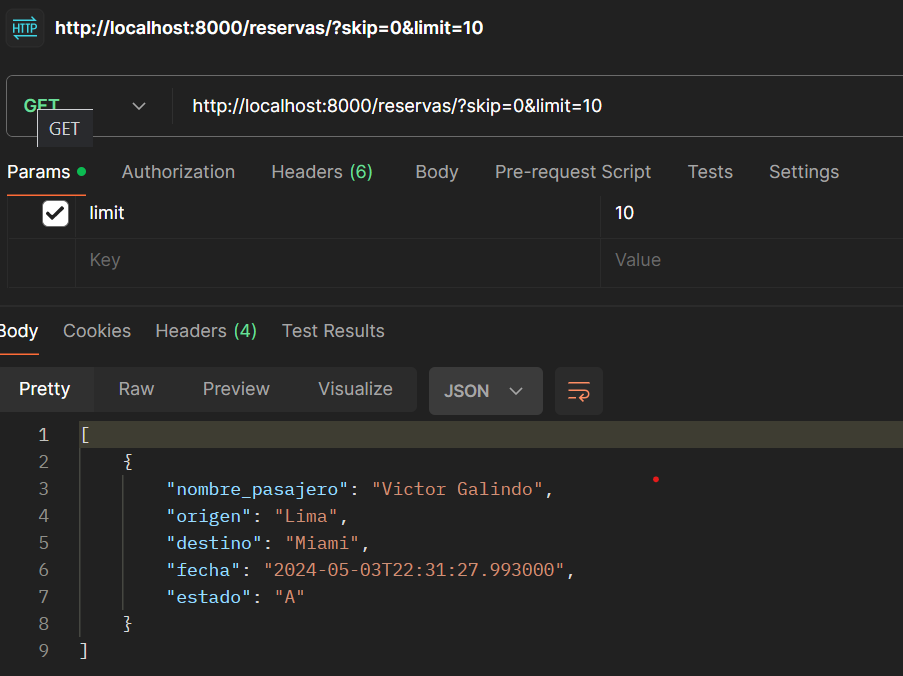
    return {"message": "book deleted"}

1. **API REST con FASTAPI**

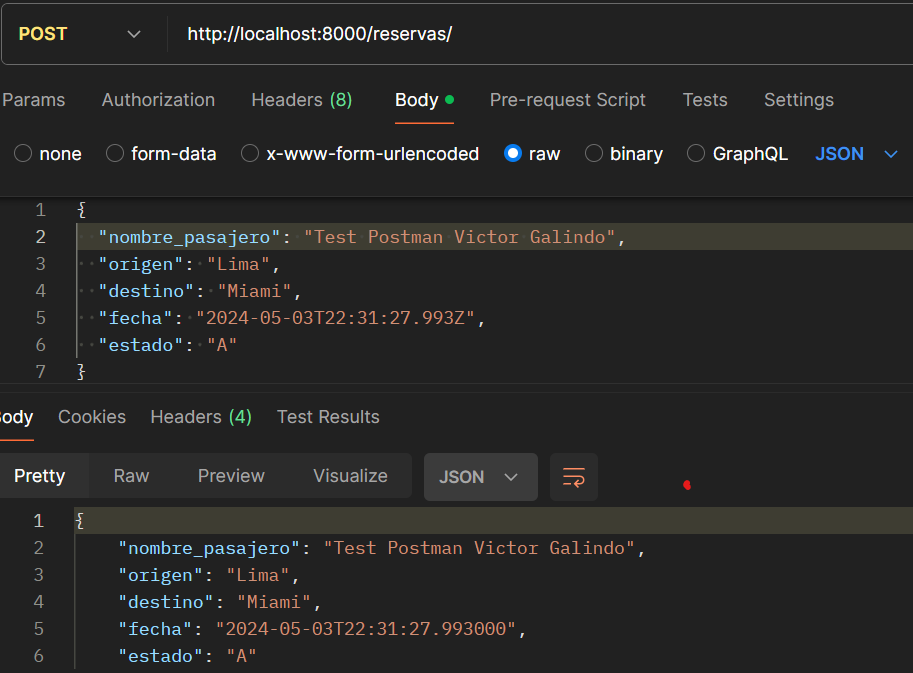
****

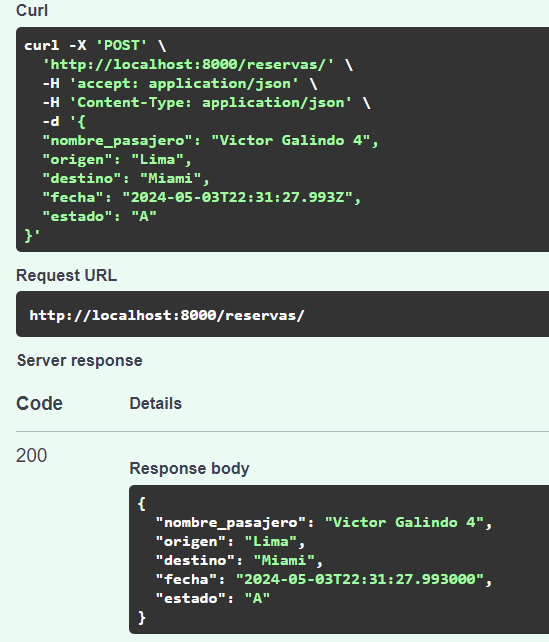
1. **Prueba cada una de las operaciones CRUD utilizando herramientas como Postman**

**GET**

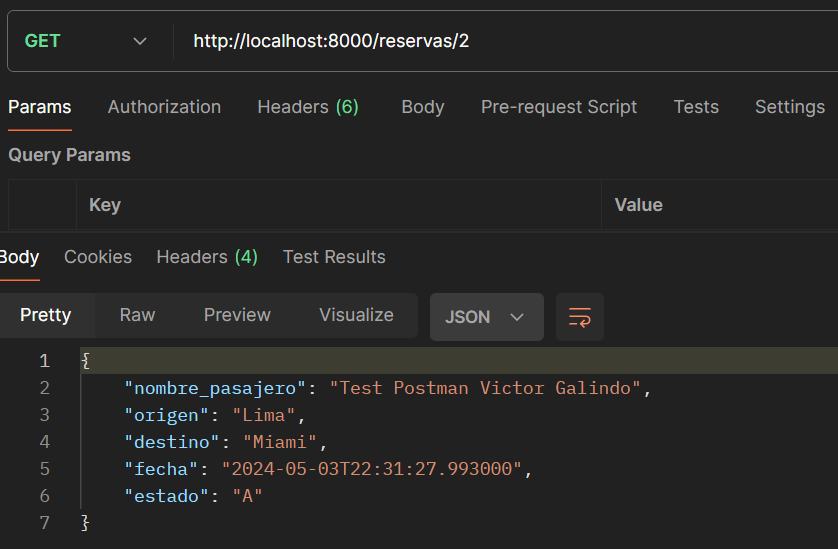
****

**POST**

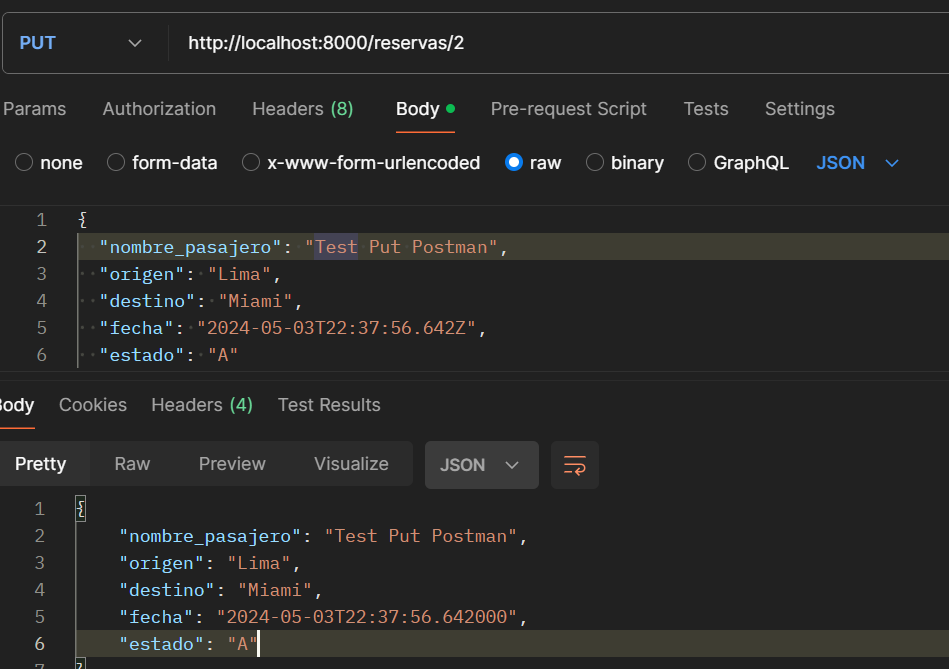
****

****

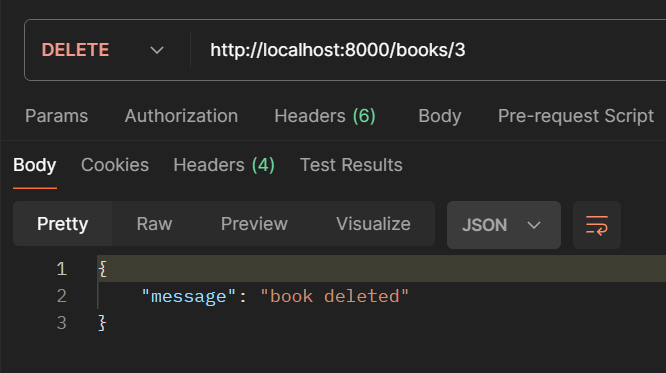
**GET con ID**

****

**PUT**

****

**DELETE**

****

1. **CONTENEDOR DOCKER**

FROM python:3.12.2-alpine3.19

LABEL maintainer="SmartVGC.com"

ENV PYTHONUNBUFFERED 1

WORKDIR /app

COPY . /app

COPY ./requirements.txt /tmp/requirements.txt

EXPOSE 8000

ARG DEV=false

RUN python -m venv /py && \

    /py/bin/pip install --upgrade pip && \

    /py/bin/pip install -r /tmp/requirements.txt && \

    rm -rf /tmp && \

    adduser \

        --disabled-password \

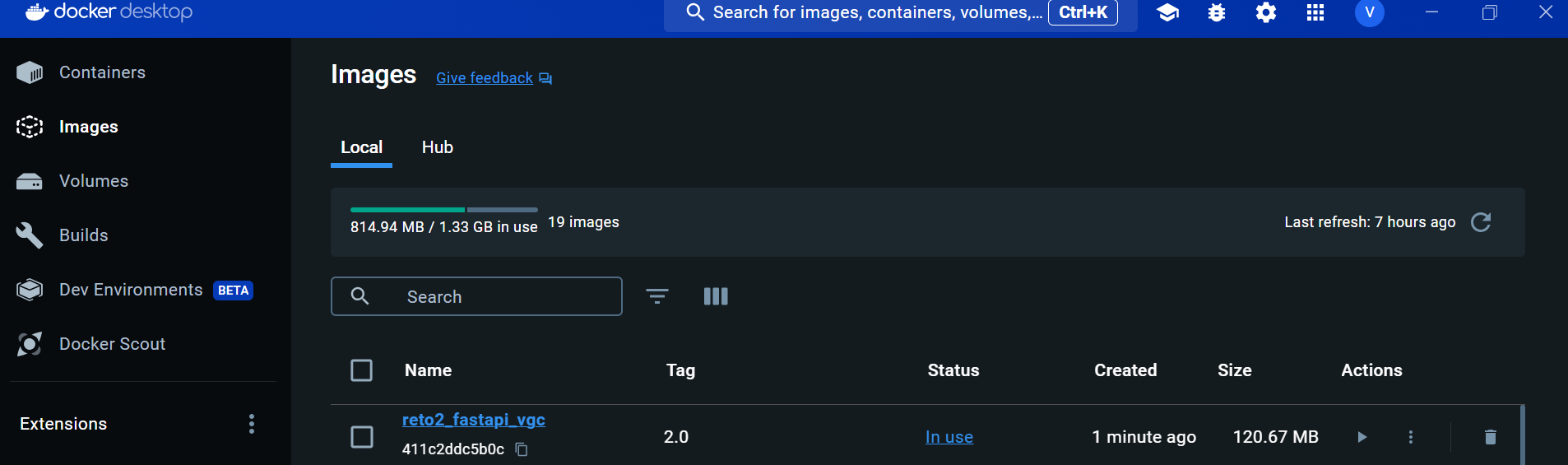
        --no-create-home \

        kub-user

ENV PATH="/py/bin:$PATH"

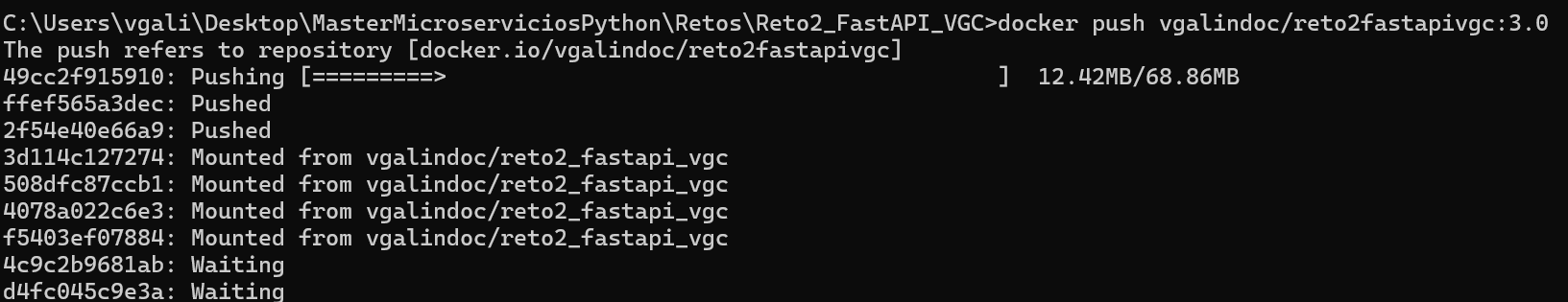
USER kub-user

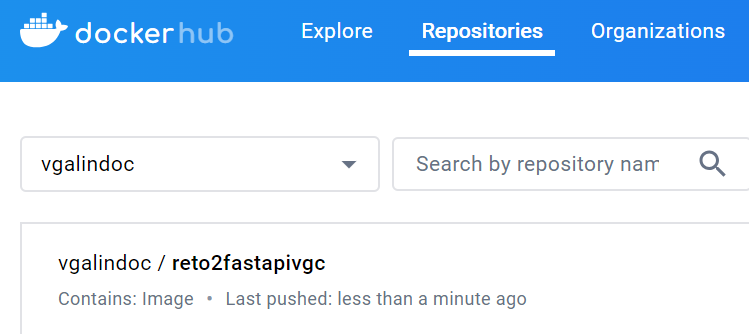
CMD ["uvicorn", "main-model-db:app", "--host", "0.0.0.0", "--port", "8000"]

****

1. **DESPLIEGUE EN KUBERNETS CON MINICUBE**

**Publicación en DockerHUB**

****

****

1. **Crea un archivo YAML con el manifiesto de Kubernetes para desplegar la aplicación en un clúster utilizando Minikube.**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: backendpy

  namespace: practice-py

  labels:

    apipy: crm

spec:

  replicas: 2

  selector:

    matchLabels:

      apipy: crm

  template:

    metadata:

      labels:

        apipy: crm

    spec:

      containers:

      - name: reto2fastapivgc

        image: vgalindoc/reto2fastapivgc:3.0

        ports:

        - containerPort: 8000

---

apiVersion: v1

kind: Service

metadata:

  name: service-crm

  namespace: practice-py

spec:

  selector:

    apipy: crm

  ports:

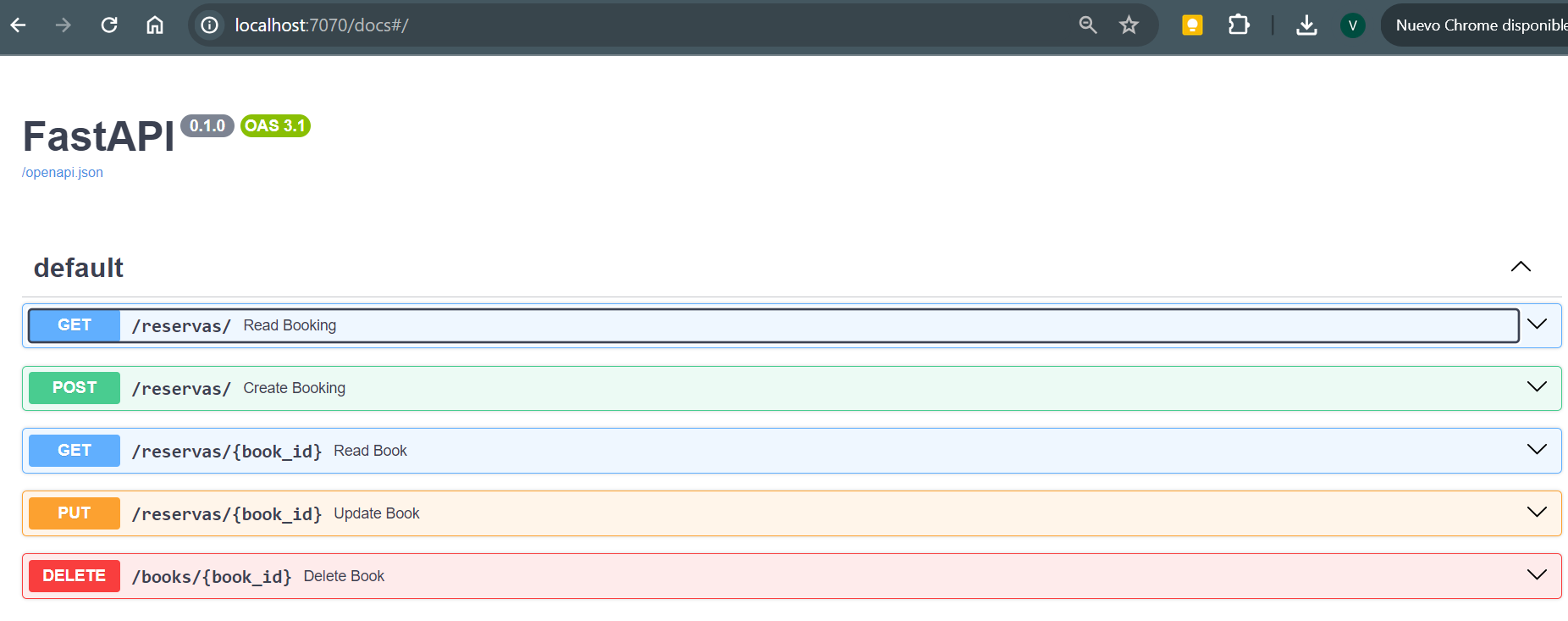
  - protocol: TCP

    port: 8000

    targetPort: 7070

1. **Despliega la aplicación en Minikube y verifica su funcionamiento**.



****