Proyecto v1

Daniel Fernando Esai Ramirez Callo

Computacion Tolerante a Fallas

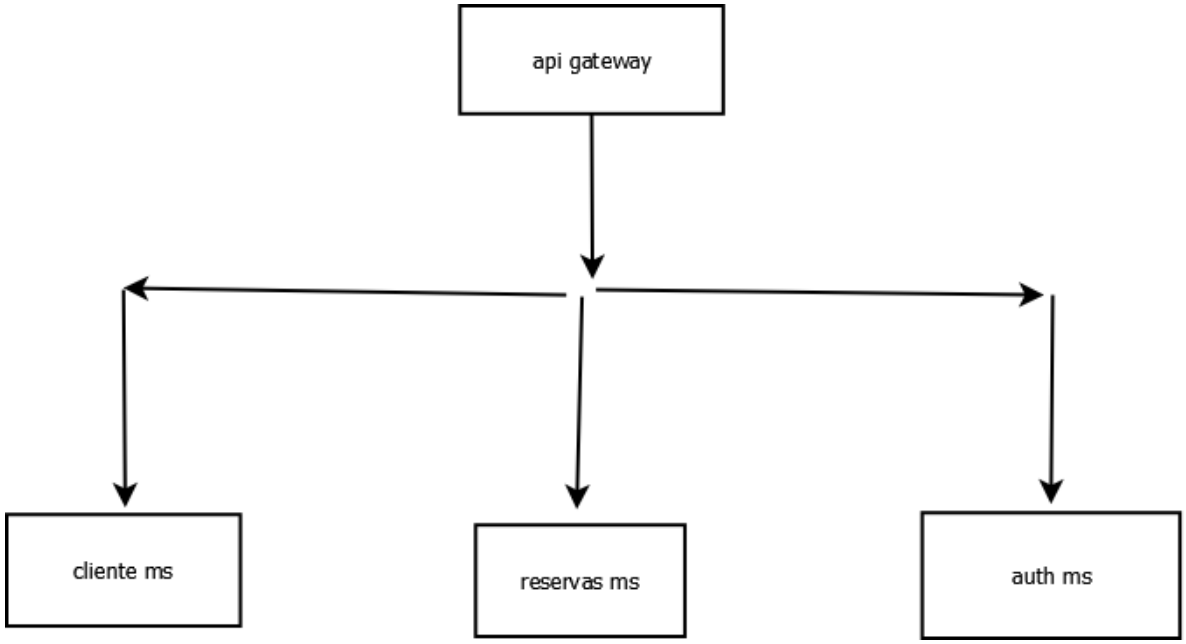
Descripción del proyecto

Este proyecto implementa un **sistema de reservas** utilizando **arquitectura de microservicios**, completamente contenerizado con **Docker** y desplegado en **Kubernetes**.
Incluye microservicios independientes, comunicación REST, mecanismos básicos de autenticación, escalabilidad, tolerancia a fallos y guía de despliegue paso a paso.

Este sistema permite:

- Registrar clientes
- Crear reservas
- Consultar reservas por cliente
- Cancelar reservas
- Comunicación entre microservicios
- Despliegue automático mediante Docker + Kubernetes

Arquitectura del sistema



Microservicios incluidos

Microservicio	Función	Puerto
clientes-ms	Registrar, consultar y eliminar clientes	5001
reservas-ms	Crear, listar y cancelar reservas	5002
auth-ms	Autenticación básica (tokens falsos)	5003
api-gateway	Entrada unificada a todo el sistema	5000

Estructura del repositorio

/proyecto-reservas

```
│
├── clientes-ms
│   ├── app.py
│   ├── database.db
│   └── Dockerfile
│
├── reservas-ms
│   ├── app.py
│   ├── database.db
│   └── Dockerfile
│
├── auth-ms
│   ├── app.py
│   └── Dockerfile
│
├── api-gateway
│   ├── app.py
│   └── Dockerfile
│
```

```
└── kubernetes
    ├── clientes.yaml
    ├── reservas.yaml
    ├── auth.yaml
    ├── gateway.yaml
```

Microservicio: CLIENTES (clientes-ms/app.py)

```
from flask import Flask, request, jsonify
import sqlite3

app = Flask(__name__)

def db():
   conn = sqlite3.connect("database.db")
   conn.row_factory = sqlite3.Row
   return conn

@app.route("/clientes", methods=["POST"])
def crear_cliente():
   data = request.json
   conn = db()
   conn.execute("INSERT INTO clientes (nombre, correo) VALUES (?, ?)",
            (data["nombre"], data["correo"]))
```

```
    conn.commit()
    return jsonify({"msg": "Cliente creado"}), 201

@app.route("/clientes/<correo>", methods=["GET"])
def obtener_cliente(correo):
    conn = db()
    cliente = conn.execute("SELECT * FROM clientes WHERE correo=?",
(correo,)).fetchone()
    if cliente:
        return jsonify(dict(cliente))
    return jsonify({"error": "Cliente no encontrado"}), 404

if __name__ == "__main__":
    conn = db()
    conn.execute("CREATE TABLE IF NOT EXISTS clientes(id INTEGER
PRIMARY KEY, nombre TEXT, correo TEXT UNIQUE)")
    conn.commit()
    app.run(host="0.0.0.0", port=5001)
```

Dockerfile — clientes-ms

```
FROM python:3.10
WORKDIR /app
COPY . .
RUN pip install flask
```

```
EXPOSE 5001
CMD ["python", "app.py"]
```

Microservicio: RESERVAS (reservas-ms/app.py)

```
from flask import Flask, request, jsonify
import sqlite3

app = Flask(__name__)

def db():
    conn = sqlite3.connect("database.db")
    conn.row_factory = sqlite3.Row
    return conn

@app.route("/reservas", methods=["POST"])
def crear_reserva():
    data = request.json
    conn = db()
    conn.execute("INSERT INTO reservas (correo, fecha, hora) VALUES (?, ?, ?)",
                 (data["correo"], data["fecha"], data["hora"]))
    conn.commit()
    return jsonify({"msg": "Reserva creada"}), 201
```

```
@app.route("/reservas/<correo>", methods=["GET"])
def obtener_reservas(correo):
    conn = db()
    reservas = conn.execute("SELECT * FROM reservas WHERE correo=?", (correo,)).fetchall()
    return jsonify([dict(r) for r in reservas])

if __name__ == "__main__":
    conn = db()
    conn.execute("CREATE TABLE IF NOT EXISTS reservas(id INTEGER PRIMARY KEY, correo TEXT, fecha TEXT, hora TEXT)")
    conn.commit()
    app.run(host="0.0.0.0", port=5002)
```

Dockerfile — reservas-ms

```
FROM python:3.10
WORKDIR /app
COPY . .
RUN pip install flask
EXPOSE 5002
CMD ["python", "app.py"]
```

Microservicio: AUTH (auth-ms/app.py)

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route("/login", methods=["POST"])
def login():
    data = request.json
    if data["usuario"] == "admin" and data["password"] == "123":
        return jsonify({"token": "token-valido"})
    return jsonify({"error": "Credenciales inválidas"}), 401

@app.route("/validar", methods=["POST"])
def validar():
    token = request.json.get("token")
    if token == "token-valido":
        return jsonify({"valid": True})
    return jsonify({"valid": False})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5003)
```

Dockerfile — auth-ms

```
FROM python:3.10
WORKDIR /app
COPY . .
RUN pip install flask
EXPOSE 5003
CMD ["python", "app.py"]
```

Microservicio: API Gateway (api-gateway/app.py)

```
from flask import Flask, request, jsonify
import requests

app = Flask(__name__)

URL_CLIENTES = "http://clientes-ms:5001"
URL_RESERVAS = "http://reservas-ms:5002"
URL_AUTH = "http://auth-ms:5003"
```

```
@app.route("/reservar", methods=["POST"])
def gateway_reserva():
    token = request.headers.get("Authorization")

    try:
        auth = requests.post(f"{URL_AUTH}/validar", json={"token": token}).json()
        if not auth["valid"]:
            return jsonify({"error": "Token inválido"}), 403
    except:
        return {"error": "Auth no disponible"}, 500

    reserva = requests.post(f"{URL_RESERVAS}/reservas", json=request.json).json()
    return reserva

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Dockerfile — api-gateway

```
FROM python:3.10
WORKDIR /app
COPY . .
```

```
RUN pip install flask requests

EXPOSE 5000

CMD ["python", "app.py"]
```

3. Kubernetes — Archivos YAML

Todos deben ir dentro de la carpeta /kubernetes/.

clientes.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: clientes-ms
spec:
  replicas: 1
  selector:
    matchLabels:
      app: clientes-ms
  template:
    metadata:
      labels:
        app: clientes-ms
    spec:
      containers:
      - name: clientes-ms
```

```
      image: clientes-ms:1.0
      ports:
      - containerPort: 5001

---
apiVersion: v1
kind: Service
metadata:
  name: clientes-ms
spec:
  selector:
    app: clientes-ms
  ports:
  - port: 5001
    targetPort: 5001

reservas.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: reservas-ms
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: reservas-ms
  template:
    metadata:
      labels:
        app: reservas-ms
    spec:
      containers:
      - name: reservas-ms
        image: reservas-ms:1.0
        ports:
        - containerPort: 5002


---
apiVersion: v1
kind: Service
metadata:
  name: reservas-ms
spec:
  selector:
    app: reservas-ms
  ports:
```

```
  - port: 5002
    targetPort: 5002
```

auth.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-ms
spec:
  replicas: 1
  selector:
    matchLabels:
      app: auth-ms
  template:
    metadata:
      labels:
        app: auth-ms
    spec:
      containers:
      - name: auth-ms
        image: auth-ms:1.0
        ports:
        - containerPort: 5003
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: auth-ms
spec:
  selector:
    app: auth-ms
  ports:
  - port: 5003
    targetPort: 5003

gateway.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
spec:
  replicas: 1
  selector:
    matchLabels:
      app: api-gateway
```

```
 template:
  metadata:
   labels:
    app: api-gateway
  spec:
   containers:
   - name: api-gateway
    image: api-gateway:1.0
    ports:
    - containerPort: 5000


---
apiVersion: v1
kind: Service
metadata:
 name: api-gateway
spec:
 type: NodePort
 selector:
  app: api-gateway
 ports:
 - port: 5000
  nodePort: 30000
```

targetPort: 5000

Guía completa de despliegue (TUTORIAL)

equisitos

- Docker
- Kubernetes (Minikube recomendado)
- kubectl
- Git

PASO 1 — Clonar el repositorio

git clone https://github.com/usuario/proyecto-reservas.git

cd proyecto-reservas

PASO 2 — Construir las imágenes Docker

docker build -t clientes-ms:1.0 clientes-ms/

docker build -t reservas-ms:1.0 reservas-ms/

docker build -t auth-ms:1.0 auth-ms/

docker build -t api-gateway:1.0 api-gateway/

Iniciar Minikube

minikube start

PASO 4 — Cargar las imágenes a Minikube

minikube image load clientes-ms:1.0

minikube image load reservas-ms:1.0

minikube image load auth-ms:1.0

minikube image load api-gateway:1.0

PASO 5 — Aplicar los manifiestos de Kubernetes

kubectl apply -f kubernetes/clientes.yaml

kubectl apply -f kubernetes/reservas.yaml

kubectl apply -f kubernetes/auth.yaml

kubectl apply -f kubernetes/gateway.yaml

PASO 6 — Obtener URL del API Gateway

Ejemplo de salida:

http://127.0.0.1:30000

PASO 7 — Probar la aplicación

1-Obtener token

```
curl -X POST http://127.0.0.1:30000/login \
-H "Content-Type: application/json" \
-d '{"usuario":"admin","password":"123"}'
```

2-Crear un cliente

curl -X POST http://127.0.0.1:30000/clientes \

-H "Content-Type: application/json" \

-d '{"nombre":"Daniel", "correo":"daniel@test.com"}'

Crear una reserva

curl -X POST http://127.0.0.1:30000/reservar \

-H "Content-Type: application/json" \

-H "Authorization: token-valido" \

-d '{"correo":"daniel@test.com","fecha":"2025-01-01","hora":"18:00"}'

4-**Obtener reservas**

curl http://127.0.0.1:30000/reservas/daniel@test.com

Monitorización y Observabilidad (Opcional)

minikube addons enable metrics-server

Chaos Engineering (Opcional)

kubectl delete pod -l app=reservas-ms