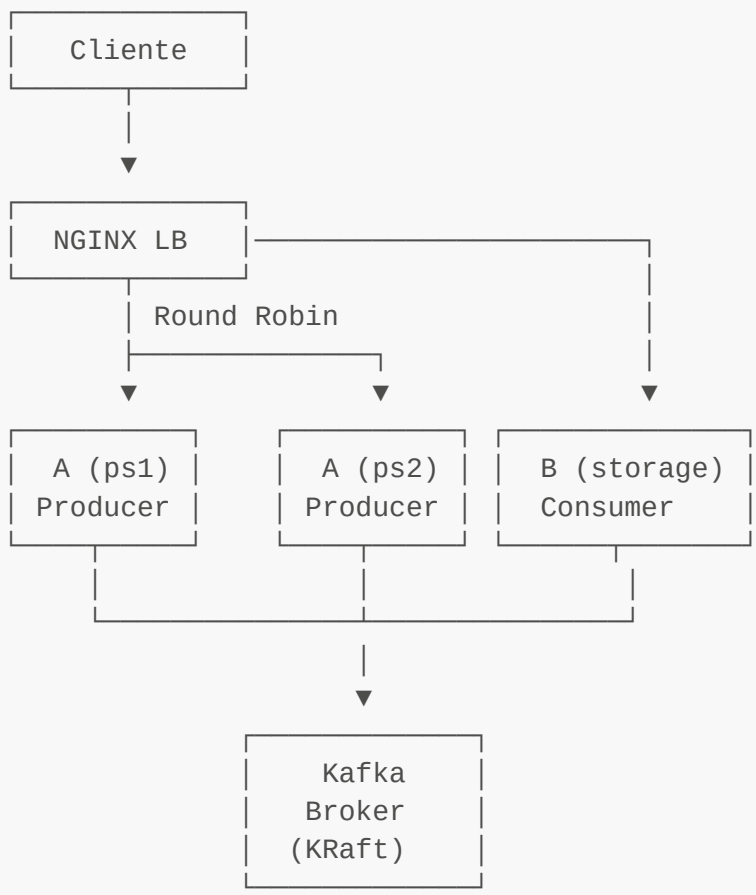
Práctica de Microservicios - M14

Descripción

Sistema de microservicios para gestión de contraseñas con arquitectura dividida en dos servicios y comunicación asíncrona mediante Kafka:

- **password-service** (Microservicio A): API pública con lógica de negocio y cifrado (Producer de Kafka)
- **storage-sqlite** (Microservicio B): API interna CRUD genérica para SQLite (Consumer de Kafka)
- **Kafka**: Broker de mensajería para eventos asíncronos

Arquitectura



Estructura del Proyecto

```
m14-microservicios/
├── microservices/
│   ├── password-service/     # Microservicio A
│   │   ├── src/
│   │   ├── Dockerfile
│   │   └── package.json
│   └── storage-sqlite/       # Microservicio B
```

```
│       ├── src/
│       ├── Dockerfile
│       └── package.json
└── deploy/
    ├── docker-compose.yml
    ├── lb-a/                 # Load Balancer
    ├── frontend/             # Frontend simple
    └── env/                  # Variables de entorno
```

Requisitos

- Docker y Docker Compose
- Node.js 20+ (para desarrollo local)

Instalación y Ejecución

> **Nota**: Migramos de la imagen Bitnami a la Docker Official Image (`apache/kafka`) para operar Kafka en modo **KRaft** (sin ZooKeeper). Esto reduce la complejidad operativa y alinea el despliegue con las guías oficiales de Docker.

1. Configurar Variables de Entorno

Copiar los archivos de ejemplo y ajustar si es necesario:

```
cd deploy/env
# Los archivos .example.env ya están configurados
# Si necesitas cambiar valores, edita los archivos directamente
```

2. Construir y Ejecutar con Docker Compose

```
cd deploy
docker compose up --build
```

Este comando:

- Construye las imágenes de los microservicios
- Configura las redes (pública y privada)
- Inicia todos los servicios (2 réplicas de A, 1 de B, LB, frontend)
- Ejecuta las migraciones automáticamente

Listeners expuestos

- `HOST` listener: `localhost:9092` para clientes externos o pruebas locales
- `DOCKER` listener: `kafka:9093` para comunicación interna entre contenedores
- Kafka corre en modo KRaft, eliminando por completo la dependencia de ZooKeeper.

Pasos rápidos de prueba

1. Crear una contraseña (produce un evento):

```
curl -X POST http://localhost:8080/api/v1/passwords \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Demo Kafka",
    "username": "demo@kafka.io",
    "password": "S3creta!",
    "masterKey": "claveDemoKafka"
  }'
```

2. Verificar que el evento llegó al consumer:

```
docker logs deploy-storage-sqlite-1 | grep "Kafka\|Demo Kafka"
```

3. Consultar los eventos creados:

```
curl -X GET http://localhost:3001/api/v1/audit/password-events \
  -H "X-API-Key: change-me"
```

4. Listar tópicos disponibles (usa el listener interno `kafka:9093`):

```
docker exec deploy-kafka-1 /opt/kafka/bin/kafka-topics.sh --bootstrap-
server kafka:9093 --list
```

3. Verificar que los Servicios Estén Funcionando

```
# Health check del load balancer (debe redirigir a A)
curl http://localhost:8080/health

# Health check directo de storage (solo desde dentro de la red)
# curl http://localhost:3001/health (no accesible desde fuera)

# Frontend
curl http://localhost:3000
```

Endpoints del Microservicio A (password-service)

Todas las llamadas deben ir al Load Balancer en el puerto **8080**.

1. Crear Contraseña

```
curl -X POST http://localhost:8080/api/v1/passwords \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Gmail Personal",
    "description": "Cuenta principal de Gmail",
    "username": "usuario@gmail.com",
    "password": "miContraseñaSegura123!",
    "url": "https://gmail.com",
    "category": "Redes Sociales",
    "notes": "Cuenta creada en 2020",
    "masterKey": "miClaveMaestraSegura123!"
  }'
```

2. Listar Todas las Contraseñas

```
curl http://localhost:8080/api/v1/passwords
```

3. Obtener Contraseña por ID

```
curl http://localhost:8080/api/v1/passwords/1
```

4. Actualizar Contraseña

```
curl -X PUT http://localhost:8080/api/v1/passwords/1 \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Gmail Personal Actualizado",
    "masterKey": "miClaveMaestraSegura123!"
  }'
```

5. Eliminar Contraseña

```
curl -X DELETE "http://localhost:8080/api/v1/passwords/1?
masterKey=miClaveMaestraSegura123!"
```

6. Descifrar Contraseña

```

```

```
curl -X POST http://localhost:8080/api/v1/passwords/1/decrypt \
  -H "Content-Type: application/json" \
  -d '{
    "masterKey": "miClaveMaestraSegura123!"
  }'
```

7. Health Check

```
curl http://localhost:8080/health
```

8. Documentación Swagger

Una vez iniciado el servicio, accede a la documentación Swagger:

- **Password Service**: http://localhost:8080/api (a través del load balancer)
- **Storage SQLite**: http://localhost:3001/api (solo accesible desde dentro de la red Docker)

Nota: Swagger está habilitado por defecto con `ENV_SWAGGER_SHOW=true` en los archivos de configuración.

Comunicación Asíncrona con Kafka

El sistema implementa comunicación asíncrona mediante Kafka para eventos de contraseñas.

Arquitectura Kafka

- **Producer**: `password-service` publica eventos cuando se crean, actualizan o eliminan contraseñas
- **Consumer**: `storage-sqlite` consume eventos y los guarda en una tabla de auditoría
- **Topic**: `passwords.v1.events` (1 partición, replication factor 1)
- **Feature Flag**: `USE_EDA=true` para habilitar/deshabilitar eventos

Esquema de Eventos

```
{
  "eventId": "uuid",
  "type": "password.created | password.updated | password.deleted",
  "at": "ISO-8601 timestamp",
  "schemaVersion": "1",
  "data": {
    "id": "number",
    "title": "string",
    "username": "string",
    "url": "string",
    "category": "string"
  }
}
```

Nota: Los eventos NO incluyen datos sensibles (no hay `encryptedPassword`, `masterKey`, `masterKeyHash`).

Probar Kafka

1. Crear una contraseña (dispara evento)

```
curl -X POST http://localhost:8080/api/v1/passwords \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Gmail Personal",
    "description": "Cuenta principal",
    "username": "usuario@gmail.com",
    "password": "miContraseña123!",
    "url": "https://gmail.com",
    "category": "Redes Sociales",
    "notes": "Test Kafka",
    "masterKey": "miClaveMaestra123!"
  }'
```

2. Verificar eventos en la auditoría

Los eventos se guardan automáticamente en la tabla `audit_password_events` en storage-sqlite.

Ver todos los eventos (desde dentro de la red Docker o usando un proxy):

```
# Consultar eventos directamente en la base de datos (desde dentro del
contenedor)
docker exec -it storage-sqlite-1 sqlite3 /data/database.sqlite "SELECT
eventId, type, occurredAt FROM audit_password_events ORDER BY receivedAt
DESC LIMIT 10;"
```

O usar el endpoint de auditoría (requiere X-API-Key):

```
curl -X GET http://localhost:3001/api/v1/audit/password-events \
  -H "X-API-Key: change-me"
```

Ver estadísticas de eventos:

```
curl -X GET http://localhost:3001/api/v1/audit/password-
events/stats/summary \
  -H "X-API-Key: change-me"
```

Ver un evento específico por eventId:

```
curl -X GET http://localhost:3001/api/v1/audit/password-events/{eventId} \
  -H "X-API-Key: change-me"
```

3. Verificar logs

```
# Ver logs del producer (password-service)
docker logs password-service-1-1 | grep "Published"

# Ver logs del consumer (storage-sqlite)
docker logs storage-sqlite-1 | grep "Kafka\|event"
```

4. Verificar idempotencia

Crear la misma contraseña múltiples veces y verificar que los eventos se procesen correctamente sin duplicados (por eventId único).

Características de Kafka

- **Idempotencia**: Los eventos se deduplican por `eventId` (índice único en la base de datos)
- **DLQ (Dead Letter Queue)**: Eventos que fallan se guardan con tipo `dlq.error` y el error en el campo `error`
- **Reintentos**: El consumer maneja reintentos automáticos
- **Graceful Shutdown**: Ambos servicios manejan cierre limpio de conexiones Kafka

Configuración de Kafka

Variables de entorno (password-service):

```
KAFKA_BROKERS=kafka:9093
KAFKA_TOPIC_PASSWORD_EVENTS=passwords.v1.events
KAFKA_CLIENT_ID=password-service
USE_EDA=true
```

Variables de entorno (storage-sqlite):

```
KAFKA_BROKERS=kafka:9093
KAFKA_TOPIC_PASSWORD_EVENTS=passwords.v1.events
KAFKA_CLIENT_ID=storage-sqlite
KAFKA_GROUP_ID=storage-sqlite-group
```

Notas Importantes

- **Consistencia Eventual**: Los eventos se procesan de forma asíncrona, por lo que puede haber un pequeño retraso entre la creación de la contraseña y el procesamiento del evento.
- **No Bloqueante**: Si Kafka no está disponible, el producer no falla la operación principal, solo registra un warning.
- **Auditoría Completa**: Todos los eventos se guardan en la tabla `audit_password_events` con timestamp, payload y posibles errores.
- **Kafka en KRaft**: El broker utiliza la Docker Official Image (`apache/kafka`) en modo KRaft, por lo que no se despliega ni requiere ZooKeeper.

Endpoints del Microservicio B (storage-sqlite)

Nota: Estos endpoints son internos y requieren el header `X-API-Key`. No están expuestos públicamente.

```
GET    /api/v1/storage/password_manager      # Listar todos
GET    /api/v1/storage/password_manager/:id  # Obtener uno
POST   /api/v1/storage/password_manager      # Crear
PUT    /api/v1/storage/password_manager/:id  # Actualizar
DELETE /api/v1/storage/password_manager/:id  # Eliminar
GET    /health                                # Health check
```

Características Técnicas

Microservicio A (password-service)

- **Cifrado**: AES usando crypto-js
- **Hash**: bcryptjs para claves maestras (12 rounds)
- **Cliente HTTP**: Axios con:
  - Timeout: 3 segundos
  - Retry exponencial: 2 intentos
  - Circuit Breaker: estados closed/open/half-open
- **Seguridad**: No expone `encryptedPassword` ni `masterKeyHash` en respuestas

Microservicio B (storage-sqlite)

- **Base de datos**: SQLite con archivo en volumen persistente
- **ORM**: TypeORM con migrations (sin synchronize)
- **PRAGMA WAL**: Modo Write-Ahead Logging habilitado
- **Autenticación**: X-API-Key para comunicación interna

Circuit Breaker

El Circuit Breaker tiene 3 estados:

- **CLOSED**: Funcionando normalmente
- **OPEN**: Demasiados fallos, rechaza peticiones
- **HALF_OPEN**: Probando si el servicio se recuperó

Configuración:

- Umbral de fallos: 5 (CB_FAILURE_THRESHOLD)
- Tiempo de reset: 15 segundos (CB_RESET_TIMEOUT_MS)

Formato de Respuestas

Respuesta Exitosa

```
{
  "data": {
    "id": 1,
    "title": "Gmail Personal",
    "username": "usuario@gmail.com",
    ...
  }
}
```

Respuesta de Error

```
{
  "code": "NOT_FOUND",
  "message": "Registro no encontrado",
  "traceId": "uuid-v4",
  "retryable": false
}
```

Desarrollo Local

Instalar Dependencias

```
# Microservicio A
cd microservices/password-service
npm install

# Microservicio B
cd microservices/storage-sqlite
npm install
```

Ejecutar en Desarrollo

```
# Microservicio B (puerto 3001)
cd microservices/storage-sqlite
npm run start:dev

# Microservicio A (puerto 3000)
```

```
cd microservices/password-service
npm run start:dev
```

Ejecutar Migraciones (B)

```
cd microservices/storage-sqlite
npm run migration:run
```

Ejecutar Tests

```
# Test unitario en A
cd microservices/password-service
npm test

# Test unitario en B
cd microservices/storage-sqlite
npm test
```

Variables de Entorno

password-service.example.env

```
PORT=3000
LOG_LEVEL=info
STORAGE_BASE_URL=http://storage-sqlite:3001
STORAGE_API_KEY=change-me
CIPHER_SECRET=dev-secret-32bytes-min
REQUEST_TIMEOUT_MS=3000
RETRY_ATTEMPTS=2
CB_FAILURE_THRESHOLD=5
CB_RESET_TIMEOUT_MS=15000
```

storage-sqlite.example.env

```
PORT=3001
LOG_LEVEL=info
SQLITE_DB_PATH=/data/database.sqlite
STORAGE_API_KEY=change-me
```

Redes Docker

- **public_net**: Expone lb-a (puerto 8080) y frontend (puerto 3000)

- **private_net**: Comunicación interna entre A y B

Volúmenes

- **storage_sqlite_data**: Persiste el archivo `database.sqlite` del servicio B

Troubleshooting

Los servicios no se comunican

Verifica que:

1. Las variables de entorno `STORAGE_API_KEY` coincidan en ambos servicios
2. La URL `STORAGE_BASE_URL` sea correcta (usar el nombre del servicio Docker)
3. Ambos servicios estén en la misma red `private_net`

Circuit Breaker está abierto

El Circuit Breaker se abre después de 5 fallos consecutivos. Espera 15 segundos para que pase a estado HALF_OPEN y vuelva a intentar.

Base de datos no se crea

Verifica que:

1. El volumen `storage_sqlite_data` esté creado
2. Las migraciones se ejecuten correctamente
3. El servicio tenga permisos de escritura en `/data`

Limpieza

```
# Detener y eliminar contenedores
cd deploy
docker compose down

# Eliminar también volúmenes (¡CUIDADO! Elimina los datos)
docker compose down -v
```