Sistema Distribuido de Control de Combustible - Empresa XYZ

Objetivo General

Desarrollar un sistema distribuido basado en arquitectura de microservicios utilizando .NET y gRPC para la gestión del consumo de combustible de maquinaria liviana y pesada en la empresa XYZ.

Objetivos Específicos

1. Implementar microservicios independientes para el control de choferes, vehículos, rutas y consumo de combustible.
2. Usar gRPC como mecanismo de comunicación eficiente entre microservicios.
3. Asegurar la disponibilidad y escalabilidad del sistema mediante una arquitectura distribuida.
4. Separar la administración de maquinaria liviana y maquinaria pesada.
5. Garantizar la interoperabilidad entre componentes del sistema distribuidos en diferentes entornos de red.

Arquitectura General del Sistema

- Estilo arquitectónico: Microservicios
- Comunicación entre microservicios mediante gRPC
- Posible uso de API Gateway para exponer servicios externos vía REST
- Persistencia por microservicio (bases de datos separadas o esquema compartido)

Componentes del Sistema

1. Servicio de Choferes
2. Servicio de Vehículos
3. Servicio de Rutas
4. Servicio de Consumo de Combustible
5. Servicio de Autenticación y Autorización

Separación por Maquinaria

- Vehículos y consumo diferenciados por tipo: maquinaria liviana y pesada.
- Servicios incluyen lógica para tratamiento diferenciado según el tipo de maquinaria.

Capas de Cada Microservicio

1. Controllers (gRPC)
2. Application (Lógica de negocio)
3. Domain (Entidades, interfaces)
4. Infrastructure (Acceso a datos, gRPC clients)
5. Persistence (Base de datos)

Seguridad y Autenticación

- Autenticación mediante JWT
- Roles: Admin, Operador, Supervisor

- Autorización por endpoints en gRPC o API Gateway

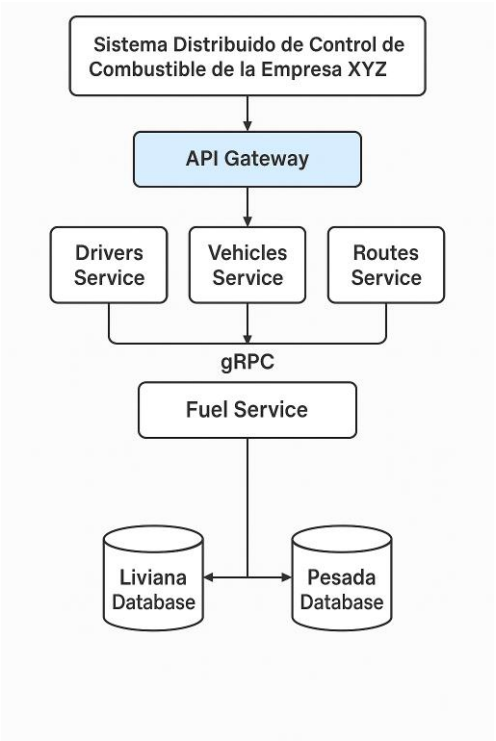
Monitoreo y Escalabilidad

- Logging con Serilog / Elastic Stack
- Escalado horizontal por microservicio
- Orquestación futura con Kubernetes

Tecnologías Utilizadas

- Backend: .NET 8 con gRPC
- Comunicación: gRPC + Protocol Buffers
- Base de Datos: SQL Server / MongoDB
- Contenedores: Docker
- Orquestación: Kubernetes (opcional)

Sistema Distribuido de Control de Combustible - Empresa XYZ



1. Vista General de la Arquitectura

plaintext Copiar Editar

```
            +----------------+
            |  API Gateway   |  <-- (REST opcional)
            +--------+-------+
                     |
      +--------------+---------------+
      |              |               |
+-----v-----+   +-----v-----+   +-----v-----+
|  Drivers  |   | Vehicles  |   |  Routes   |
|  Service  |   |  Service  |   |  Service  |
+-----+-----+   +-----+-----+   +-----+-----+
      |              |               |
      +--------------+---------------+
                     |
              +------v------+
              |  FuelService|
              +------+------+
                     |
              +------v------+
              |  DB (Fuel)  |
              +-------------+
```

Todos los servicios se comunican entre sí mediante **gRPC**.

2. Estilo Arquitectónico: Microservicios
Servicios independientes desplegables y escalables por separado.
Comunicación mediante gRPC (más eficiente que REST).
Bases de datos desacopladas por servicio (opcionalmente compartidas si se requiere integridad cruzada).
Separación por dominios de negocio: **choferes**, **vehículos**, **rutas**, **combustible**.

3. Capas de Cada Microservicio

Cada microservicio tendrá esta estructura:

Sistema Distribuido de Control de Combustible - Empresa XYZ

4. Diseño Modular por Dominio

🔶 Choferes (DriversService)
Registrar choferes.

Consultar disponibilidad.

Asignar choferes por tipo de maquinaria.

🔶 Vehículos (VehiclesService)
Clasificación: liviano o pesado.

Estado operativo del vehículo.

Asociación con choferes y rutas.

🔶 Rutas (RoutesService)
Definir rutas con distancias.

Asociar rutas con vehículos y choferes.

Calcular consumo estimado.

🔶 Combustible (FuelService)
Registrar consumo real por ruta.

Reportes por tipo de maquinaria.

Comparación entre consumo estimado y real.

5. Separación por Maquinaria
En los servicios de vehículos y combustible se añade un campo tipo:

```
public enum TipoMaquinaria {
   Liviana,
   Pesada
}
```

Esto permite que el comportamiento del sistema pueda adaptarse dinámicamente según el tipo de vehículo.

6. Seguridad y Autenticación
Autenticación con JWT.
Roles definidos: Admin, Operador, Supervisor.
Autorización por endpoints en gRPC (con Interceptors o API Gateway si se expone REST).

7. Monitoreo y Escalabilidad
Logging centralizado: Serilog / Elastic Stack.
Escalado horizontal por microservicio.
Orquestación futura: Kubernetes.

Sistema Distribuido de Control de Combustible - Empresa XYZ

8. Repositorios y Desarrollo

Estructura recomendada:

/src
  /Services
    /XYZ.DriversService
    /XYZ.VehiclesService
    /XYZ.RoutesService
    /XYZ.FuelService
  /Protos
  /Shared
  /Gateway (opcional)
  /Infrastructure (docker-compose, DBs)