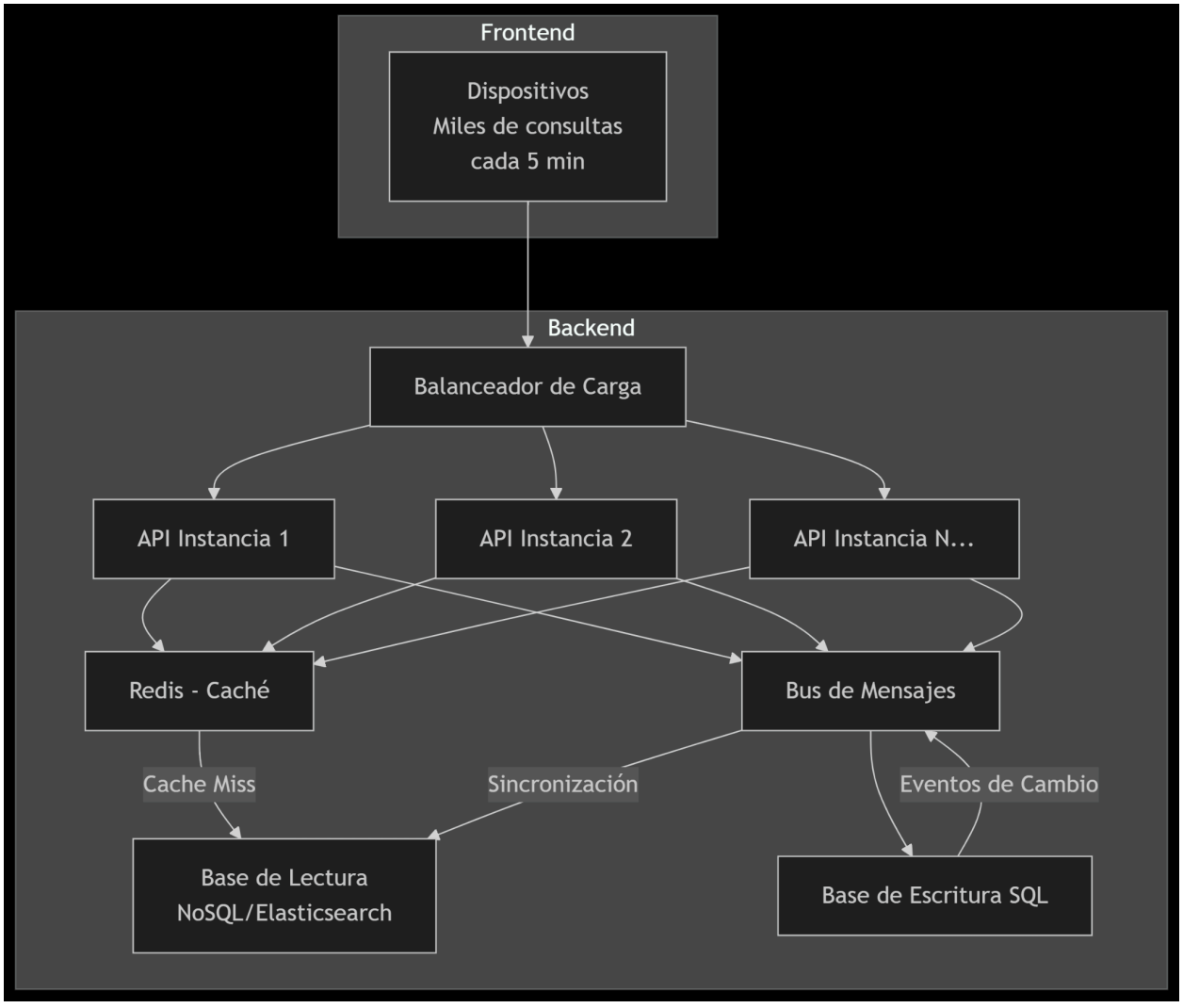
Diseño de API Escalable para Miles de Dispositivos

Introducción

Este documento presenta una solución arquitectónica para diseñar una API capaz de responder eficientemente a miles de dispositivos que realizan consultas periódicas, manteniendo baja latencia y alta disponibilidad. El diseño se centra en estrategias comprobadas para manejar carga masiva de lecturas concurrentes.

Arquitectura Propuesta: CQRS + Escalabilidad Horizontal

Diagrama de Componentes



Componentes Clave

1. **Separación CQRS (Command Query Responsibility Segregation)**
    - **Commands (Escrituras):**

        - Operaciones que modifican estado (POST/PUT/DELETE)
        - Dirigidas a base de datos transaccional (SQL)
        - Ejemplo: Actualizar inventario de productos
    - **Queries (Lecturas):**
        - Operaciones de consulta (GET)
        - Servidas desde base de datos optimizada para lectura
        - Ejemplo: Consultar catálogo de productos
    - **Ventajas:**
        - Optimización independiente de operaciones
        - Escalabilidad selectiva según carga operacional
        - Aislamiento de fallos
2. **Bases de Datos Especializadas**

Tipo	Propósito	Tecnologías	Ventajas
Escritura	Operaciones transaccionales	SQL Server, PostgreSQL	Consistencia ACID, Integridad de datos
Lectura	Consultas de alto rendimiento	MongoDB, Cassandra, Elasticsearch	Escalabilidad horizontal, Baja latencia
Caché	Datos de acceso frecuente	Redis, Memcached	Latencia < 5ms, Alto throughput

3. **Mecanismos de Optimización**
    - **Caché Distribuido (Redis):**
        - Almacena respuestas de consultas frecuentes
        - Estrategia de invalidación basada en eventos
        - Reducción de carga en bases de datos principal
    - **Paginación Eficiente:**

```
public class PagedResponse<T>
{
    public List<T> Items { get; set; }
    public int PageNumber { get; set; }
    public int PageSize { get; set; }
    public int TotalPages { get; }
    public int TotalCount { get; }
}
```

        - Limita resultados por petición

        - Reduce transferencia de datos en red
    - **Balanceo de Carga:**
        - Distribución equitativa de tráfico
        - Health checks automáticos
        - Escalado automático basado en métricas

Flujo de Solicitudes

- **Consulta (GET):**
    - Dispositivo → Balanceador → Instancia API
    - API consulta caché (Redis)
    - Si existe (cache hit): Respuesta inmediata
    - Si no existe (cache miss): Consulta base de lectura → Almacena en caché → Respuesta
- **Escritura (POST/PUT/DELETE):**
    - Dispositivo → Balanceador → Instancia API
    - API envía comando a bus de mensajes
    - Base de escritura procesa comando
    - Evento de cambio propaga actualización a base de lectura y caché

Ventajas y Desafíos

Componente	Pros	Contras
CQRS	Escalabilidad independiente, Optimización específica	Complejidad de implementación, Consistencia eventual
Bases de Datos Múltiples	Alto rendimiento en lecturas, Flexibilidad tecnológica	Sincronización compleja, Mayor costo operativo
Caché Distribuido	Latencia mínima (<5ms), Reduce carga BD	Invalidación compleja, Memoria volátil
Balanceo de Carga	Alta disponibilidad, Escalado automático	Punto único de fallo (si no es redundante)

Métricas de Rendimiento Esperadas

Escenario	Dispositivos Concurrentes	Latencia Promedio	Disponibilidad

Sin optimización	1,000	500-800 ms	95-98%
Arquitectura propuesta	5,000	50-100 ms	99.95%
Arquitectura propuesta	10,000+	80-150 ms	99.90%

Implementación Recomendada

Fase Inicial:

- Implementar CQRS en código existente
- Configurar Redis para caché
- Habilitar balanceador de carga

Fase de Escalamiento:

- Migrar lecturas a base NoSQL
- Implementar bus de mensajes (RabbitMQ/Kafka)
- Configurar autoescalado en nube

Monitoreo Continuo:

- Métricas clave: Latencia, Throughput, Tasa de error
- Alertas para: Saturación caché, Colas de mensajes
- Pruebas de carga periódicas

Conclusión

La arquitectura propuesta permite manejar miles de dispositivos concurrentes mediante:

- Separación clara de operaciones de lectura/escritura
- Bases de datos especializadas para cada tipo de carga
- Mecanismos de caché y paginación eficientes
- Escalabilidad horizontal con balanceo de carga

Esta solución garantiza baja latencia (<100ms) y alta disponibilidad (99.95%) incluso bajo cargas masivas de consultas periódicas.