🏗️ Blog Microservices System Architecture

🌐 Overview

Hệ thống blog được xây dựng theo kiến trúc microservices hiện đại, sử dụng:

- TypeScript + ExpressJS cho backend services
- API Gateway với Consul service discovery
- PostgreSQL cho User và Post services
- Redis cho caching trong Feed service
- Kafka cho event-driven communication
- Next.js + TypeScript cho Frontend
- Docker + Docker Compose cho containerization
- PM2 cho process management (development)

Gồm 4 thành phần chính:

- API Gateway (Express + Consul)
- User Service (TypeScript + Express + PostgreSQL)
- Post Service (TypeScript + Express + PostgreSQL)
- Feed Service (TypeScript + Express + Redis)
- Frontend (Next.js + TypeScript + Tailwind CSS)

🧱 High-Level Components

1. API Gateway (Express + Consul)

- Single entry point cho toàn bộ hệ thống
- Service discovery sử dụng Consul
- Routing tự động đến các microservices
- Health check và monitoring
- CORS và security headers

📌 Ports:

- Main API: `8080`
- Health check: `8081`

- Admin: `9876`

📌 Endpoints:

- `/users/*` → User Service
- `/posts/*` → Post Service
- `/feed/*` → Feed Service

---

2. User Service

- Authentication (register, login) với JWT
- User management và profile
- Follow relationships giữa users
- Database: PostgreSQL với Prisma ORM
- Service discovery với Consul

📌 Endpoints:

- `POST /register` - Đăng ký user mới
- `POST /login` - Đăng nhập
- `GET /users/{id}` - Lấy thông tin user
- `GET /users/{id}/following` - Danh sách following
- `POST /users/{id}/follow` - Follow user khác

📌 Port: `3001` (Docker), `3001-3010` (PM2 cluster)

---

3. Post Service

- CRUD operations cho blog posts
- Author information caching từ User Service
- Kafka integration để nhận user updates
- Database: PostgreSQL với Prisma ORM
- Cache username trong `user_reference` table

📌 Endpoints:

- `POST /posts` - Tạo bài viết mới
- `GET /posts` - Lấy danh sách posts
- `GET /posts?user_ids=[]` - Posts theo user IDs

📌 Port: `3002` (Docker), `3002-3011` (PM2 cluster)

---

4. Feed Service

- Personalized feed generation
- Redis caching cho performance
- Integration với User và Post services
- Real-time feed updates

📌 Endpoints:

- `GET /feed` - Lấy personalized feed

📌 Port: `3003` (Docker), `3003-3012` (PM2 cluster)

---

5. Frontend (Next.js)

- Modern React application với TypeScript
- Responsive UI với Tailwind CSS
- Component library với shadcn/ui
- Authentication với JWT tokens
- Real-time updates với Context API

📌 Features:

- User registration/login
- Create và view posts
- Personalized feed
- User profiles và following
- Dark/light mode toggle

📌 Port: `3000`

🔄 Data Flow & Architecture Patterns

Service Discovery Flow

1. Services **khởi động** → tự động register với Consul
2. API Gateway → discover services thông qua Consul
3. Health checks → continuous monitoring

4. Load balancing → automatic failover

Authentication Flow

1. User **đăng ký/đăng nhập** → User Service tạo JWT token
2. Client **gửi** requests → API Gateway validate JWT
3. Authorized requests → forward đến target services
4. Services → trust API Gateway (internal communication)

Content Creation Flow

1. User **tạo** post → Post Service lưu vào PostgreSQL
2. Cache username → từ User Service vào `user_reference`
3. Kafka event → notify về post mới (nếu cần)
4. Feed invalidation → clear Redis cache cho followers

Feed Generation Flow

1. User request feed → Feed Service
2. Get following list → call User Service
3. Get posts → call Post Service với user_ids
4. Cache result → store trong Redis với TTL
5. Return cached feed → subsequent requests

Event-Driven Updates

1. User update profile → User Service emit Kafka event
2. Post Service listen → update `user_reference` cache
3. Feed Service invalidate → clear related cache entries

🗃 Database Design

🧑 User Service

users

ColumnTypeNote		
id	UUID (PK)	
username	VARCHAR(50)	unique, not null
email	VARCHAR(100)	unique, not null

password_hash	TEXT	
created_at	TIMESTAMP	default now()

follows

ColumnTypeNote		
id	UUID (PK)	
follower_id	UUID	FK → users(id)
followed_id	UUID	FK → users(id)
followed_at	TIMESTAMP	default now()
Unique(follower_id, followed_id)		

✍️ Post Service

posts

ColumnTypeNote		
id	UUID (PK)	
author_id	UUID	
username	VARCHAR(50)	cached
title	VARCHAR(255)	
content	TEXT	
created_at	TIMESTAMP	default now()

user_reference

ColumnTypeNote		
user_id	UUID (PK)	
username	VARCHAR(50)	cached
updated_at	TIMESTAMP	default now()

📰 Feed Service Cache (Redis)

Redis Keys:

- `feed:{user_id}` → JSON array of posts cho user
- `user:{user_id}:following` → Set of following user IDs

- `post:{post_id}` → Cached post data

TTL Policies:

- Feed cache: 5 phút
- Following cache: 30 phút
- Post cache: 1 giờ

🔌 Integration & Communication

Service Communication Patterns

1. Synchronous REST API Calls

- API Gateway ↔ All Services (HTTP routing)
- Feed Service → User Service (get following list)
- Feed Service → Post Service (get posts by user IDs)
- Post Service → User Service (get username for caching)

2. Asynchronous Event-Driven (Kafka)

- User Service → Post Service: `user.updated` events
- Post Service → Feed Service: `post.created` events (future)

3. Service Discovery (Consul)

- All services register themselves với Consul
- API Gateway dynamically discover service endpoints
- Health checking và automatic failover

Integration Technologies

API Gateway Configuration:

```
serviceEndpoints:
  userService: 'http://user-service:3001'
  postService: 'http://post-service:3002'
  feedService: 'http://feed-service:3003'
```

Service Registration:

- Consul agent trên mỗi service
- Auto-registration on startup
- Health check endpoints: `/health`

Authentication Flow:

- JWT tokens issued by User Service
- API Gateway validates tokens
- Services trust API Gateway (no re-validation)

🔧 Technology Stack

Backend Infrastructure

Component	Technology	Purpose
API Gateway	Express.js + Consul	Service discovery, routing
Microservices	TypeScript + Express.js	Business logic
Databases	PostgreSQL + Prisma ORM	Data persistence
Cache	Redis	Feed caching, session storage
Message Queue	Apache Kafka	Event-driven communication
Service Discovery	HashiCorp Consul	Service registry, health checks
Authentication	JWT	Stateless authentication

Frontend Stack

Component	Technology	Purpose
Framework	Next.js 14 + TypeScript	React-based web app
Styling	Tailwind CSS	Utility-first CSS
UI Components	shadcn/ui	Modern component library
State Management	React Context API	Global state
HTTP Client	Fetch API	API communication

DevOps & Infrastructure

Component	Technology	Purpose
Containerization	Docker + Docker Compose	Service orchestration
Process Management	PM2	Development clustering
Development	ts-node-dev	Hot reloading
Testing	Jest	Unit testing
Database Migration	Prisma Migrate	Schema management

Development Tools

Component	Technology	Purpose
Language	TypeScript	Type safety
Package Manager	npm	Dependency management
Code Quality	ESLint + Prettier	Code formatting
API Documentation	OpenAPI/Swagger	API specification
Health Monitoring	Custom endpoints	Service health checks

🗂 Project Structure

```
blog-microservices/
├── 🌐 api-gateway/              # API Gateway với Consul service discovery
│   ├── config/
│   │   ├── gateway.config.yml   # Gateway routing configuration
│   │   └── system.config.yml    # System configuration
│   ├── consul-service.js        # Consul integration
│   ├── server.js               # Main gateway server
│   ├── entrypoint.sh           # Docker startup script
│   └── logs/                   # Gateway logs
│
├── 👤 user-service/             # User management microservice
│   ├── src/
│   │   ├── index.ts            # Service entry point
│   │   ├── controllers/        # REST API controllers
│   │   ├── services/           # Business logic
│   │   ├── routes/             # Express routes
│   │   ├── middleware/         # Authentication, validation
│   │   └── types/              # TypeScript type definitions
│   ├── prisma/
│   │   └── schema.prisma       # Database schema
│   ├── scripts/                # Database initialization
│   └── logs/                   # Service logs
│
├── 📝 post-service/             # Blog post microservice
│   ├── src/
│   │   ├── index.ts            # Service entry point
│   │   ├── controllers/        # Post CRUD controllers
│   │   ├── services/           # Post business logic
│   │   ├── routes/             # Express routes
│   │   ├── middleware/         # Kafka, caching middleware
│   │   └── types/              # TypeScript definitions
│   ├── prisma/
│   │   └── schema.prisma       # Posts & user_reference tables
│   ├── scripts/                # Database setup
│   └── logs/                   # Service logs
│
├── 📰 feed-service/             # Content feed microservice
│   ├── src/
```

```
│   │   ├── index.ts            # Service entry point
│   │   ├── controllers/        # Feed generation
│   │   ├── services/           # Redis caching, API calls
│   │   ├── routes/             # Feed endpoints
│   │   ├── middleware/         # Cache middleware
│   │   └── types/              # TypeScript definitions
│   ├── scripts/                # Redis setup scripts
│   └── logs/                   # Service logs
│
├── 💻 Frontend/                  # Next.js web application
│   ├── app/                    # Next.js 14 app router
│   │   ├── layout.tsx          # Root layout
│   │   ├── page.tsx            # Home page
│   │   ├── login/              # Authentication pages
│   │   ├── register/
│   │   ├── create-post/        # Post creation
│   │   ├── post/[id]/          # Post detail pages
│   │   └── profile/            # User profiles
│   ├── components/             # React components
│   │   ├── AuthForm.tsx        # Authentication forms
│   │   ├── PostCard.tsx        # Post display
│   │   ├── FeedList.tsx        # Feed rendering
│   │   ├── Header.tsx          # Navigation
│   │   └── ui/                 # shadcn/ui components
│   ├── contexts/               # React Context providers
│   ├── hooks/                  # Custom React hooks
│   ├── lib/                    # Utility functions
│   └── public/                 # Static assets
│
├── 🔧 script/                   # PowerShell deployment scripts
│   ├── start-with-pm2.ps1      # Production PM2 start
│   ├── start-with-pm2-dev.ps1  # Development PM2 start
│   ├── stop-pm2.ps1            # Stop all services
│   ├── reset-and-start.ps1     # Full system reset
│   └── consul-status.ps1       # Check Consul status
│
├── 🐳 Docker Files
│   ├── docker-compose.yml      # Full system deployment
│   ├── docker-compose.infrastructure.yml  # Infrastructure only
│   └── init-scripts/           # Database initialization
│
├── ⚙️ Configuration
│   ├── ecosystem.config.js     # PM2 production config
│   ├── ecosystem.dev.config.js # PM2 development config
│   ├── package.json            # Root package.json
│   └── README.md               # Project documentation
│
├── 📚 libs/
│   └── common/                 # Shared libraries
│       ├── types/              # Common TypeScript types
│       ├── events/             # Kafka event definitions
│       └── utils/              # Shared utilities
│
├── 📋 logs/                     # Centralized logging
└── 📚 docs/                     # Documentation
```

```
    ├── blog_microservices_architecture.md
    └── openapi_blog_microservices.json
```

Deployment Options

🐳 Docker Deployment (Recommended)

```
docker-compose up -d    # Full system với infrastructure
```

⚡ PM2 Development

```
# Infrastructure only
docker-compose -f docker-compose.infrastructure.yml up -d

# Services với PM2
.\script\start-with-pm2-dev.ps1  # Development mode
.\script\start-with-pm2.ps1      # Production mode
```

🚀 Deployment Guide

System Requirements

- Docker Desktop (recommended) hoặc Docker Engine 20+
- Node.js 18+ (cho local development)
- PostgreSQL 13+ (nếu không dùng Docker)
- Redis 6+ (nếu không dùng Docker)
- RAM: Minimum 4GB, Recommended 8GB+
- Storage: Minimum 2GB free space

Environment Variables

Global Configuration (.env)

```
# JWT Authentication
JWT_SECRET=your-super-secret-jwt-key-change-in-production

# Service Discovery
CONSUL_HOST=consul
CONSUL_PORT=8500

# Database URLs
USER_DB_URL=postgresql://postgres:password@postgres-user:5432/user_service
POST_DB_URL=postgresql://postgres:password@postgres-post:5433/post_service

# Redis Cache
REDIS_URL=redis://redis:6379

# Kafka Configuration
KAFKA_BROKERS=kafka:9092
KAFKA_CLIENT_ID=blog-microservices
```

Quick Start Commands

1. Full Docker Deployment

```
# Clone repository
git clone https://github.com/quockhanh41/blog-microservice.git
cd blog-microservices

# Start all services
docker-compose up -d

# Check service status
docker-compose ps

# Follow logs
docker-compose logs -f
```

2. Development **với** PM2

```
# Start infrastructure only
docker-compose -f docker-compose.infrastructure.yml up -d

# Install dependencies
npm install

# Start microservices with PM2
.\script\start-with-pm2-dev.ps1

# Monitor services
pm2 monit
```

Service Endpoints & Health Checks

Service	Main Port	Health Check	Status
API Gateway	8080	`http://localhost:8081/health`	✅
User Service	3001	`http://localhost:3001/health`	✅
Post Service	3002	`http://localhost:3002/health`	✅
Feed Service	3003	`http://localhost:3003/health`	✅
Frontend	3000	`http://localhost:3000`	✅
Consul UI	8500	`http://localhost:8500/ui`	✅

📊 Monitoring & Observability

Service Discovery Dashboard

Consul UI: `http://localhost:8500/ui`

- Service registration status

- Health check results
- Service discovery configuration
- Key-value store management

Application Monitoring

PM2 Dashboard (Development)

```
pm2 monit          # Real-time monitoring
pm2 list           # Process list
pm2 logs           # All service logs
pm2 logs api-gateway  # Specific service logs
```

Docker Monitoring

```
docker-compose ps              # Service status
docker-compose logs -f         # Follow all logs
docker-compose logs user-service  # Specific service
docker stats                   # Resource usage
```

Health Check Endpoints

Mỗi service có health endpoint cho monitoring:

```
# API Gateway
curl http://localhost:8081/health
# Response: {"status":"healthy","timestamp":"2024-01-15T10:30:00Z"}

# Individual Services
curl http://localhost:3001/health  # User Service
curl http://localhost:3002/health  # Post Service
curl http://localhost:3003/health  # Feed Service
```

Performance Monitoring

Redis Cache Monitoring

```
# Connect to Redis container
docker exec -it blog-redis redis-cli

# Check cache statistics
INFO stats
MONITOR  # Watch real-time commands
```

Future Enhancements

Planned Features

- [ ] API Rate Limiting với Redis
- [ ] Distributed Tracing với Jaeger/Zipkin
- [ ] Centralized Logging với ELK Stack
- [ ] Message Queue UI với Kafka Manager

- [ ] Database Replication cho high availability
- [ ] API Versioning strategy
- [ ] GraphQL Gateway alternative
- [ ] WebSocket Support cho real-time features

Scalability Improvements

- [ ] Horizontal scaling với Kubernetes
- [ ] Database sharding strategy
- [ ] CDN Integration cho static assets
- [ ] Caching strategy optimization
- [ ] Load testing với Artillery/K6

Security Enhancements

- [ ] OAuth2/OpenID Connect integration
- [ ] API Security với OWASP guidelines
- [ ] Secret Management với Vault
- [ ] SSL/TLS certificates
- [ ] Security scanning trong CI/CD
