Lupisori Web Application

A Scalable Node.js Application with Docker and Nginx Load Balancing

Project Overview

- Modern web application built with Node.js and TypeScript
- RESTful API for movie data management
- MongoDB for data persistence
- Containerized with Docker
- Horizontally scaled with multiple instances
- Load balanced with Nginx

Node.js Application Architecture

Core Technologies

- **Node.js**: JavaScript runtime
- **TypeScript**: Type-safe JavaScript
- **Express**: Web framework
- **Mongoose**: MongoDB ODM

Application Structure

- **MVC Architecture**
  - Models: Data schemas and database interaction
  - Controllers: Request handling and response formatting
  - Services: Business logic
  - Routes: API endpoint definitions

Database Integration

MongoDB Connection

- Connection with retry mechanism
- Error handling and logging
- Configurable via environment variables

Data Model

- Comprehensive movie schema
- Type-safe interfaces
- Validation rules
- Timestamps for auditing

Data Operations

Docker Configuration

Application Containerization

- Node.js 18 Alpine base image
- Multi-stage build process
- Dependency management
- Exposed port: 3000

Docker Compose Setup

- Multiple service orchestration
- Environment variable configuration
- Volume mapping for data persistence
- Network configuration

Networking

Docker Network

- Bridge network for container communication
- Service discovery via container names
- Port mapping for external access

Internal Communication

- Node.js instances → MongoDB
- Nginx → Node.js instances

External Access

- Port 81 exposed for web access
- Ports 3001-3003 for direct instance access

Load Balancing with Nginx

Configuration

- Round-robin load balancing
- Upstream server group
- Proxy configuration
- Header forwarding

Request Routing

- API requests → Node.js instances
- Static files → Nginx direct serving
- Health and documentation endpoints

Benefits

Nginx Caching Layer

Cache Configuration

- Dedicated cache zone for movie data
- Cache storage in persistent volume
- Configurable cache timeouts
- Cache status headers

URL-Based Caching

- Cache keys include full URL with parameters
- Different cache times for different response types
- Cache bypass options for fresh data

Performance Benefits

Deployment and Testing

Deployment

```
docker-compose up -d
```

Load Balancer Testing

```
./test-load-balancer.sh
```

Accessing the Application

- Web: http://localhost
- API: http://localhost/api/movies
- Docs: http://localhost/api-docs
- Health: http://localhost/health

Conclusion

Key Strengths

- Scalable architecture
- Resilient design
- Well-documented API
- Modern development practices
- Containerized deployment

Future Enhancements

- Authentication and authorization
- CI/CD pipeline
- Monitoring and alerting
- Cloud deployment