Vehicle Platform Maintenance Manual

Gruppe 09

June 7, 2025

Contents

1 Development Environment

1.1 Prerequisites

- Minikube (for local Kubernetes cluster)
- kubectl (Kubernetes command-line tool)
- Docker (container platform)
- Make (build automation tool)
- Helm (Kubernetes package manager)
- Python 3.8+ (for service development)

2 Frameworks and Libraries

2.1 Core Technologies

- Kubernetes (Container orchestration)
- Docker (Containerization)
- Helm (Package management)
- Kong (API Gateway)
- RabbitMQ (Message broker)

2.2 Service Dependencies

- Flask (Web framework)
- pika (RabbitMQ client)
- SQLite3 (Local data storage)
- logging (System logging)

3 Build Process

3.1 Automated Build System

The system uses a Makefile-based build system with the following key targets:

```
make start                # Start Minikube
make deploy-all           # Deploy all services
make deploy-k8s           # Deploy Kubernetes services
make deploy-docker        # Deploy Docker services
make vehicle-stack-deploy # Deploy vehicle stacks
```

3.2 Service Building

Each service follows a standard build process:

1. Docker image creation with timestamp-based versioning
2. Kubernetes deployment file updates
3. Service deployment to cluster

4 Testing Process

4.1 Testing Framework

The system uses pytest for testing. Tests are automatically run during the Docker build process for each service. The test execution is configured in the Dockerfile:

```
RUN python -m pytest tests/
```

Each service contains its own test suite in the `tests` directory, which is executed during the build process to ensure code quality and functionality.

5 Deployment

5.1 Service Architecture

The system consists of the following microservices:

- Location Tracker
- Emergency Brake
- Data Mock
- Location Sender
- Central Director
- Visor
- Distance Monitor

5.2 Deployment Configuration

5.2.1 Kubernetes Namespaces

The system uses a multi-namespace architecture:

- `backend` namespace - Contains all core services:
  - Location Tracker
  - Central Director
  - Visor
  - Distance Monitor (one instance per vehicle)
  - Message Broker (RabbitMQ)
  - API Gateway (Kong)
- Vehicle-specific namespaces (e.g., `vehicle-1`, `vehicle-2`) - Each contains:
  - Data Mock
  - Emergency Brake
  - Location Sender
  - Distance Monitor

5.2.2 Service Configuration

Each service is configured through:

- Environment variables
- Kubernetes ConfigMaps
- Helm values

6 API Gateway Documentation

6.1 Kong API Gateway

The API Gateway is deployed using Kong with the following configuration:

```
proxy:
  http:
    enabled: true
    servicePort: 80
    containerPort: 80
ingressController:
  installCRDs: false
```

The API documentation can be accessed via Swagger UI at http://127.0.0.1/swagger.html