

EmRoLab

Technische Dokumentation

Projekt "Smart-Car"

Baris Usluoglu, Daniel Shapiro, Steven Haupenthal

Saarbrücken, den 3. Februar 2026

Inhaltsverzeichnis

1 Systemuebersicht	1
1.1 Architekturprinzipien	1
1.2 Technologie-Stack	1
1.3 Netzwerk-Topologie	1
2 Komponenten im Detail	2
2.1 Mosquitto MQTT Broker	2
2.2 Node-RED	2
2.3 InfluxDB 2.x	2
2.4 Grafana	3
2.5 Calendar Webhook Server	3
2.6 Vehicle Sync Service	3
3 Datenfluss	4
3.1 End-to-End Flow	4
3.2 Alert-Workflow	4
3.3 Latenz-Erwartungen	5
4 MQTT-Protokoll	5
4.1 Topic-Struktur	5
4.2 Nachrichtenformate	5
4.3 QoS-Level	6
5 Datenmodell (InfluxDB)	6
5.1 Measurements	6
5.2 Beispiel-Queries (Flux)	7
6 Node-RED Flows	8
6.1 Hauptflows	8
6.2 CSV Parser Logik	8
6.3 HTTP Request Konfiguration	8
6.4 Alert-Regeln Konfiguration	8
7 Grafana Dashboards	9
7.1 Hauptdashboard (Flottenuebersicht)	9
7.2 Detail-Dashboard (Fahrzeug Details)	10
7.3 Dashboard-Provisioning	10
8 ESP32 Integration	10
8.1 Hardware-Setup	10
8.2 LoRa TTN Bridge (loraempfang.py)	11
9 Sicherheit	12
10 Troubleshooting	12
10.1 Haeufige Probleme	12
10.2 Log-Dateien	12
10.3 Nuetzliche Befehle	12

11 API-Referenz	13
11.1 Calendar Webhook API	13
11.2 InfluxDB HTTP API	14
11.3 Grafana API	14
11.4 Node-RED API	14
11.5 Vehicle Sync Service	15
12 Reflektion	15
12.1 Nicht erfüllte Themen	15
13 Anhang	16
13.1 Umgebungsvariablen	16
13.2 Konfigurationsdateien	16
13.3 Google Calendar Integration	16
13.4 Backup und Wiederherstellung	17
13.5 Referenzen	17
13.6 Externe Dokumentationen	17

1 Systemuebersicht

1.1 Architekturprinzipien

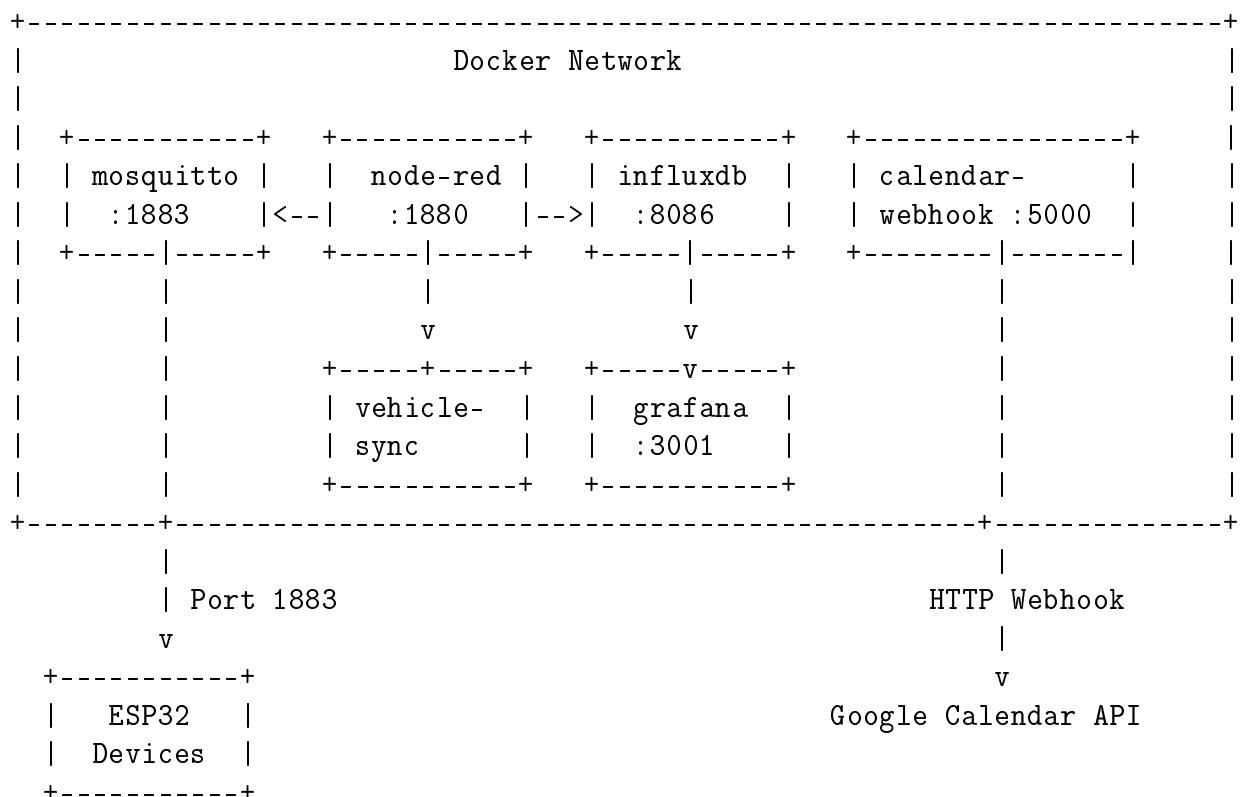
Das Smart-Car System folgt einer ereignisgesteuerten Microservice-Architektur:

- Lose Kopplung: Komponenten kommunizieren ueber MQTT.
- Skalierbarkeit: Jeder Service laeuft in eigenem Container.
- Ausfallsicherheit: Message Queue puffert bei Ausfaellen.
- Erweiterbarkeit: Neue Datenquellen einfach integrierbar.

1.2 Technologie-Stack

Schicht	Technologie	Version	Zweck
Hardware	ESP32	-	Datenerfassung
Transport	MQTT (Mosquitto)	2.x	Nachrichtenuebertragung
Verarbeitung	Node-RED	3.x	ETL und Alarmierung
Speicherung	InfluxDB	2.x	Zeitreihendatenbank
Visualisierung	Grafana	10.x+	Dashboards
Integration	Google Calendar	v3	Kalender-Integration
Backend	Python/Flask	3.11	Calendar Webhook Server
Sync	Python Scripts	3.11	Fahrzeugdaten-Synchronisation
Orchestrierung	Docker Compose	3.8	Container-Management

1.3 Netzwerk-Topologie



2 Komponenten im Detail

2.1 Mosquitto MQTT Broker

Funktion: Zentrale Nachrichtenvermittlung zwischen ESP32 und Node-RED.

Konfiguration (mosquitto/config/mosquitto.conf):

```
# Unverschluesselter Port (nur intern)
listener 1883
protocol mqtt

# Anonyme Verbindungen erlaubt
allow_anonymous true
```

Ports:	Port	Protokoll	Verwendung
	1883	MQTT	Kommunikation (Node-RED / ESP32)

2.2 Node-RED

Funktion: Datenverarbeitung, Transformation und Alarmierung.

Konfiguration (node-red/settings.js):

- Flow-Speicherort: /data/flows.json
- Credentials: /data/flows_cred.json

Flow-Funktionalitaet:

- CSV-Parsing von MQTT-Nachrichten
- Konvertierung zu InfluxDB Line Protocol
- HTTP-basiertes Schreiben nach InfluxDB

2.3 InfluxDB 2.x

Funktion: Speicherung und Abfrage von Zeitreihendaten.

	Parameter	Wert
Konfiguration:	Organisation	vehicle_org
	Bucket	vehicle_data
	Retention	Standard (unbegrenzt)
	Admin-Token	vehicle-admin-token

Wichtige Konzepte:

- Bucket: Container fuer Zeitreihendaten
- Measurement: Aequivalent zu SQL-Tabelle
- Tag: Indexierte Metadaten (z.B. vehicle_id)
- Field: Messwerte (z.B. fuel_l, battery_v)

2.4 Grafana

Funktion: Visualisierung und Alerting.

Provisioning-Struktur:

```
grafana/provisioning/
|-- dashboards/
|   |-- dashboards.yml      # Dashboard-Provider
|   |-- main-dashboard.json # Hauptuebersicht
|   +-- vehicle-detail-dashboard.json # Einzelne Fahrzeugdaten
|-- datasources/
|   +-- datasources.yml     # Datenquellen
+-- alerting/
    |-- alert-rules.yml      # Alert-Regeln
    |-- contact-points.yml   # Benachrichtigungsziele
    +-- notification-policies.yml # Benachrichtigungsrichtlinien
```

2.5 Calendar Webhook Server

Funktion: Google Calendar Integration fuer Alerts und Termine. **Technologie:** Python 3.11 + Flask.

Konfiguration:

- Service Account Key: /config/google-calendar-key.json
- Alerts Config: /config/alerts.json
- Port: 5000

	Endpoint	Methode	Beschreibung
API-Endpunkte:	/health	GET	Health Check
	/event	POST	Kalendereintrag erstellen
	/test	GET	Test-Event erstellen

Anforderungen:

- Google Cloud Service Account mit Calendar API-Berechtigung
- Kalender muss mit Service Account geteilt sein

2.6 Vehicle Sync Service

Funktion: Periodische Synchronisation von Fahrzeugdaten mit InfluxDB. **Technologie:** Python 3.11.

Konfiguration (config/vehicles.json):

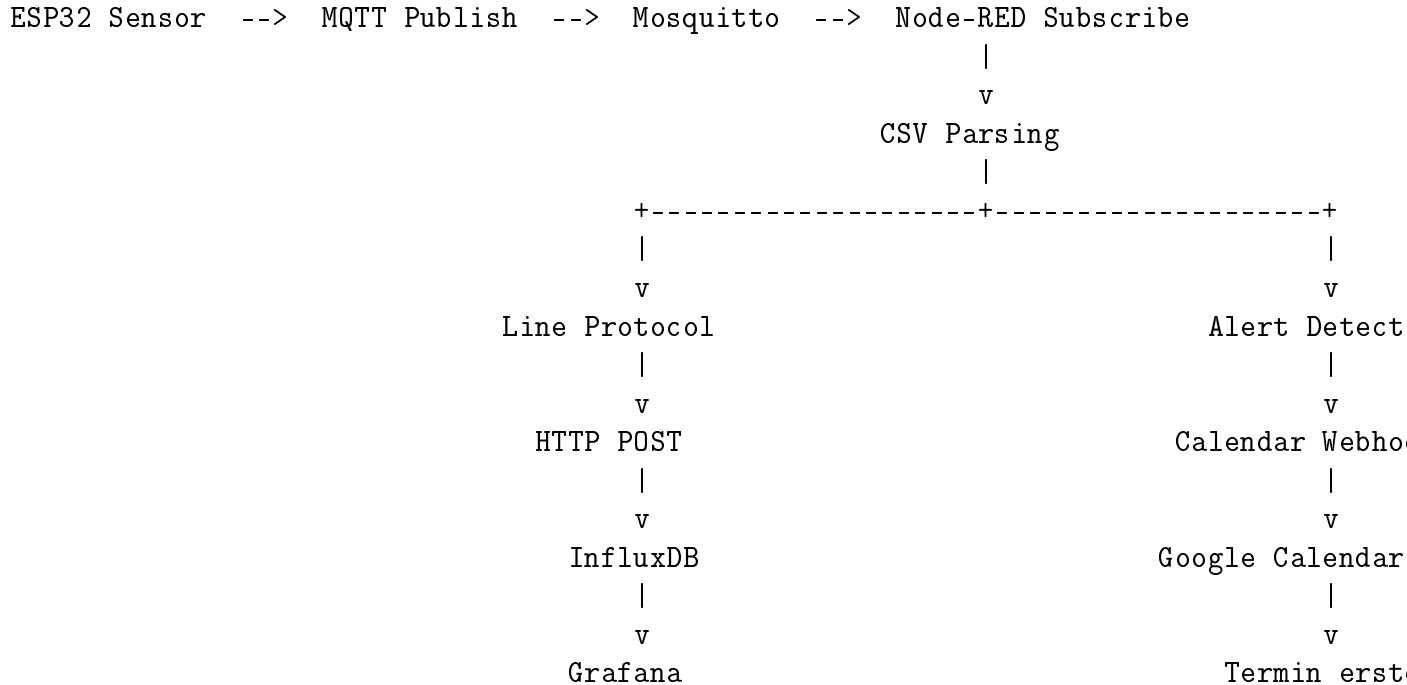
```
{
  "vehicles": [
    {
      "id": "F001",
      "name": "Fahrzeug 1",
      "fuel_capacity": 60.0,
      "battery_nominal": 12.0
    }
  ]
}
```

Funktionalitaet:

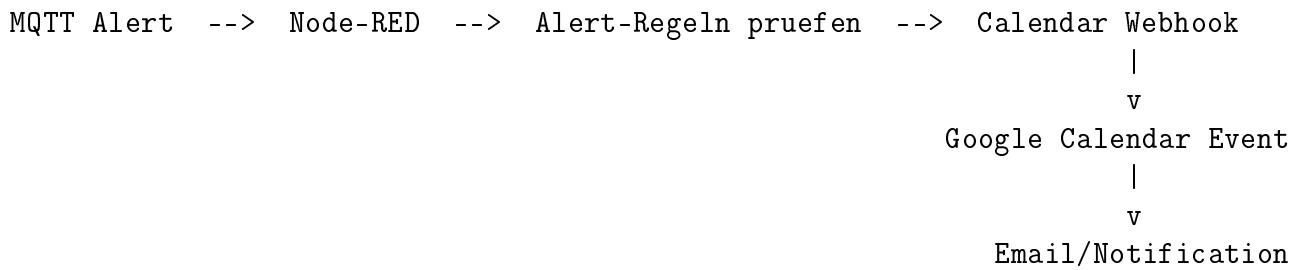
- Liest Fahrzeugkonfiguration aus JSON
- Schreibt initiale Daten in InfluxDB
- Stellt sicher, dass Fahrzeuge in InfluxDB vorhanden sind

3 Datenfluss

3.1 End-to-End Flow



3.2 Alert-Workflow



3.3 Latenz-Erwartungen

Strecke	Erwartete Latenz
ESP32 -> Mosquitto	10-50ms (WLAN)
Mosquitto -> Node-RED	< 5ms
Node-RED -> InfluxDB	5-20ms
Node-RED -> Calendar Webhook	10-50ms
Calendar Webhook -> Google API	100-500ms
InfluxDB -> Grafana	50-200ms (Query)
Gesamt (Telemetrie)	< 300ms
Gesamt (Alarm mit Kalender)	< 600ms

4 MQTT-Protokoll

4.1 Topic-Struktur

smartcar/{vehicle_id}

Alle Nachrichtentypen werden auf diesem Topic publiziert.

4.2 Nachrichtenformate

Fahrzeugstatus (state)

state,{vehicle_id},{state_name},{fuel_l},{battery_v}

Feld	Typ	Beschreibung	Beispiel
vehicle_id	String	Fahrzeug-Kennung	CAR001
state_name	String	Fahrzeugzustand	driving, parked
fuel_l	Float	Kraftstoff in Litern	45.5
battery_v	Float	Batteriespannung	12.8

Fahrt-Zusammenfassung (trip)

trip,{vehicle_id},{trip_id},{duration_s},{fuel_used},{max_acc},{max_brake}

Feld	Typ	Beschreibung
trip_id	String	Eindeutige Fahrt-ID
duration_s	Integer	Fahrtzeit in Sekunden
fuel_used	Float	Verbrauchter Kraftstoff
max_acc	Float	Maximale Beschleunigung
max_brake	Float	Maximale Bremsung

Fehler (error)

error,{vehicle_id},{error_code},{active}

Feld	Typ	Beschreibung
error_code	String	OBD-II Fehlercode (z.B. P0420)
active	Integer	1 = aktiv, 0 = gelöst

GPS-Position (gps)

gps,{vehicle_id},{latitude},{longitude},{speed_kmh}

Feld	Typ	Beschreibung
latitude	Float	Breitengrad
longitude	Float	Laengengrad
speed_kmh	Float	Geschwindigkeit in km/h

Alarm (alert)

alert,{vehicle_id},{alert_type},{message}

Feld	Typ	Beschreibung
alert_type	String	Alarm-Typ (z.B. fuel_low)
message	String	Alarm-Nachricht

4.3 QoS-Level

Level	Beschreibung	Empfehlung
QoS 0	At most once	Nicht empfohlen
QoS 1	At least once	Standard fuer Telemetrie
QoS 2	Exactly once	Empfohlen fuer Fehler

5 Datenmodell (InfluxDB)

5.1 Measurements

vehicle_state

Measurement: vehicle_state

Tags:

- vehicle_id (String)
- state (String)

Fields:

- fuel_l (Float)
- battery_v (Float)
- online (Integer)

trip_summary

Measurement: trip_summary

Tags:

- vehicle_id (String)
- trip_id (String)

Fields:

- duration_s (Integer)
- fuel_used (Float)
- max_acceleration (Float)
- max_braking (Float)

vehicle_errors

Measurement: vehicle_errors

Tags:

- vehicle_id (String)
- error_code (String)

Fields:

- active (Integer)

vehicle_gps

Measurement: vehicle_gps

Tags:

- vehicle_id (String)

Fields:

- latitude (Float)
- longitude (Float)
- speed_kmh (Float)

alerts

Measurement: alerts

Tags:

- vehicle_id (String)
- alert_type (String)

Fields:

- message (String)

5.2 Beispiel-Queries (Flux)

Letzte Fahrzeugzustaende:

```
from(bucket: "vehicle_data")
|> range(start: -1h)
|> filter(fn: (r) => r._measurement == "vehicle_state")
|> filter(fn: (r) => r.vehicle_id == "CAR001")
|> last()
```

Aktive Fahrzeuge zaehlen:

```
from(bucket: "vehicle_data")
|> range(start: -1h)
|> filter(fn: (r) => r._measurement == "vehicle_state")
|> filter(fn: (r) => r._field == "online")
|> group(columns: ["vehicle_id"])
|> last()
|> group()
|> count()
```

Aktive Fehler:

```
from(bucket: "vehicle_data")
|> range(start: -24h)
|> filter(fn: (r) => r._measurement == "vehicle_errors")
|> filter(fn: (r) => r._field == "active")
|> group(columns: ["vehicle_id", "error_code"])
|> last()
|> filter(fn: (r) => r._value == 1)
```

6 Node-RED Flows

6.1 Hauptflows

Flow 1: MQTT --> InfluxDB

[MQTT In] --> [Function: CSV zu Line Protocol] --> [HTTP Request: InfluxDB] --> [Debug]

Flow 2: Alert Detection --> Google Calendar

[MQTT In] --> [Function: CSV Parser] --> [Switch: Alert Filter] --> [Function: Event Trigger]

Flow 3: Vehicle Sync Trigger

[Inject: On Start] --> [HTTP Request: vehicle-sync] --> [Debug]

6.2 CSV Parser Logik

- Topic wird per / getrennt, Fahrzeug-ID aus Teil 2 oder UNKNOWN.
- Payload wird getrimmt und per Komma gesplittet.
- Erster Wert bestimmt den Datentyp, wird in Kleinbuchstaben konvertiert.
- Zeitstempel in Nanosekunden: Date.now() * 1000000.

6.3 HTTP Request Konfiguration

InfluxDB Write URL: http://influxdb:8086/api/v2/write?org=vehicle_org&bucket=vehicle
Methode: POST, Header: Authorization Token.

Calendar Webhook URL: <http://calendar-webhook:5000/event>, Methode: POST,
Header: Content-Type: application/json. Body:

```
{
  "summary": "[ALARM] Fahrzeug F001 - Kraftstoff niedrig",
  "description": "Kraftstoffstand unter 10 Liter. Fahrzeug: F001",
  "duration_minutes": 30,
  "colorId": "11"
}
```

6.4 Alert-Regeln Konfiguration

Datei: config/alerts.json

```
{
  "google_calendar": {
    "enabled": true,
    "calendar_id": "iotwssmartcar@gmail.com"
  },
  "alerts": {
    "fuel_low": {
      "enabled": true,
      "threshold": 10.0,
      "severity": "HOCH",
      "calendar_duration": 30
    },
    "battery_low": {
      "enabled": true,
      "threshold": 11.5,
      "severity": "MITTEL",
      "calendar_duration": 15
    },
    "error_detected": {
      "enabled": true,
      "severity": "KRITISCH",
      "calendar_duration": 60
    }
  }
}
```

7 Grafana Dashboards

7.1 Hauptdashboard (Flottenuebersicht)

UID: smart-car-main.

- **Aktive Fahrzeuge (Stat):** Zeigt Anzahl der aktiven Fahrzeuge
- **Aktive Fehler (Stat):** Anzahl nicht gelöster Fehler
- **Fahrten 7 Tage (Stat):** Anzahl der Fahrten in den letzten 7 Tagen
- **Fehler 7 Tage (Stat):** Anzahl der Fehler in den letzten 7 Tagen
- **Flottenstatus (Stat):** Gesamtstatus der Flotte basierend auf kritischen Werten
- **Kraftstoff niedrig (Stat):** Anzahl Fahrzeuge mit niedrigem Kraftstoff
- **Batterie niedrig (Stat):** Anzahl Fahrzeuge mit niedriger Batterie
- **Heutige Termine (Table):** Tabelle mit den anliegenden Aufgaben des aktuellen Tages
- **Letzte Fahrten (Table):** Tabelle der letzten Fahrten aller Fahrzeuge
- **Geschwindigkeitsverlauf (Time Series):** Zeitlicher Verlauf der Fahrzeuggeschwindigkeiten eines Trips

7.2 Detail-Dashboard (Fahrzeug Details)

UID: smart-car-detail.

- **Fahrzeug-ID (Dropdown Menue):** Anzeige der ausgewählten Fahrzeug-ID
- **Status (Stat):** Aktueller Fahrzeugstatus (Fahren, Parken)
- **Kraftstoff (Gauge):** Aktueller Kraftstoffstand in Litern
- **Batterie (Gauge):** Aktuelle Batteriespannung in Volt
- **Fahrten 7 Tage (Stat):** Anzahl der Fahrten in den letzten 7 Tagen
- **Kraftstoffverlauf (Time Series):** Zeitlicher Verlauf des Kraftstoffstands
- **Batterieverlauf (Time Series):** Zeitlicher Verlauf der Batteriespannung
- **Letzte Fahrten (Table):** Tabelle der letzten Fahrten des Fahrzeugs
- **Fahrzeugfehler (Table):** Tabelle der Fehler des Fahrzeugs

7.3 Dashboard-Provisioning

`dashboards.yml`

- Die Datei dient zur automatischen Bereitstellung von Dashboards in Grafana.
- Mit apiVersion: 1 wird festgelegt, welche Version der Provisioning-Schnittstelle verwendet wird.
- Unter „providers“ wird definiert, woher Grafana die Dashboards laden soll.
- Es wird ein Anbieter mit dem Namen „default“ angelegt.
- Die Dashboards werden in einem Ordner mit dem Namen „Smart-Car“ gespeichert und angezeigt.
- Der Typ „file“ bedeutet, dass die Dashboards aus Dateien im Dateisystem geladen werden.
- Unter „options“ wird der Speicherort der Dashboard-Dateien angegeben.
- Der Pfad /etc/grafana/provisioning/dashboards gibt an, in welchem Verzeichnis die JSON-Dashboard-Dateien liegen.
- Beim Start von Grafana werden alle Dashboards aus diesem Ordner automatisch eingelesen.
- Dadurch müssen Dashboards nicht manuell im Webinterface importiert werden, sondern stehen direkt zur Verfügung.

8 ESP32 Integration

8.1 Hardware-Setup

WLAN-Modul

Standard in ESP32 (802.11 b/g/n, 2.4 GHz).

LoRa-Modul (extern)

Empfohlene Module: SX1276 / SX1278, Frequenz 868 MHz (EU) / 915 MHz (US), Reichweite bis 10 km.

	LoRa Pin	ESP32 Pin
Pinbelegung:	VCC	3.3V
	GND	GND
	SCK	GPIO 18
	MISO	GPIO 19
	MOSI	GPIO 23
	NSS	GPIO 5
	RST	GPIO 14
	DIO0	GPIO 26

8.2 LoRa TTN Bridge (loraempfang.py)

Die TTN Bridge empfaengt Daten von The Things Network und schreibt sie direkt nach InfluxDB.

Starten:

```
python esp32/lora_sender/loraempfang.py
```

Konfiguration:

Parameter	Wert
TTN Broker	eu1.cloud.thethings.network
TTN Port	1883
InfluxDB URL	http://localhost:8086/

Payload-Format (4 Bytes minimal):

[Vehicle_ID, Message_Type, Value1, Value2]

Message Types:

ID	Typ	Bedeutung
1	MSG_STATUS	Fahrstatus (Fuel, Battery10)
2	MSG_TRIP_END	Fahrt beendet (Duration_min, FuelUsed10)
3	MSG_ERROR	Fehlermeldung (ErrorCode, Battery10)
4	MSG_IDLE	Fahrzeug steht (Fuel, Battery10)

Vehicle ID Mapping:	ID	Fahrzeugname
	1	Ford_Siesta

Error Code Mapping:	Code	Bedeutung
	1	BATTERY_LOW
	2	CHECK_ENGINE
	3	LOW_OIL
	4	TIRE_PRESSURE

9 Sicherheit

Siehe urspruengliche Dokumentation. (Abschnitt 9 wurde im Ausgangstext genannt, aber ohne Detailinhalte.)

10 Troubleshooting

10.1 Häufige Probleme

Container startet nicht

```
# Logs pruefen  
docker-compose logs <service>  
  
# Container neu starten  
docker-compose restart <service>
```

MQTT-Verbindung fehlgeschlagen

```
# Mosquitto-Status pruefen  
docker exec mosquitto mosquitto_sub -t '$SYS/#' -C 1  
  
# Verbindung testen  
docker exec mosquitto mosquitto_pub -t test -m "hello"
```

Keine Daten in InfluxDB

1. Node-RED Debug-Panel pruefen
2. InfluxDB-Verbindung in Node-RED testen
3. Bucket-Berechtigung pruefen

Grafana zeigt keine Daten

1. Datenquelle testen (Data Sources -> Test)
2. Query im Explore-Modus testen
3. Zeitbereich pruefen

10.2 Log-Dateien

Service	Log-Zugang
Mosquitto	docker logs mosquitto oder ./mosquitto/log/
Node-RED	docker logs node-red
InfluxDB	docker logs influxdb
Grafana	docker logs grafana

10.3 Nützliche Befehle

```
# Alle Container neustarten
docker-compose down && docker-compose up -d

# Container-Status
docker-compose ps

# In Container einloggen
docker exec -it <container> sh

# Netzwerk pruefen
docker network inspect smartcar-network

# Ressourcenverbrauch
docker stats
```

11 API-Referenz

11.1 Calendar Webhook API

Base URL: <http://calendar-webhook:5000> (intern) oder <http://localhost:5000> (extern).

Health Check

```
curl http://localhost:5000/health
```

Response:

```
{
  "status": "ok",
  "google_api": true
}
```

Event erstellen

```
curl -X POST http://localhost:5000/event \
  -H "Content-Type:application/json" \
  -d '{
    "summary": "Test Event",
    "description": "Test-Beschreibung",
    "duration_minutes": 30
  }'
```

Erfolgs-Response:

```
{
  "success": true,
  "event_id": "abc123...",
  "link": "https://calendar.google.com/..."
}
```

Fehler-Response:

```
{  
    "success": false,  
    "error": "Fehlerbeschreibung"  
}
```

Test-Event

```
curl http://localhost:5000/test
```

11.2 InfluxDB HTTP API

Health Check:

```
curl http://localhost:8086/health
```

Query (Flux):

```
curl -X POST http://localhost:8086/api/v2/query \  
  -H "Authorization:Token vehicle-admin-token" \  
  -H "Content-Type:application/vnd.flux" \  
  -d 'from(bucket:"vehicle_data")>range(start:-1h)'
```

Write (Line Protocol):

```
curl -X POST "http://localhost:8086/api/v2/write?org=vehicle_org&bucket=  
  vehicle_data&precision=ns" \  
  -H "Authorization:Token vehicle-admin-token" \  
  -H "Content-Type:text/plain" \  
  -d 'vehicle_state,vehicle_id=CAR001,state=idle,fuel_l=45.5,battery_v=12.8,  
  online=1i'
```

11.3 Grafana API

Dashboards auflisten:

```
curl -u admin:admin http://localhost:3001/api/search
```

Dashboard exportieren:

```
curl -u admin:admin http://localhost:3001/api/dashboards/uid/smart-car-main
```

11.4 Node-RED API

Flows exportieren:

```
curl http://localhost:1880/flows
```

11.5 Vehicle Sync Service

Fahrzeuge synchronisieren:

```
docker exec vehicle-sync python /app/sync_vehicles.py
```

Oder via HTTP:

```
curl -X POST http://localhost:8080/sync
```

12 Reflektion

Das Smart-Car Monitoring System wurde in den angegebenen Punkten erfolgreich implementiert und bietet eine robuste Plattform zur Überwachung und Verwaltung von Fahrzeugdaten in Echtzeit. Die Nutzung von MQTT für die Datenübertragung gewährleistet eine effiziente Kommunikation zwischen den ESP32-Geräten und dem Backend. Node-RED ermöglicht eine flexible Datenverarbeitung und Integration mit InfluxDB, die als leistungsfähige Zeitreihendatenbank dient. Grafana bietet ansprechende Visualisierungen und Alarmierungsfunktionen, die den Überblick über die Fahrzeugflotte erleichtern.

Im Laufe des Projektes mussten wir allerdings feststellen, dass einige geplante Funktionen nicht vollständig umgesetzt werden konnten. Grund dafür waren Zeitbeschränkungen und technische Herausforderungen, die den Umfang des Projekts beeinflusst haben. Trotz dieser Einschränkungen konnten wir ein funktionierendes System entwickeln, das die Kernanforderungen erfüllt und eine solide Grundlage für zukünftige Erweiterungen bietet.

Insgesamt sind wir mit dem Ergebnis zufrieden und sehen großes Potenzial für weitere Verbesserungen und Ergänzungen in zukünftigen Iterationen des Projekts.

12.1 Nicht erfüllte Themen

- Auslesen der Daten mittels OBD-II Adapter/CAN Bus
- Energieverwaltung
- Fahrverhalten
- Wetter Routine
- Gyroskop Anbindung
- Sicherheitsaspekte

13 Anhang

13.1 Umgebungsvariablen

Variable	Dienst	Beschreibung
DOCKER_INFLUXDB_INIT_USERNAME	InfluxDB	Admin-Benutzer
DOCKER_INFLUXDB_INIT_PASSWORD	InfluxDB	Admin-Passwort
DOCKER_INFLUXDB_INIT_ORG	InfluxDB	Organisation
DOCKER_INFLUXDB_INIT_BUCKET	InfluxDB	Standard-Bucket
DOCKER_INFLUXDB_INIT_ADMIN_TOKEN	InfluxDB	API-Token
GF_SECURITY_ADMIN_PASSWORD	Grafana	Admin-Passwort
GOOGLE_KEY_FILE	calendar-webhook	Service Account Key Pfad
ALERTS_FILE	calendar-webhook	Alert-Konfiguration Pfad

13.2 Konfigurationsdateien

Datei	Beschreibung
config/alerts.json	Alert-Regeln und Calendar-Konfiguration
config/vehicles.json	Fahrzeugdaten und -konfiguration
config/google-calendar-key.json	Google Service Account Key
docker-compose.yml	Container-Orchestrierung
node-red/flows.json	Node-RED Flows
node-red/flows_cred.json	Verschluesselte Credentials
mosquitto/config/mosquitto.conf	MQTT Broker Konfiguration

13.3 Google Calendar Integration

Service Account Setup

1. Google Cloud Console: <https://console.cloud.google.com>
2. Projekt erstellen/auswaehlen
3. APIs & Services → Library → Google Calendar API aktivieren
4. IAM & Admin → Service Accounts → Service Account erstellen
5. Key erstellen (JSON) als `google-calendar-key.json` speichern
6. Kalender mit Service Account Email teilen (Aenderungen vornehmen und Freigabe verwalten)

Event-Farbcodes	colorId	Farbe	Verwendung
	9	Blau	Standard-Alarme
	6	Orange	HOCH Prioritaet
	11	Rot	KRITISCH Prioritaet
	10	Gruen	Erfolgreiche Aktionen

Troubleshooting Calendar

- Invalid JWT Signature: Key ungültig oder abgelaufen → neuen Key erstellen, Container neu starten (docker restart calendar-webhook).

- Calendar not found: Kalender nicht mit Service Account geteilt → teilen.
- API not enabled: Google Calendar API nicht aktiviert → in Cloud Console aktivieren.

13.4 Backup und Wiederherstellung

InfluxDB Backup

```
# Backup erstellen
docker exec influxdb influx backup /backup -t vehicle-admin-token

# Backup aus Container kopieren
docker cp influxdb:/backup ./influxdb-backup
```

InfluxDB Restore

```
# Backup in Container kopieren
docker cp ./influxdb-backup influxdb:/restore

# Restore durchfuehren
docker exec influxdb influx restore /restore -t vehicle-admin-token
```

Grafana Dashboards Backup

```
# Alle Dashboards exportieren
curl -u admin:admin http://localhost:3001/api/search?type=dash-db | \
jq -r '.[]|.uid' | \
xargs -I {} curl -u admin:admin http://localhost:3001/api/dashboards/uid/{} >
dashboard-{}.json
```

Node-RED Flows Backup

```
# Flows sichern
cp node-red/flows.json node-red/flows_backup_$(date +%Y%m%d).json
cp node-red/flows_cred.json node-red/flows_cred_backup_$(date +%Y%m%d).json
```

13.5 Referenzen

Service	Port (Host)	Port (Container)	Zugriff
Mosquitto (MQTT)	1883	1883	Intern/Extern
Node-RED	1880	1880	Web-UI
InfluxDB	8086	8086	API
Grafana	3001	3000	Web-UI
Calendar Webhook	5000	5000	API (intern)

13.6 Externe Dokumentationen

- InfluxDB 2.x: <https://docs.influxdata.com/influxdb/v2/>
- Grafana: <https://grafana.com/docs/grafana/latest/>
- Node-RED: <https://nodered.org/docs/>
- Eclipse Mosquitto: <https://mosquitto.org/documentation/>
- ESP32 Arduino Core: <https://docs.espressif.com/projects/arduino-esp32/>

- LoRa Library: <https://github.com/sandeepmistry/arduino-LoRa>