Gwinnett County Public Schools Technology and Innovation Division - Bus Monitoring through Kafka

Design Document

Amali McHie

Sarah Fashinasi

Tyler Hood

Jeffrey Sanderson

Alex Baker

Date: (09/22/2024)
Department: CCSE
Course: SWE 4724 (01)
Professor: Yan Huang

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This Design Document details the development of an advanced Real-Time Bus Monitoring System for Gwinnett County Public Schools. The goal is optimizing student safety and operational efficiency by showing real-time updates on bus schedule information.

1.2 Scope

This project will include technologies like Apache Kafka for event streaming, Docker for containerization, and SQL Server for robust data management. At completion, this project will have progressed Gwinnett county's bus monitoring system.

1.3 Overview

This document includes an introduction to the project, an overview of the system and its architectures, design plans of the system and database, and a requirements matrix.

1.4 Definitions and Acronyms

- GCPS - Gwinnett County Public Schools
- SQL- Structured Query Language

2. SYSTEM OVERVIEW

Functionality: The project involves using Samsara Kafka connector to create a Real-Time Bus Monitoring System at Gwinnett County Public Schools (GCPS). All relevant data will be stored using the SQL Server database. Docker will be used for containerization; the system will ensure constant deployment. It pulls the bus location data every 5 seconds, helping make sure that tracking is consistent and reliable.

Context: The project is sponsored by the Technology and Innovation Division of GCPS to address inefficiencies in their current transportation system, creating the ability to track school buses on their routes. The system will help users monitor buses in real-time, improving time efficiency and communication.

Design: The project uses Kafka Connecter to connect to the Samsara real-time bus data with the system. The data is then consumed, processed, and stored using SQL server. Docker ensures seamless deployment and consistency across the districts multiple operating system. We are also focusing on security, ensuring all data is properly encrypted and secure.

Background: The current transportation system in Gwinnett County Public Schools lacks real-time monitoring capabilities, leading to inefficiencies and communication delays. Currently, the school district uses a manual or semi-automated system for tracking bus operations. This initiative will support the district's future growth, with potential for further system expansion.

3. System Architecture

3.1 Architectural Design

Kafka Consumer:

Responsibilities: The Kafka consumer is the heart of the system, responsible for consuming the real-time bus data provided by the Samsara Kafka Connector. It polls data from the Kafka stream every 5 seconds, processing the location and status of each school bus.

Collaboration: The data gets sent to the data processing system/ SQL server database.

SQL Server/Data Processing:

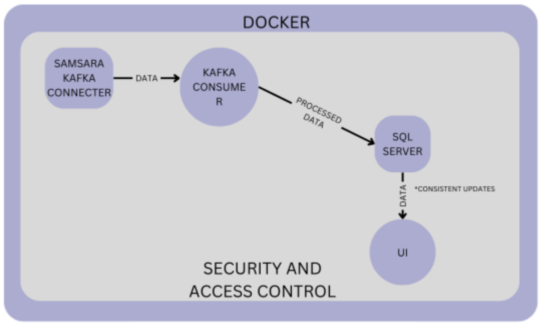
Responsibilities: Receives processed data from Kafka. It then validates and organizes the data before its placed in storage.

Collaboration: The SQL Server stores the data. It is then used by the User Interface to display the stored data through real-time relaying.

User Interface:

Responsibilities: Displays real-time data on bus locations and statuses. This data is displayed through a dashboard for the school district's transportation department. It pulls data from the SQL Server to generate real-time reports and updates.

Collaboration: It retrieves data from the SQL Server and transforms it into trackable Infromation for users. It relies on continuous communication with the Data Processing and Storage subsystem to ensure the user interface is constantly updated with the latest information



The Samsara Kafka Connecter sends data to the Kafka Consumer. The data is then processed and sent to the SQL Server for storage and organization. The Ui then gets constant updates from the SQL Server. The security and access control surround the complete system, showing its impact on the entire system. Docker encapsulates the entire system, encouraging consistency and efficiency amongst the components so they work smoothly.

3.2 Technical design

Kafka Connector- Allows real-time updates from Samsara System, which provides bus location and status updates. The connector works by creating a pipeline that streams the data. The data is utilized by providing status on a consistent basis, updating every 5 seconds.

SQL Server- Is used as a database repository for the project. It is essential for storing data in an organized and efficient manner. The database schema is designed to organize data by bus ID, timestamps, and statuses, helping maintain the real-time update concept.

Docker- Is used in containerizing the application, allowing the system to be consistent regardless of users or deployment environment. The other subsystems have their own container, allowing for isolated updates that impact the entirety of the program.

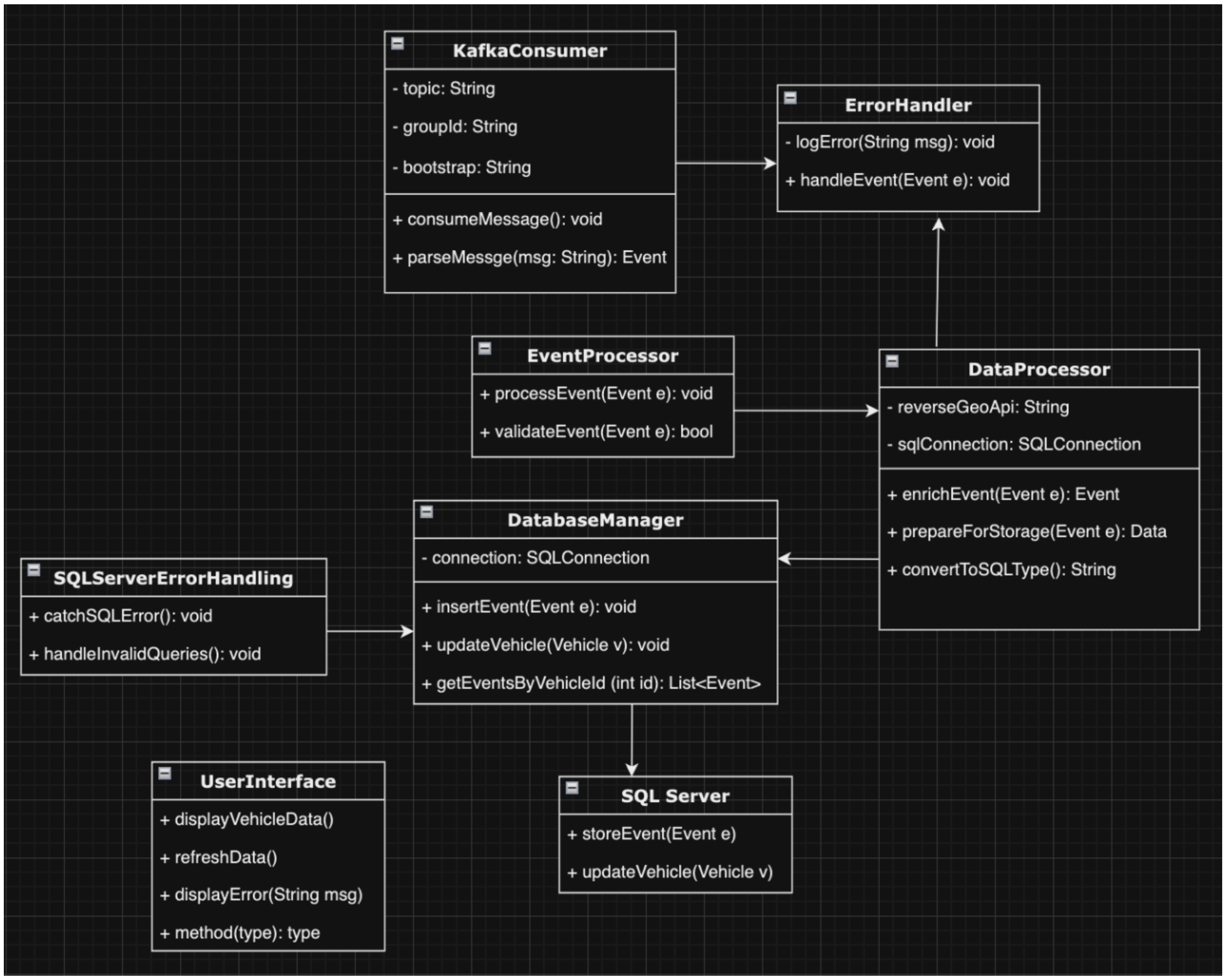
User interface- The Ui allows the data to be displayed in dashboard, making it easy to visualize and use. The UI interacts with the SQL Server to fetch and display data dynamically.

Security and access control- ensures the data is secure and only accessed by an authorized developer or user. Implements encrypted software to ensure that the data is secure.

3.3 Design Rationale

The chosen architecture employs a microservices approach, leveraging Kafka for real-time data streaming, SQL Server for structured storage, and Docker for deployment. This architecture allows for modular development, where each subsystem can be developed, deployed, and scaled independently. The software chosen was also recommended by the sponsor based on previous work done and ease of access from the Technology and Innovation Division of the GCPS. Kafka was chosen because the GCPS already uses the system to poll for information every 5 seconds. SQL Server is used due to the fatality of the software between both parties (developers and the sponsors, GCPS). Docker is used for easy containerization and efficient updates created by developers and approved by the sponsors.

4. DETAILED DESIGN


KafkaConsumer
- topic: String
- groupId: String
- bootstrap: String
+ consumeMessage(): void
+ parseMessge(msg: String): Event
ErrorHandler
- logError(String msg): void
+ handleEvent(Event e): void
EventProcessor
+ processEvent(Event e): void
+ validateEvent(Event e): bool
DataProcessor
- reverseGeoApi: String
- sqlConnection: SQLConnection
+ enrichEvent(Event e): Event
+ prepareForStorage(Event e): Data
+ convertToSQLType(): String
DatabaseManager
- connection: SQLConnection
+ insertEvent(Event e): void
+ updateVehicle(Vehicle v): void
+ getEventsByVehicleId (int id): List<Event>
SQLServerErrorHandling
+ catchSQLError(): void
+ handleInvalidQueries(): void
UserInterface
+ displayVehicleData()
+ refreshData()
+ displayError(String msg)
+ method(type): type
SQL Server
+ storeEvent(Event e)
+ updateVehicle(Vehicle v)

Kafka Consumer: Reads events from Samsara, processes them, and sends them for validation and storage to **EventProcessor**.

EventProcessor: Processes incoming events, ensures events are valid and sends events to either **ErrorHandler** or **DataProcessor.**

Error Handling: Logs any errors, handles invalid events, and works alongside **SQLServerErrorHandling.**

Data Processor: Prepares event data to be stored in the SQL database by mapping to the appropriate format (converts the data into SQL compatible types).

Database Manager: Inserts events into SQL Server, updates vehicle information, retrieves event data by vehicle ID, and sends SQL errors/invalid queries to **SQLServerErrorHandling.**

SQL Server: Inserts valid records into storage and logs rejections using SQL queries. Passes records through **SQL Error Handling** before arriving at **UserInterface.**

(SQL) Error Handling: Catches SQL specific errors and deals with invalid queries. Passes critical

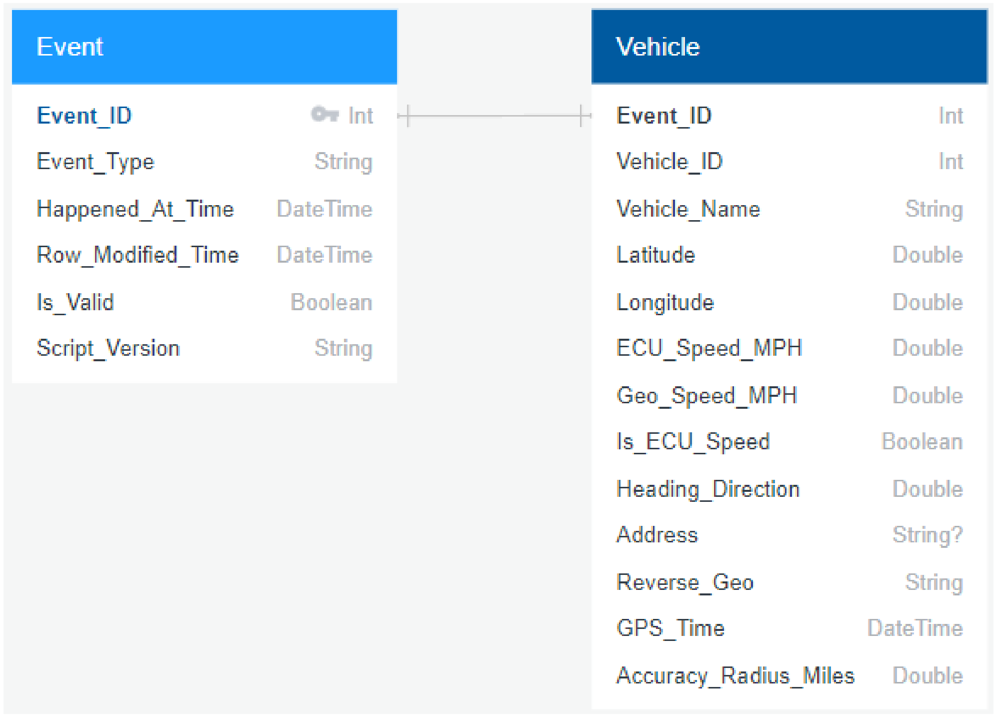
errors to **UserInterface.**

User Interface: Displays processed data, refreshes the UI with updated data, and shows any errors to the users.

5. DATA (DATABASE) DESIGN

The database for this system will be hosted on Micrsoft SQL Server and will accept data from Kafka events. These Kafka events will contain all the data about the events themselves as well as information about the vehicles that Kafka receives from. Due to this split in needs, the database will contain two main tables for the events and the vehicles. This structure strikes a balance between necessary tables and querying speed.

The Event table contains information about the events themselves, like the type of event, validity, time of occurring, and time of insertion into the database. The Vehicle table instead focuses just on the vehicle data gathered, including the location and speed of them at any given moment. The Event_ID links these two tables, allowing for easier identification of issues and potential solutions. As an example, if there is a significant difference between the Happened_At_Time and Row_Modified_Time values for several data points in the Event table, the two tables could be connected to show that one particular vehicle has more latency than others. This would indicate a problem with the tracking hardware on the bus and can be fixed accordingly. The full design is shown below.



Event	
Event_ID	Int
Event_Type	String
Happened_At_Time	DateTime
Row_Modified_Time	DateTime
Is_Valid	Boolean
Script_Version	String

Vehicle	
Event_ID	Int
Vehicle_ID	Int
Vehicle_Name	String
Latitude	Double
Longitude	Double
ECU_Speed_MPH	Double
Geo_Speed_MPH	Double
Is_ECU_Speed	Boolean
Heading_Direction	Double
Address	String?
Reverse_Geo	String
GPS_Time	DateTime
Accuracy_Radius_Miles	Double

Data collected from the Kafka connector will also be processed to find certain values like Reverse_Geo, Address, and Geo_Speed_MPH before being inserted into the database. Geo_Speed_MPH will compare the previous data point of that Vehicle_ID and calculate the MPH of the vehicle from the previous data point to the newest data point. Another example of processing is the Address column, as the latitude and longitude from the collected data will be compared to an existing set of geo fences to find if the bus is at a school and which school that is. The Reverse_Geo value is processed similarly to the Address value, but instead it is compared to a map of all addresses and finds the closest address to the vehicle.

6. HUMAN INTERFACE DESIGN

6.1 UI design

The system will have a simple look and clear buttons to increase the usability. There will be a map able to view to see where a tracked bus is located on route.

6.2 UX design

Users of all ages should find the app intuitive and easy to use.

7. REQUIREMENTS MATRIX

Requirement ID	Description	Design	Development	Testing	Test ID
01	Creation of Kafka Consumer	Done	WIP	WIP	01
02	Kafka connection created and information flows through Validation and error handling into data processor	Done	WIP	WIP	02
03	Creation of data Processor	Done	WIP	WIP	03
04	Data processor successfully reads and writes data into the data base from Kafka consumer	Done	WIP	WIP	04

05	Creation and connection of Microsoft SQL Server	Done	WIP	WIP	05
06	Microsoft SQL Server successfully receives event details	Done	WIP	WIP	06
07	Creation of User Interface	Done	WIP	WIP	07
08	User interface successfully connects to Microsoft SQL Server and accurately displays vehicle information	Done	WIP	WIP	08
09	SQL Error handler Prevents SQL errors from showing in UI	Done	WIP	WIP	09
10	Docker Containerization Allows for the application to be successfully run on multiple devices	Done	WIP	WIP	10
11	Data Validation Ensures Correct data from Kafka Connector	Done	WIP	WIP	11
12	Error handling removes noise and any error reads from the connector	Done	WIP	WIP	12