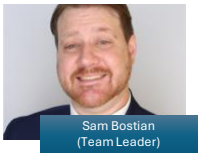
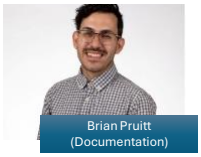
12-T3: GCPS REAL-TIME BUS MONITORING SYSTEM

CAPSTONE INDUSTRY PARTNER PROJECT
CS 4850 - SECTION 01 – FALL 2024
PROFESSOR PERRY – NOVEMBER 13, 2024





Name	Role	Cell Phone / Email
Sam Bostian	Team Leader	404.555.1212 sbostian@students.kennesaw.edu
Michael Rizig	Developer	678.668.3294 mrizig@students.kennesaw.edu
Charlie McLarty	Developer/QA	470.303.9544 cmclarty21@gmail.com
Brian Pruitt	Documentation	404.207.6548 bpruitt9@students.kennesaw.edu
Allen Roman	Developer	aroman14@students.kennesaw.edu

Website:	**https://team3-gwinnettbus.github.io/GwinnettBusWebpage**
Github:	https://github.com/Team3-GwinnettBus/Gwinnett-Bus

Lines of Code	Over 2.5M!
Number of Components/Tools	6 (*Kafka, SQL Server, Podman, React.js, Flask, Leaflet*)
Total Man Hours	Approx. 1250 Hours

TABLE OF CONTENTS

INTRODUCTION

PROJECT OVERVIEW

The GCPS Real-Time Bus Monitoring System was developed to enhance the efficiency of monitoring bus operations for Gwinnett County Public Schools. The project replaces an outdated polling-based system with Kafka-based real-time event streaming, enabling GCPS to access real-time telemetry data for over 2,000 buses.

PROJECT OBJECTIVES

The key objectives of this project are:

1. **Reduce Latency**: Switch from a polling system to Kafka event streaming to improve data freshness and reduce retrieval delays.
2. **Enable Scalable Deployment**: Utilize Podman containerization for easy deployment and scalability across environments.
3. **Improve Data Processing and Storage**: Validate and store bus telemetry data in an SQL Server database for consistent, reliable access.

SCOPE

The scope of this project includes data ingestion, validation, storage, and visualization of bus telemetry data, as well as establishing a deployment pipeline using Podman for container management.

REQUIREMENTS

FUNCTIONAL REQUIREMENTS

Real-Time Data Ingestion: The system must consume real-time telemetry data from Kafka, capturing information like bus location, speed, and heading.

Data Validation: Incoming data must be validated for accuracy before storage.

Data Storage in SQL Server: Validated data should be stored in SQL Server for easy access.

Frontend Visualization: The system should provide a visual map showing bus locations, updated in near real-time.

Deployment Automation: The system must be deployable through Podman containers, ensuring consistent setup across environments.

NON-FUNCTIONAL REQUIREMENTS

Performance: The system must process and display data with minimal latency.

Scalability: The system must handle telemetry data from up to 2,000 buses.

Security: Data should be encrypted and secured, with access limited to authorized users.

CONSTRAINTS

- The system must run on Red Hat Linux environments.
- GCPS is a Microsoft shop and uses Microsoft SQL Server.
- The entire solution should be containerized through Podman.

ANALYSIS

DESIGN

SYSTEM ARCHITECTURE

The system consists of four main components:

1. **Kafka Producer**: Simulates and streams bus telemetry data to Kafka.
2. **Kafka Consumer**: Consumes, validates, and stores data in SQL Server.
3. **SQL Server**: Central storage for validated telemetry data.
4. **Frontend**: A React.js application that visualizes bus locations on a map.

DATA FLOW

Producer generates simulated telemetry data and sends it to Kafka.

Consumer receives the data, performs validation, and stores it in SQL Server.

Frontend queries SQL Server periodically to update the map with the latest bus locations.

CONTAINERIZATION

Using Podman, each component (producer, consumer, frontend) is containerized for consistent and scalable deployment. The pod.yml file configures the network and connects containers within the same pod, enabling seamless communication.

DEVELOPMENT

DEVELOPMENT APPROACH

We used an Agile approach with sprints focused on building core components like the Kafka producer, Kafka consumer, and frontend. Iterative feedback allowed us to refine requirements and implementation details to meet sponsor needs.

SYSTEM COMPONENTS

Kafka Producer: Simulates real-time data and streams it to Kafka.

Kafka Consumer: Validates and stores telemetry data in SQL Server.

Frontend Visualization: Uses React.js and Leaflet to display bus locations.

CONTAINERIZATION

Each component was containerized using Podman, enabling easy deployment and scalability. Docker files were created for each component, specifying dependencies and startup commands.

TEST REPORT

Requirement	Description	Pass	Fail	Severity
Test Requirement	This is a Test Requirement to serve as an example	✓	✗	Low
Real-time data ingestion through Kafka	Verify that the Kafka producer sends data in real time	✓		High
Data validation rules	Check that invalid GPS data is flagged and logged	✓		High

Data storage in SQL Server	Confirm that validated data is stored accurately	✓		High
Error handling for invalid data	Verify that invalid data is stored in separate table	✓		Medium
Container initialization	Check if all containers start without errors	✓		High
Inter-container communication	Confirm Kafka-to-SQL data flow in pod environment	✓		High
Pod shutdown and cleanup	Ensure stop_pod.sh script removes all containers/pods	✓		Medium
Monitoring setup (optional feature)	Verify monitoring tool displays container status	✓		Low
Scalability test with 2,000 bus data points	Check system performance under high data volume	✓		High
System latency under load	Measure if latency remains within acceptable range	✓		High
Data retrieval from SQL Server	Verify that bus location data is retrievable	✓		High
Consistent container deployment across environments	Test pod deployment on different environments	✓		Medium

VERSION CONTROL

SUMMARY

PROJECT ACHIEVEMENTS

- Real-time data ingestion and validation through Kafka.
- Scalable, containerized deployment with Podman.
- Interactive frontend displaying real-time bus locations.

LESSONS LEARNED

Agile development allowed us to adapt quickly to changing requirements.

Containerization with Podman provided a robust and consistent deployment environment.

FUTURE RECOMMENDATIONS

Implement advanced monitoring for better system insights.

Extend the system to support additional data sources or integrate with other school district systems.

APPENDIX

PROJECT PLAN

GANTT CHART

GCPS Bus Monitoring System

Sam Bostian, Michael Rizig, Charlie McLarty, Brian Pruitt

Project Start

TASK	ASSIGNED TO	PROGRESS	START	END
Planning				
Define goals and project scope	ALL	100%	8/12/2024	8/15/2024
Define timeline	Michael Rizig	100%	8/15/2024	8/17/2024
Set Milestone Dates	Sam Bostain	100%	8/17/2024	8/21/2024
Define Resources Required	Charlie McLarty	100%	8/21/2024	8/24/2024
Complete Project Plan	Michael Rizig	100%	8/18/2024	9/1/2024
Design				
Define Dependencies	Sam Bostian	0%	8/12/2024	8/31/2024
Create SDD Document	Brian Pruitt	0%	8/12/2024	8/28/2024
Create SRS Documents	ALL	0%	8/12/2024	8/31/2024
Define Design Elements	ALL	0%	8/12/2024	8/31/2024
Compare/Analyze designs	Sam,Michael,Brian	0%	8/12/2024	9/2/2024
Complete SDD	Sam,Charlie,Brian	0%	8/12/2024	9/2/2024
Complete SRS	ALL	0%	8/12/2024	9/2/2024
Execution				
Create Development Document	Michael Rizig	0%	9/10/2024	9/30/2024
Build Backend	Sam Bostian	0%	9/10/2024	9/30/2024
Establish Database	Charlie McLarty	0%	9/15/2024	9/30/2024
Containerization	Sam,Michael	0%	10/8/2024	10/29/2024
Create Presentation and Demo	Brian Pruitt	0%	10/8/2024	10/29/2024
Test, evaluate, and debug	Charlie McLarty	0%	9/29/2024	10/29/2024
Evaluation				
Create Software Test Plan	Charlie McLarty	0%	9/23/2024	11/27/2024
Clean up Prototype	Sam,Michael,Brian	0%	9/28/2024	11/27/2024
Final debugging	Sam, Charlie	0%	9/28/2024	11/27/2024
Complete Testing Documentation	Michael, Charlie	0%	9/29/2024	11/27/2024
Complete Final report	ALL	0%	9/17/2024	12/2/2024

Milestone	Description	Due Date	Completion
Plan	Project Plan Document	9/1/2024	☐
M1	Initial Development	9/16/2024	☐
M2	Database and Event Lo	10/14/2024	☐
M3	Containerization	10/28/2024	☐
M4	C-Day	11/19/2024	☐
M5	Final Deliverable	12/2/2024	☐

PROJECT SET-UP

INSTALL REQUIRED TOOLS

Podman: Install Podman for containerization. This tool will be used to create, manage, and deploy containers for the project's components.

- Command: On a Red Hat-based system, use **sudo dnf install podman**.

Kafka: Install and configure Kafka for event streaming.

- Ensure that Kafka is correctly configured to create and manage topics that will handle bus telemetry data.

SQL Server: Install and configure Microsoft SQL Server, as it will store validated telemetry data from Kafka.

- Configure SQL Server to allow connections from the Podman containers and set up necessary user credentials.

CONFIGURE ENVIRONMENT

Environment Variables: Set up environment variables as defined in the README.md file.

These include:

- SERVER_IP: IP address of the server hosting Kafka and SQL Server.
- KAFKA_TOPIC: The name of the Kafka topic to which the producer will send data.
- DB_NAME, DB_USER, DB_PASSWORD: Database name and credentials for SQL Server.

Verify that these environment variables are accessible to each container by configuring them in the pod's YAML file or exporting them before deployment.

BUILD AND DEPLOY CONTAINERS

Use the **run_pod.sh** script to build and deploy the containers within a Podman pod.

This script will:

- Build the **producer container** that simulates bus data and sends it to Kafka.
- Build the **consumer container** that consumes data from Kafka and interacts with SQL Server.
- Start these containers within a shared Podman pod network, ensuring they can communicate.

Run **run_pod.sh** from the terminal to initiate this setup

MONITOR AND TEST

Container Status: Confirm that each container is running as expected by checking Podman's status output.

- Command: **podman ps** to view running containers.
- Command: **podman logs <id>** to view logs for running container.

Data Flow Verification:

- Verify that the Kafka producer is successfully sending data to the Kafka topic.
- Confirm that the Kafka consumer is receiving this data and inserting it into SQL Server.
- Run sample API requests to ensure the data can be retrieved and displayed accurately in the web interface.

SHUT DOWN AND CLEAN UP

When finished, use the stop_pod.sh script to stop and remove all containers and pods, freeing up resources and leaving a clean environment for future deployments.

Run **stop_pod.sh** from the terminal:

- This script will:
    a. Stop and remove all containers and pods associated with the project.
    b. Prune any unused resources, such as images, to prevent storage clutter.