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 4](#_Toc182510442)

[Project Overview 4](#_Toc182510443)

[Project Objectives 4](#_Toc182510444)

[Scope 4](#_Toc182510445)

[Requirements 4](#_Toc182510446)

[Functional requirements 4](#_Toc182510447)

[Non-Functional Requirements 5](#_Toc182510448)

[Constraints 5](#_Toc182510449)

[Analysis 5](#_Toc182510450)

[Design 5](#_Toc182510451)

[System Architecture 5](#_Toc182510452)

[Data Flow 5](#_Toc182510453)

[Containerization 6](#_Toc182510454)

[Development 6](#_Toc182510455)

[Development Approach 6](#_Toc182510456)

[System Components 6](#_Toc182510457)

[Containerization 6](#_Toc182510458)

[Test Report 6](#_Toc182510459)

[Version Control 7](#_Toc182510460)

[Summary 7](#_Toc182510461)

[Project Achievements 7](#_Toc182510462)

[Lessons Learned 7](#_Toc182510463)

[Future recommendations 8](#_Toc182510464)

[Appendix 8](#_Toc182510465)

[Project Plan 8](#_Toc182510466)

[Project Set-Up 8](#_Toc182510467)

[Install Required Tools 8](#_Toc182510468)

[Configure Environment 9](#_Toc182510469)

[Build and Deploy Containers 9](#_Toc182510470)

[Monitor and Test 10](#_Toc182510471)

[Shut Down and Clean Up 10](#_Toc182510472)

# 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 The project replaces an outdated polling-based system with Kafka-based real-time event streaming, enabling GCPS to access real-time telemetry data for over 2,000 buses.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 | Checkmark with solid fill | Close with solid fill | **Low** |
| Real-time data ingestion through Kafka | Verify that the Kafka producer sends data in real time | Checkmark with solid fill |  | **High** |
| Data validation rules | Check that invalid GPS data is flagged and logged | Checkmark with solid fill |  | **High** |
| Data storage in SQL Server | Confirm that validated data is stored accurately | Checkmark with solid fill |  | **High** |
| Error handling for invalid data | Verify that invalid data is stored in separate table | Checkmark with solid fill |  | **Medium** |
| Container initialization | Check if all containers start without errors | Checkmark with solid fill |  | **High** |
| Inter-container communication | Confirm Kafka-to-SQL data flow in pod environment | Checkmark with solid fill |  | **High** |
| Pod shutdown and cleanup | Ensure stop\_pod.sh script removes all containers/pods | Checkmark with solid fill |  | **Medium** |
| Monitoring setup (optional feature) | Verify monitoring tool displays container status | Checkmark with solid fill |  | **Low** |
| Scalability test with 2,000 bus data points | Check system performance under high data volume | Checkmark with solid fill |  | **High** |
| System latency under load | Measure if latency remains within acceptable range | Checkmark with solid fill |  | **High** |
| Data retrieval from SQL Server | Verify that bus location data is retrievable | Checkmark with solid fill |  | **High** |
| Consistent container deployment across environments | Test pod deployment on different environments | Checkmark with solid fill |  | **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

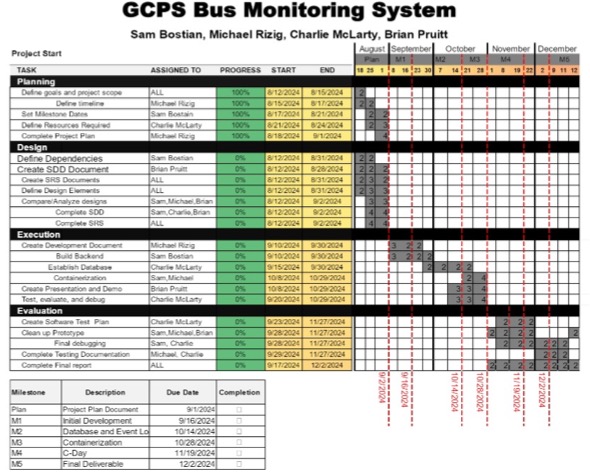
Implement advanced monitoring for better system insights.

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

# Appendix

## Project Plan

### Gantt Chart



## 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:
  1. Stop and remove all containers and pods associated with the project.
  2. Prune any unused resources, such as images, to prevent storage clutter.