12-T3: GCPS REAL-TIME BUS MONITORING SYSTEM

DEVELOPMENT DOCUMENT
CS 4850 - SECTION 01 – FALL 2024
NOVEMBER 10, 2024

Sam Bostian
(Team Leader)

Michael Rizig
(Developer)

Charlie Mclarty
(Developer)

Brian Pruitt
(Documentation)

Allen Roman
(Developer)

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

TABLE OF CONTENTS

EVENT STREAMING ARCHITECTURE

KAFKA PRODUCER

The Kafka producer is responsible for sending telemetry data, simulating GCPS bus data in real time. It streams this data continuously to the Kafka server, enabling near-instantaneous access to location, speed, and heading information for GCPS buses. The producer was configured using Python and Podman to ensure seamless integration with other components.

KAFKA CONSUMER

The Kafka consumer component listens for incoming telemetry data from the Kafka topic. Upon receiving data, the consumer processes it, checking for validity and preparing it for storage in the SQL Server database. This ensures real-time data flow and minimal latency in retrieving information.

DATA VALIDATION AND PROCESSING

Data validation is a critical step to ensure that only accurate data is stored. We implemented rules for GPS coordinates, speed limits, and heading degrees. Invalid data is logged for further analysis, while validated data is passed to the storage layer. This process protects data integrity and supports accurate reporting.

DATA STORAGE AND MANAGEMENT

SQL SERVER DATABASE

The SQL Server database is the primary data repository. Validated telemetry data from the consumer is inserted into the database, where it can be accessed for analysis and reporting. SQL Server's robust querying capabilities allow GCPS to retrieve and analyze data quickly and effectively.

DATA INTEGRITY AND ERROR HANDLING

To maintain data integrity, error handling is implemented to manage anomalies in telemetry data. When data validation fails, the erroneous data is flagged and stored in a separate table for quality control. This dual-storage setup provides a reliable foundation for both active monitoring and post-incident analysis.

CONTAINERIZATION AND DEPLOYMENT

CONTAINER SETUP WITH PODMAN

Each component of the project (Kafka producer, Kafka consumer, and web interface) is containerized using Podman. Containers isolate environments, ensuring consistent operation across different systems. This setup allows for easy scaling and updating of individual components without impacting the rest of the system.

DEPLOYMENT SCRIPTS

Two main scripts manage the container lifecycle:

- **run_pod.sh**: Builds and runs the containers using podman play kube with the pod.yml configuration. This script ensures that each container starts in the correct sequence and configuration.
- **stop_pod.sh**: Stops all containers, removes them, and cleans up system resources. This script ensures a clean teardown after deployment, which prevents resource strain and leftover files.

MONITORING (OPTIONAL)

An optional monitoring feature is included to track container performance. This monitoring can be configured in the README.md, allowing users to view metrics on system health and performance over time.

DOCUMENTATION

README.MD OVERVIEW

The README.md file provides comprehensive guidance for installing, configuring, and using the system. It covers all setup requirements, environment variable configurations, and troubleshooting steps, serving as a go-to resource for developers and administrators.

USAGE AND CONFIGURATION INSTRUCTIONS

The documentation includes step-by-step instructions for running the deployment scripts, modifying container settings, and adjusting pod configurations. This allows future developers or GCPS staff to make adjustments and deploy updates seamlessly.

PROJECT SETUP STEPS

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.