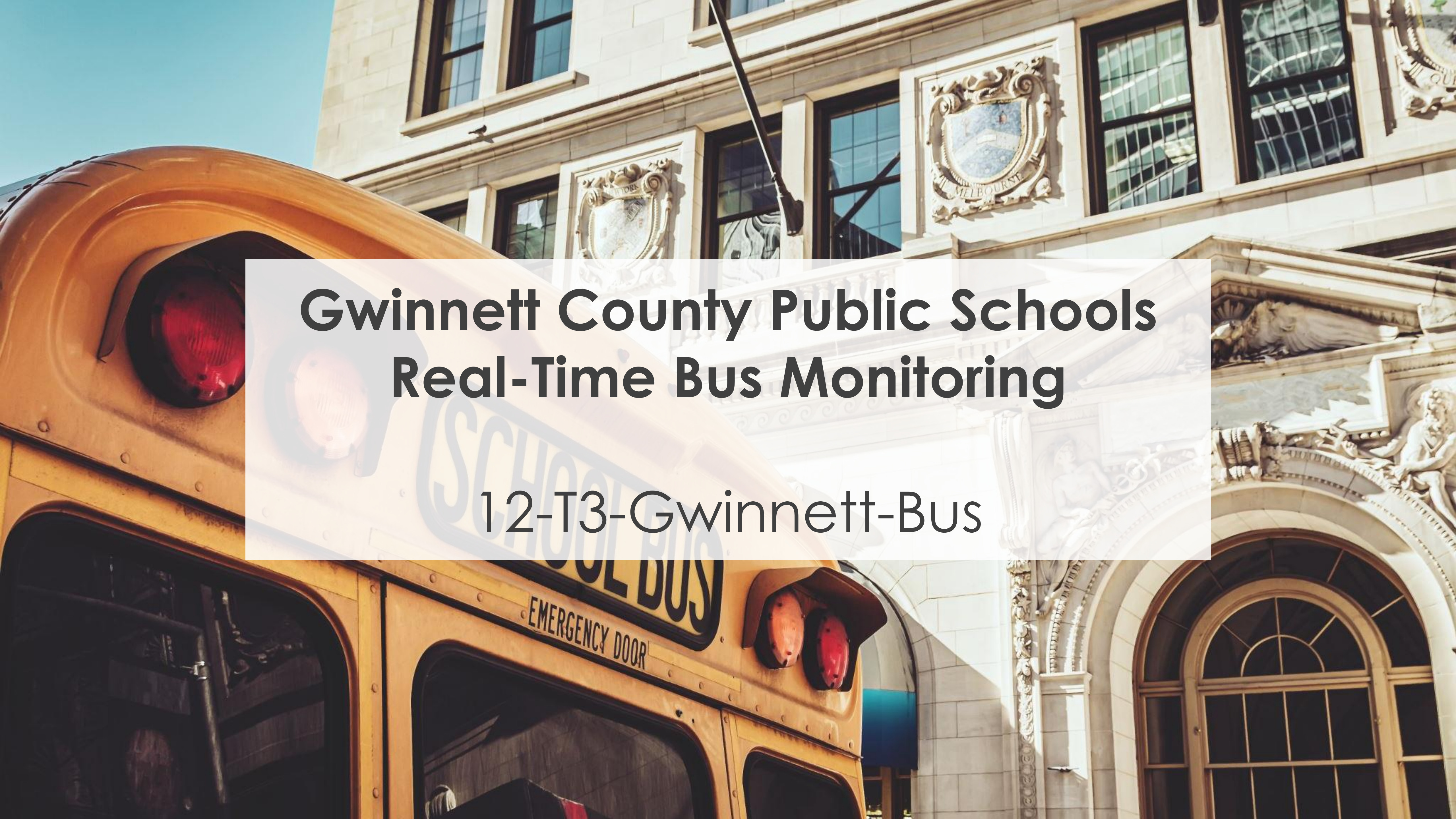


Gwinnett County Public Schools Real-Time Bus Monitoring

12-T3-Gwinnett-Bus

Our Team

Sam Bostian

- Team Leader
- General Help
- RHEL9 Install

Michael Rizig

- Primary Development
- Kafka, Python Dev, MSSQL

Charlie McLarty

- Data Generation
- Data Simulation

Brian Pruitt

- Documentation

Allen Roman

- Containerization
- Deployment

GWINNETT
COUNTY
PUBLIC
SCHOOLS

Sponsor

Gwinnett County Public Schools (GCPS) is the largest school system in Georgia and the 14th largest school district in the US.

Approximately 182,000 students attend one of GCPS's 182 Schools with a fleet of almost 2,000 buses.

School Breakdown:

- 81 Elementary Schools
- 29 Middle Schools
- 24 High Schools
- 7 Specialty Schools
- 1 Charter School

Problem Statement

Currently GCPS uses Samsara REST API calls to poll for bus data approximately every 5 seconds.

Due to the vast bus fleet and how many calls made per minute GCPS makes, they are currently in the top 1% of all of Samsara's API calls.

GCPS would like to switch to using the Samsara Connector and store this bus data into an SQL relational database that can be called by other applications for near real-time data processing.

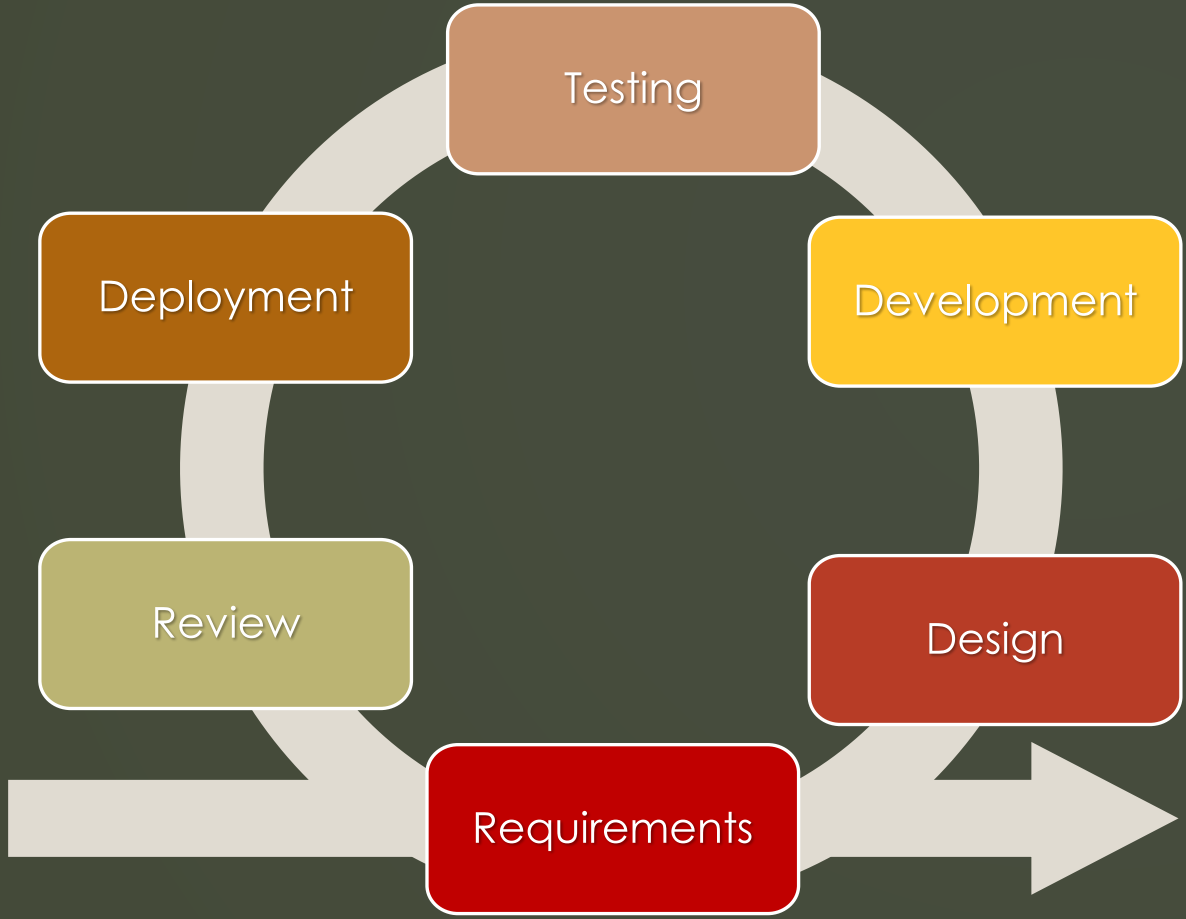
Project Overview

Objective:

- Transition from Samsara API polling to Kafka-based event streaming for real-time bus tracking, creating an in-house solution for efficiency and data security.
- Efficiently process and validate bus telemetry data and store it in SQL Server.

Planning

Planning



- **Agile Parallel Development**
  - Divided work into parallel development streams
- **Meeting Cadence**
  - Industry Sponsor Meetings
  - Team meetings
- **Tools and Collaboration**
  - Version Control
  - Communication Platforms

Key System Components

RHEL9 Server

Integrates Kafka, MSSQL, and all other components into one seamless application.

Kafka Broker

Publisher-Subscriber Paradigm

Bus Simulation

Generates synthetic bus data and streams it to Kafka.

MSSQL Database

Stores validated data; logs anomalies in a separate table.

Data Manager

Handles data validation and manages SQL operations.

Containerization

Ensures consistent deployment across environments.

Development

Tech Stack

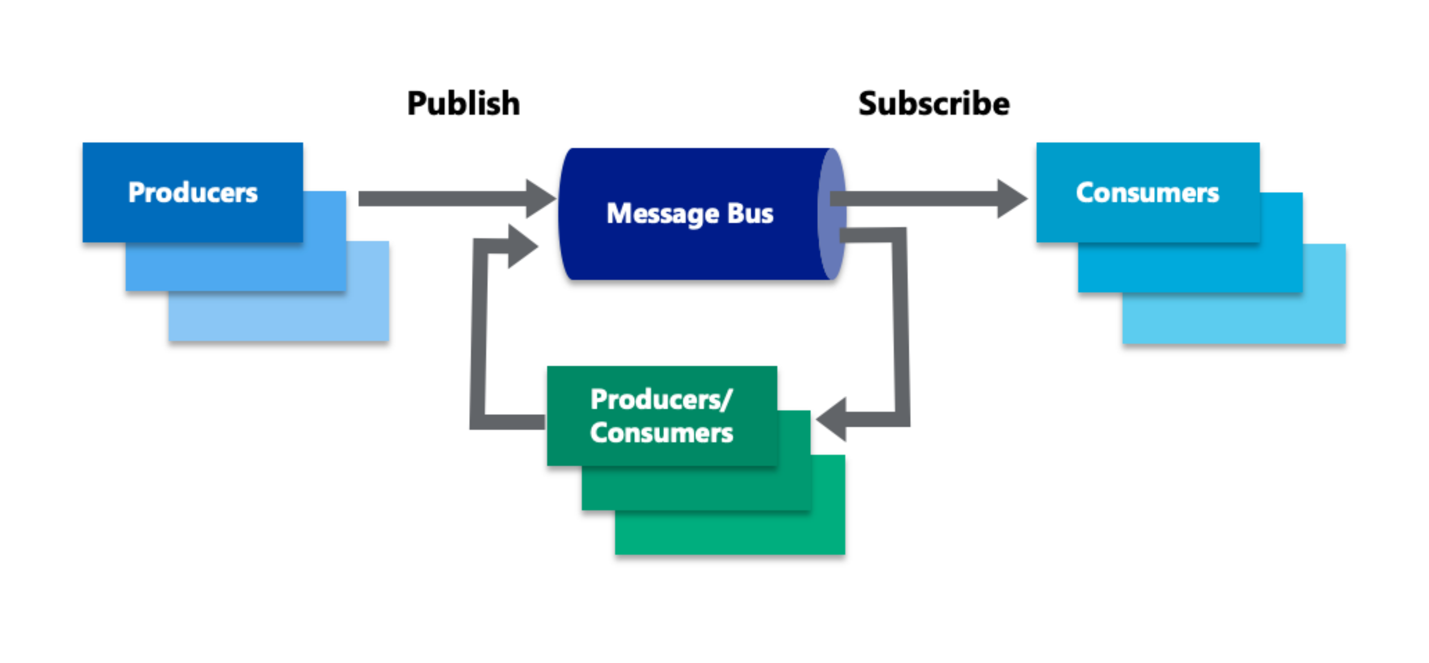
kafka
podman
Flask
Microsoft®
SQL Server®

Kafka Basics

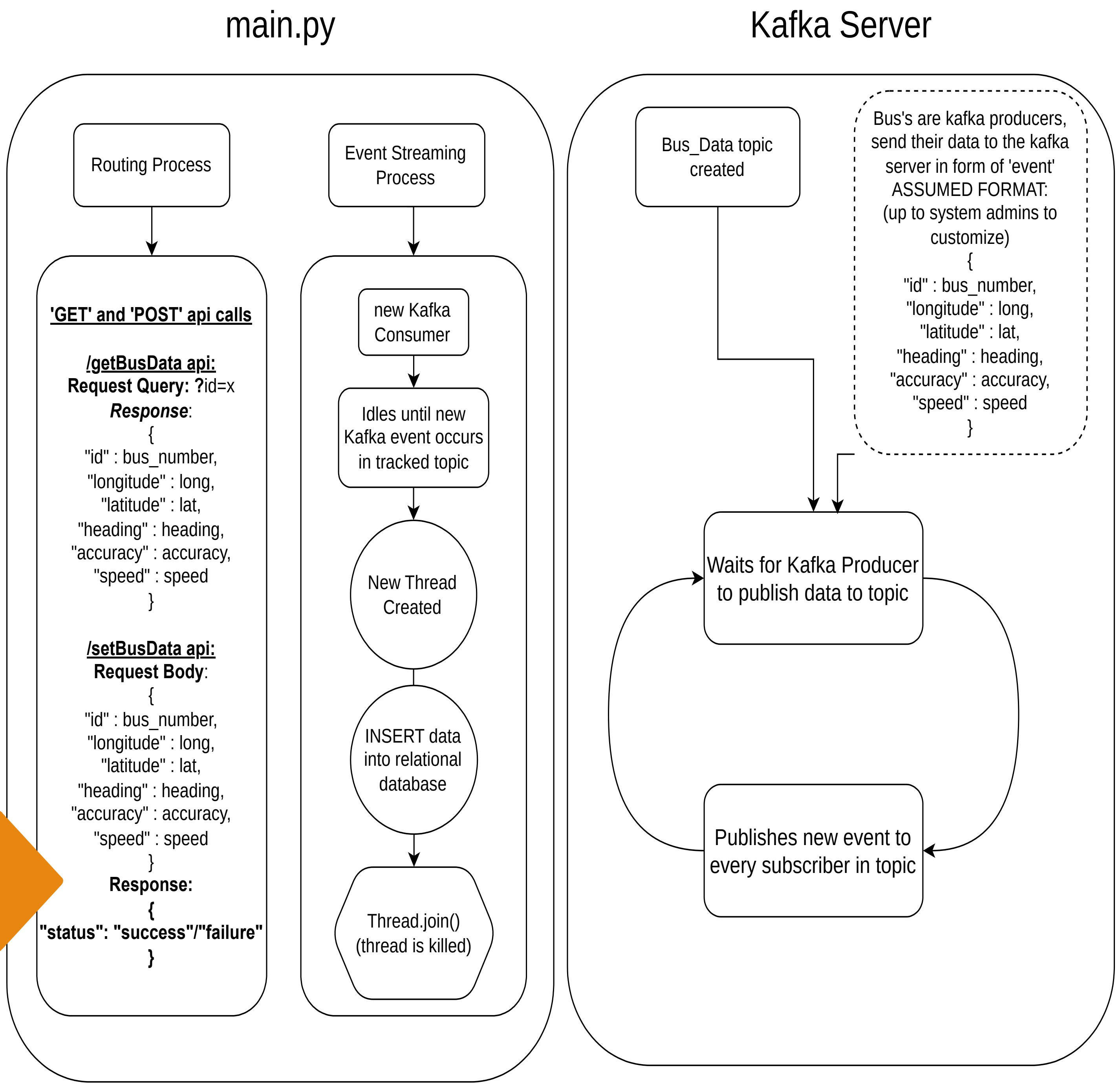
Apache Kafka can be generally described as an implementation of the *Publisher Subscriber Paradigm*.

Publishers (**Producers**) send data to the broker (**Kafka Server**), which distributes the data to all Subscribers (**Consumers**) of that topic.

Ex. Live Streaming



Server process overview



main.py

Routing Process

'GET' and 'POST' api calls

/getBusData api:
Request Query: ?id=x
Response:
{
"id" : bus_number,
"longitude" : long,
"latitude" : lat,
"heading" : heading,
"accuracy" : accuracy,
"speed" : speed
}

/setBusData api:
Request Body:
{
"id" : bus_number,
"longitude" : long,
"latitude" : lat,
"heading" : heading,
"accuracy" : accuracy,
"speed" : speed
}
Response:
{
"status": "success"/"failure"
}

Event Streaming Process

new Kafka Consumer

Idles until new Kafka event occurs in tracked topic

New Thread Created

INSERT data into relational database

Thread.join() (thread is killed)

Kafka Server

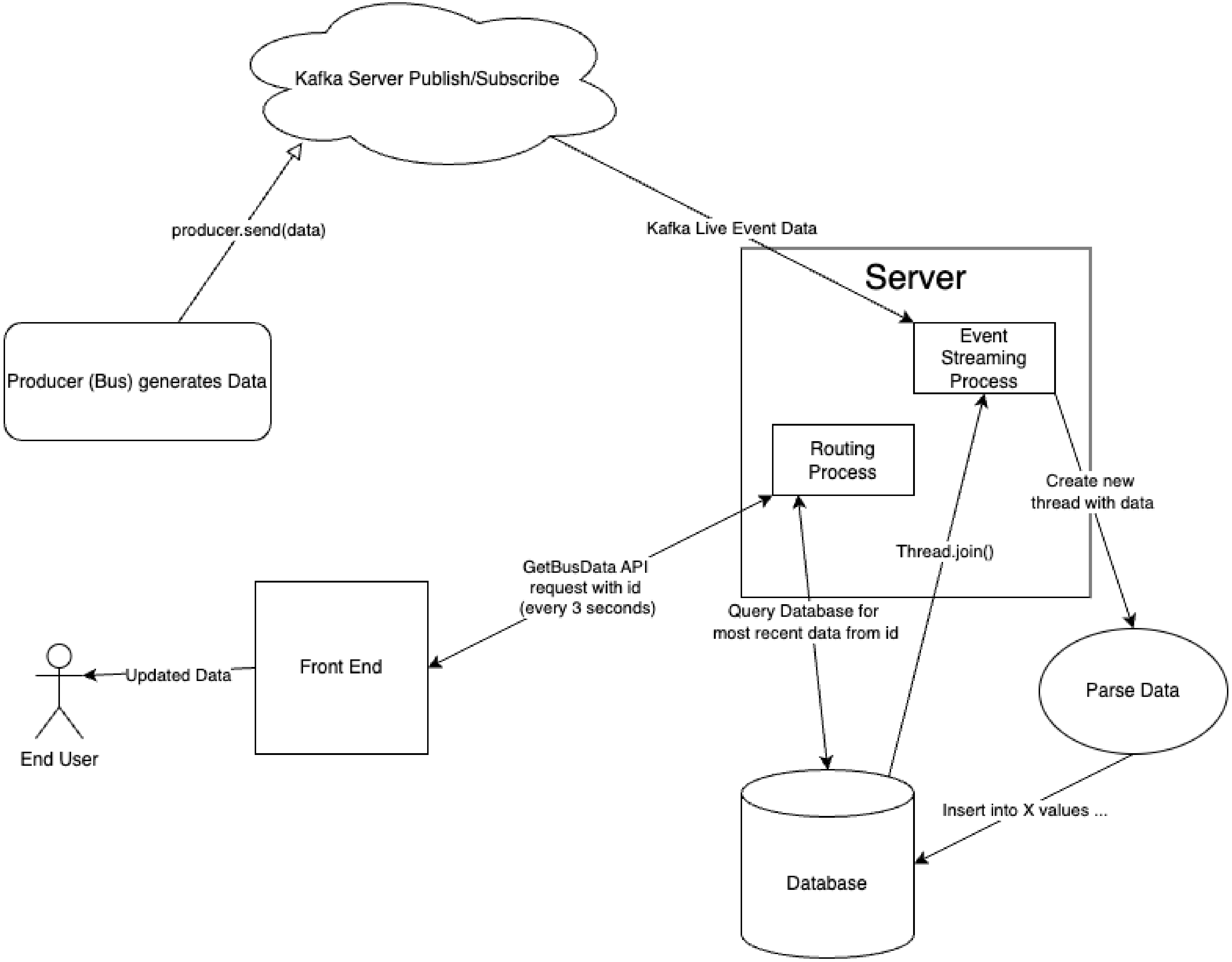
Bus_Data topic created

Bus's are kafka producers, send their data to the kafka server in form of 'event'
ASSUMED FORMAT:
(up to system admins to customize)
{
"id" : bus_number,
"longitude" : long,
"latitude" : lat,
"heading" : heading,
"accuracy" : accuracy,
"speed" : speed
}

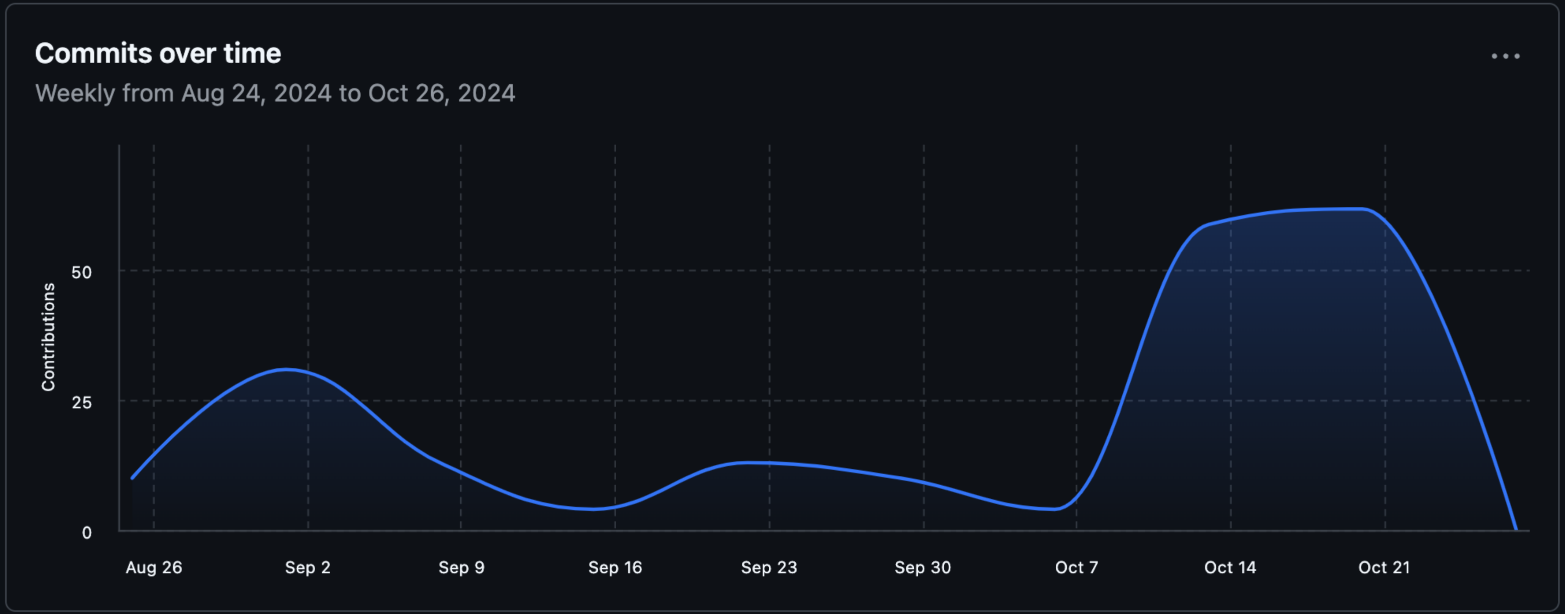
Waits for Kafka Producer to publish data to topic

Publishes new event to every subscriber in topic

Dataflow

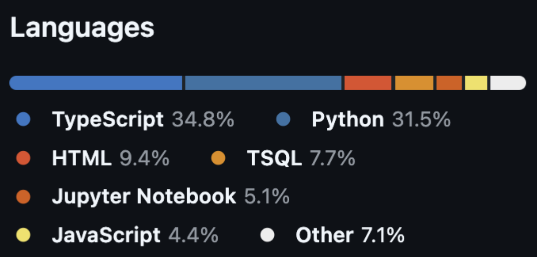

Kafka Server Publish/Subscribe
producer.send(data)
Kafka Live Event Data
Server
Producer (Bus) generates Data
Event Streaming Process
Routing Process
Create new thread with data
Thread.join()
GetBusData API request with id (every 3 seconds)
Query Database for most recent data from id
Front End
Updated Data
End User
Parse Data
Insert into X values ...
Database

Execution



Commits over time

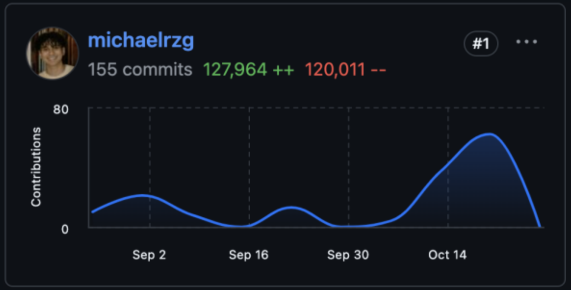
Weekly from Aug 24, 2024 to Oct 26, 2024



Languages

- ● TypeScript 34.8%
- ● Python 31.5%
- ● HTML 9.4%
- ● TSQL 7.7%
- ● Jupyter Notebook 5.1%
- ● JavaScript 4.4%
- ● Other 7.1%

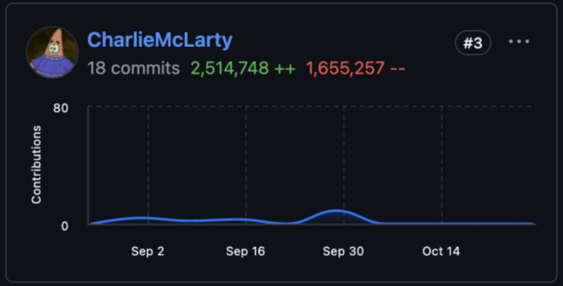
Language Breakdown



michaelrzg #1

155 commits 127,964 ++ 120,011 --

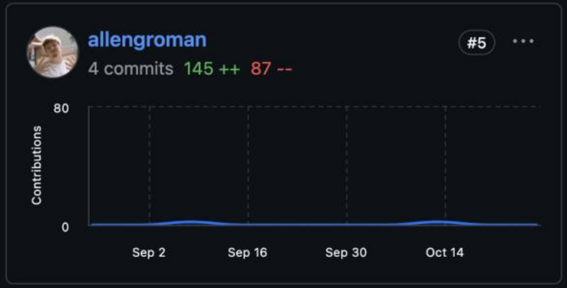
Python Dev, MSSQL, Kafka



CharlieMcLarty #3

18 commits 2,514,748 ++ 1,655,257 --

Data Generation / Simulation



allengroman #5

4 commits 145 ++ 87 --

Containerization

Testing and Quality Assurance

Synthetic Data Generation

- Gwinnett County map converted to directed graph
- Bus calculates distance traveled from speed (m/s) * 5 second's along current edge of graph
- Data collector receives updates from each bus and streams to the Kafka producer

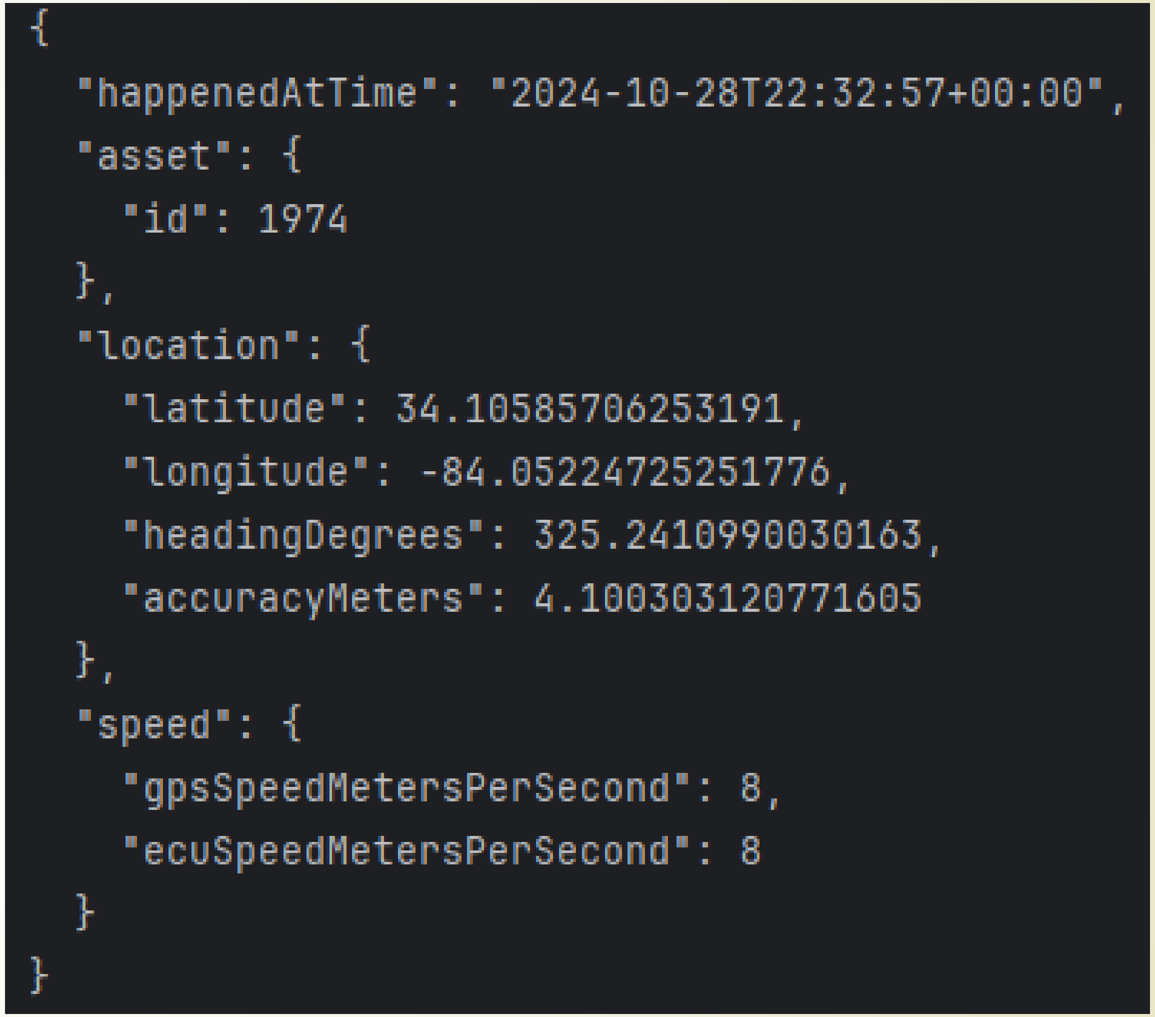
In Progress:

  - Better bus route logic (starts from school and eventually returns)
  - Generating geofences for each school
  - Realistic time simulation of each bus traveling in morning before school to track whether a bus is running on time

Gwinnett county represented as a strongly connected graph

Scaling and Quality Assurance

- Bus location and direction needs to be calculated every 5 seconds to simulate real-life scenario
  - GCPS contains >1500 active buses at peak times
- Object Serialization for graph, node locations, and edge direction/lengths
- Bus location update and data collection tasks run asynchronously

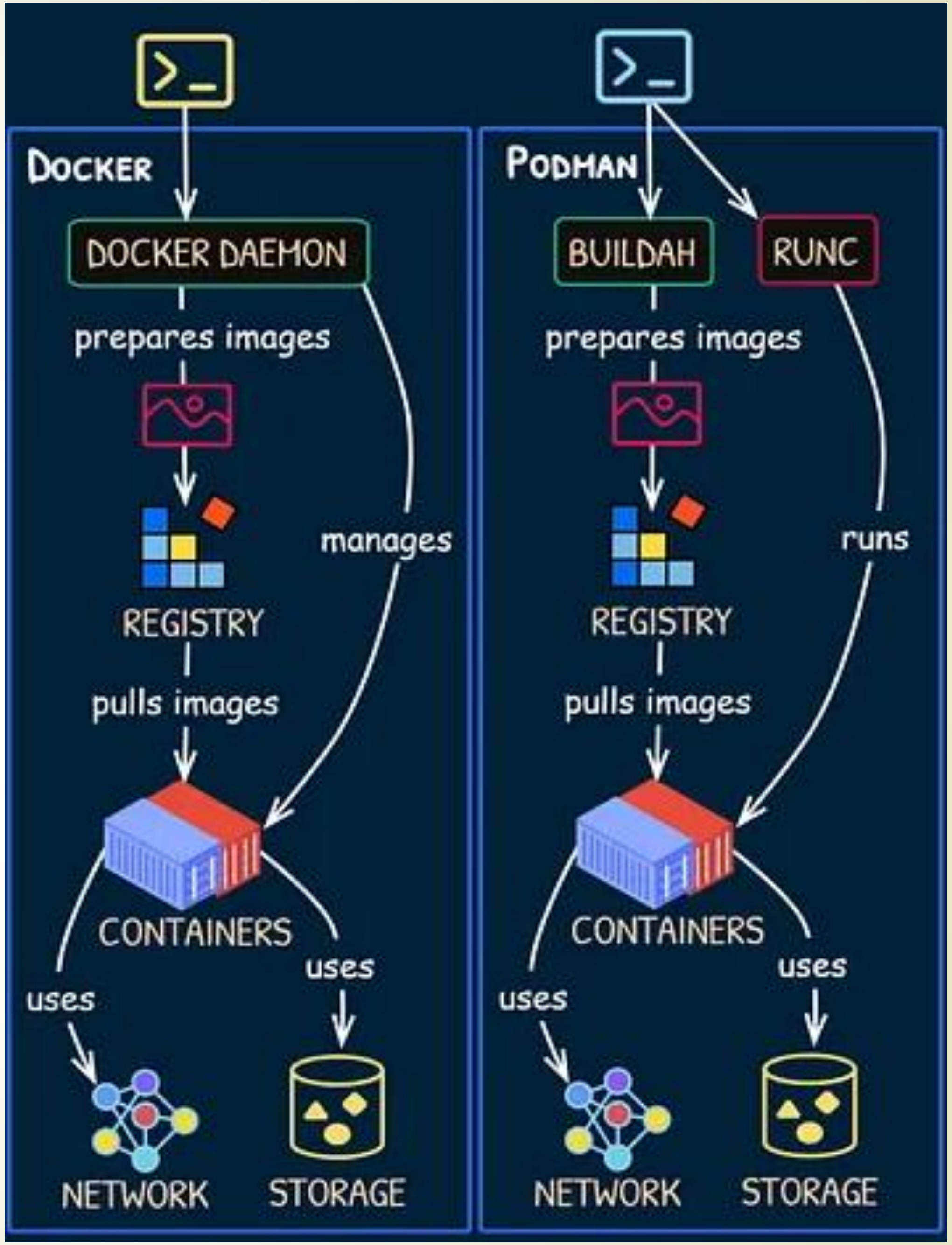


```
{
  "happenedAtTime": "2024-10-28T22:32:57+00:00",
  "asset": {
    "id": 1974
  },
  "location": {
    "latitude": 34.10585706253191,
    "longitude": -84.05224725251776,
    "headingDegrees": 325.2410990030163,
    "accuracyMeters": 4.100303120771605
  },
  "speed": {
    "gpsSpeedMetersPerSecond": 8,
    "ecuSpeedMetersPerSecond": 8
  }
}
```

Deployment

Containerization

- Containers provide isolated environment for applications
- They offer reliable and consistent testing and deployment across different machines
- Podman is a daemon-less alternative to Docker for containerization
- Pods deploy groups of containers together and share resources
- Application runs a Pod with a container for the producer-side and one for the consumer-side



But wait, there’s more!

Front-End Live Bus Tracker (Phase 2)

- **(Optional)** Implement a Front-End live bus tracking application

- When proposing this idea, GCPS mentioned an optional extension to the project to include a front-end app to monitor bus locations on a macro scale.

- While this was optional, we took on the challenge

Tech Stack (Front End)

HTML

React

JS

OpenStreetMap

CSS

Leaflet

Demo

Demonstration

Phase 3: Connection Monitoring and More…

- **Implement a second application that monitors system status, health, and data connection for IT administration.**
- Create multiple roles to allow for different views for users.
- Enhance the frontend icons to allow schools to know which buses are running late.

Challenges

Changing Milestones and Requirements

This was the sponsor's first time collaborating with a senior project class, let alone 3!

Milestones and requirements evolved throughout the project, requiring the team to stay adaptable and proactive.

Frequent adjustments affected planning and resource allocation, making it necessary to revise deliverables on the go.

Technical Learning Curve

Team members had to quickly familiarize themselves with Kafka, containerization (Podman), and MS SQL Server integration on Linux.

The multi-threaded environment introduced complexity, particularly in managing real-time event streams efficiently.

Simulating Real-World Data Loads

Creating realistic simulations to mimic 2,000 buses streaming data in real-time was a challenge.

Testing the system for error handling and performance bottlenecks under synthetic load required iterative development.

Communication Across Platforms

Managing effective communication through Discord, Microsoft Teams, and GitHub was crucial but challenging.

Ensuring that the entire team stayed aligned on changing requirements and timelines required frequent meetings and status updates.

Questions?