# Trimet Bytes - Data Engineering Project

Sri Lakshmi Polavarapu
PSU ID: 914559661

Portland State University
Maseeh College of Engineering & Computer Science Department
CS410/510 - Data Engineering
Professor: Bruce Irvin
TA: Alex Harris, Kira Klingenberg

# System Description

Hello everyone,

I recently worked on a project, which focuses on public transportation in Portland, Oregon, by leveraging real-time transit data from **TriMet**, the region's main public transit service. TriMet is the public agency that operates most buses, and trains in the Portland, Oregon metro area.

This project is focused on analyzing breadcrumb data refers to GPS signals transmitted from TriMet buses through cellular or satellite connections. These signals capture important information like the real-time location, direction, and speed of each bus as it travels through the city.

In addition to breadcrumbs, I also integrated stop event data, which logs when and where buses stop to pick up or drop off passengers. By combining these two data sources, I was able to visualize and analyze complete trip flows across the city.

I designed a system that:

- Streams and processes live data using a Google Cloud Pub/Sub pipeline.
- Stores and organizes the data in a PostgreSQL database for analysis.
- Identifies patterns like frequent delays, and congestion areas.
- Provides insights to support better route planning and overall trimet transit operations.

# Data Pipeline Overview

**Data Gathering:**

To start with, I built an automated system that fetches both **breadcrumb data** (real-time GPS signals) and **stop event data** from TriMet's API. The API returns GPS and stop event details for buses based on specified vehicle IDs. This data is retrieved daily, using scheduled cron jobs set to run at **8:00 PM**, ensuring consistent updates for each vehicle's daily operations.

**Data Ingestion and Streaming:**

To handle data ingestion, I used **Google Cloud Platform**, specifically **Cloud Pub/Sub**, to set up a reliable streaming infrastructure. A custom **Python-based publisher script** retrieves and publishes daily batches of data to the Pub/Sub topic. These batches typically include one full day's data per vehicle ID.

**Data Processing:**

Once ingested, the data flows through a series of processing steps:

- **Validation** – Ensures the incoming data is accurate and complete.
- **Transformation** – Formats timestamps and GPS points, and computes additional metrics like **bus speed** based on sequential location data.
- **Enhancement** – Merges breadcrumb and stop event data to create a unified and detailed view of each trip.

# Data Pipeline Overview

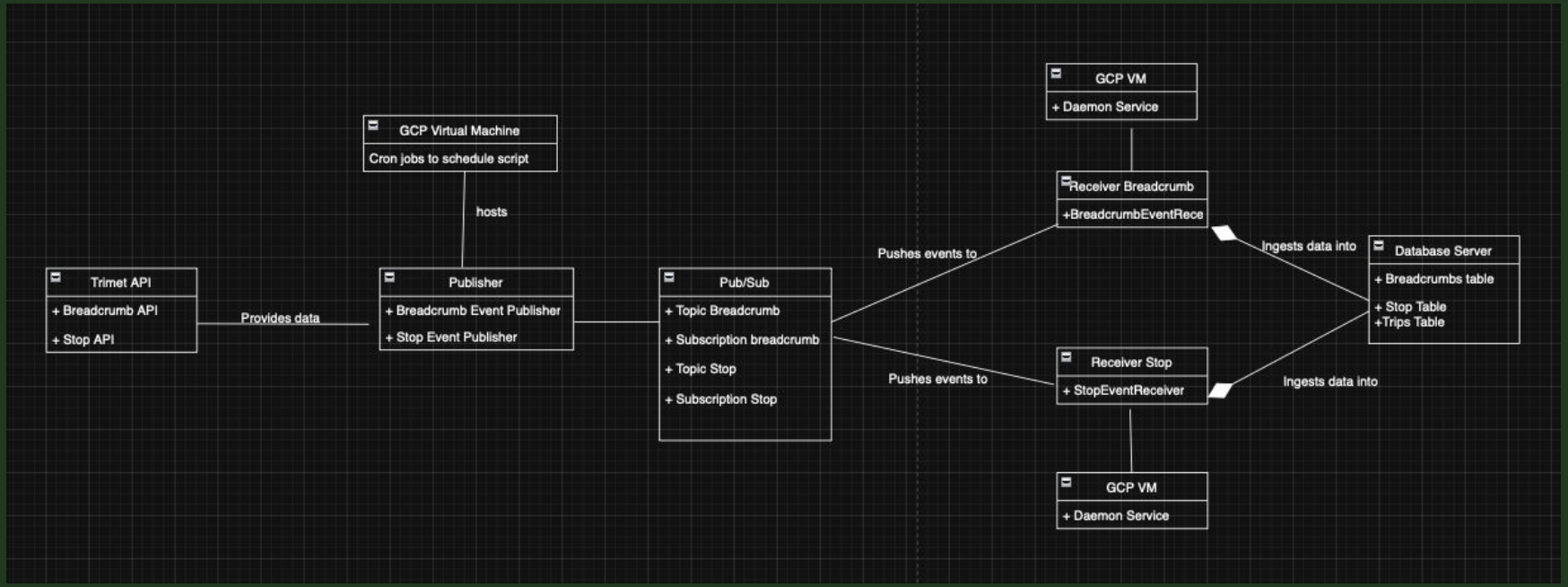
**Data Storage and Management:**

The processed data is stored in a **PostgreSQL** database. I designed the schema to support efficient storage, filtering, and querying of large volumes of transit data, which enables deeper analysis.

**Data Visualization:**

For the visualization layer, I used **Mapbox GL** to map the bus routes and overlay performance metrics such as **speed**, **delays**, and **stop activity** on a real-time map of Portland. This helps in identifying operational issues, monitoring route efficiency, and suggesting improvements to the existing transit network.

# Component Diagram



# Architecture Overview

1. **Data Source**
    a. We use two TriMet APIs:
        i. **Breadcrumb API**: Provides GPS coordinates and timestamps.
        ii. **Stop Events API**: Offers details about bus stop times and durations.

2. **Google Cloud Platform (GCP) Virtual Machines**
    a. I deployed a **single Virtual Machine (VM)** on Google Cloud Platform to handle both data collection and processing tasks efficiently. This VM runs scheduled **Python scripts** that:
        i. **Fetch Data:** Use the requests library to pull **breadcrumb** and **stop event** data from TriMet's APIs, based on vehicle IDs. Data retrieval is automated through a **cron job**.
        ii. **Process Data:** Once fetched, the same VM handles **data cleaning**, **validation**, and **transformation**.

3. **Publisher**
    a. Implemented in Python, these services utilize the google-cloud-pubsub library to publish data to topics within the Google Cloud Pub/Sub system. Each piece of data is formatted as a JSON message and includes metadata like collection timestamps and data source identifiers.

# Architecture Overview (Conti…)

4. **Google Cloud Pub/Sub System**
   - **Topics and Subscriptions**: I have separate topics for breadcrumb and stop event data to segregate the data flows.

5. **Subscriber**
   - These are also Python-based applications subscribing to the Pub/Sub topics. They perform additional validations and transformations. The subscribers then load the data into the PostgreSQL database.

6. **Database Server**
   - **PostgreSQL on GCP**: The database schema is optimized for time-series data analysis, with tables structured to efficiently query spatial and temporal data. I used indices on trip IDs, timestamps, and geographical coordinates to speed up queries.

# Detailed Overview

Google Cloud Virtual Machines (VMs):

- Utilizing VM: I used Python scripts on optimized Google Cloud Platform Virtual Machines to fetch data from TriMet APIs.
- Scheduling of VM: To align with processing demands and enhance cost efficiency, VMs are scheduled to run daily from 08:00 PM to 02:00 AM.

Google Cloud Pub/Sub:

- The system employs GCP Pub/Sub for real-time messaging across different components. Using distinct topics for breadcrumb and stop event data ensures organized and scalable data flows, essential for large-scale data handling.

Cron Job Scheduling:

- Cron jobs on GCP VMs automate the execution of data-fetching and publishing scripts, specifically scheduled to run after VM startup. The runscript.sh triggers these operations at 08:00 PM daily, aligning perfectly with the VMs' operational window to gather data without manual input.

# Publisher

**Data Fetching**:

- Breadcrumb and Stop event publisher scripts operate asynchronously.
- The breadcrumb data script fetches GPS coordinates and timestamps for buses, which detail their routes and movements at 5-second intervals.
- The stop event data script retrieves data about bus stops, including arrival and departure times, which are extracted from HTML content using BeautifulSoup.

**Data Processing and Transformation**:

- **Breadcrumb Data**: After fetching, each GPS record is serialized into JSON and immediately published to a specific Pub/Sub topic.
- **Stop Event Data**: The HTML content is parsed to extract relevant data, convert it into structured formats using pandas, and then serialize it into JSON. This data includes additional context like trip IDs and precise timestamps, enhancing the richness of the dataset.

# Publisher (Conti…)

**Data Publishing**:

- Both scripts use google.cloud.pubsub to connect to Google Cloud Pub/Sub.
- Breadcrumb and stop event data are published to separate topics (breadcrumb-topic for breadcrumbs and stop-events-topic for stop events). This segregation helps in managing data flow and allows for tailored processing downstream.

**Error Handling and Resilience**:

- The scripts include error handling to manage issues such as API unavailability. It handles HTTP 404 errors gracefully by logging and skipping missing data without breaking the execution flow.
- Exception handling mechanisms ensure that any unexpected errors are logged, providing traceability and aiding in quick resolution.

# Subscriber

**Message Handling**:

- The system uses Google Cloud Pub/Sub, a real-time messaging service, to receive and manage streams of data from two separate topics: trimet-topic for GPS breadcrumb data and trimet-stop-topic for stop event data.
- Python-based subscriber  scripts are tasked with subscribing to these topics.

**Speed Calculation**:

- For breadcrumb data, the subscribers calculate the speed of buses by calculating the distance between consecutive GPS data and dividing by the time interval, applying the formula speed=distance / time.

**Error Handling and System Monitoring**:

- Our system uses exception handling across all stages of data processing and data loading. They systematically log errors to prevent data corruption and ensure quick solution. Integrated directly into the subscribers, this approach manages exceptions effectively during data fetching, processing, and loading.

# Subscriber (Conti…)

**Data Insertion into PostgreSQL**:

- Using the psycopg2 library's copy_from method, the receivers efficiently load data into PostgreSQL by using batch processing techniques. This approach was chosen because it handles large data batches.
- By aggregating data into batches before to insertion into database, the system enhances overall performance and minimizes processing times.

**Data Validation Techniques:**

- **Type & Format Checks:** Ensure fields match expected data types and formats (e.g., checking number fields like GPS is actually numeric values).
- **Range Validation:** Confirm values fall within valid limits (e.g., latitude between -90 to 90, longitude between -180 to 180).
- **Non-Negative Checks:** Enforce non-negative values for fields like speed or distance to catch invalid data.
- **Duplicate Detection:** Identify and remove repeated entries to maintain data accuracy and prevent skewed analysis.

# Data Storage

Below is the detailed explanation of the tables - breadcrumb, stop, and trip:

1. **Breadcrumb Table**:
    - This table captures real-time GPS data for transit vehicles.
    - **Columns**:
        - tstamp: Timestamp of the GPS record.
        - latitude and longitude: GPS coordinates of the vehicle's position.
        - speed: Vehicle's speed at the time of the GPS record.
        - trip_id: A foreign key linking the breadcrumb to a specific trip in the trip table.

2. **Stop Table**:
    - Stores data related to transit stops for each vehicle trip.
    - **Columns**:
        - **vehicle_number**: Unique identifier for the vehicle.
        - **trip_number**: Links stop data to the specific trip.
        - **stop_time**: Timestamp of stop event.
        - **maximum_speed**: Performance metric indicating speed at the stop.
        - **direction**: Direction of travel for the trip.
        - **service_key**: Type of service (e.g., weekday, weekend).

# Data Storage (Conti…)

- B**arrive_before_leave**: Ensures arrival time is before leave time.
- **estimated_load**: Estimated passenger load after the stop.
- **dwell**: Duration the vehicle remained at the stop.
- **location_id**: Unique identifier of the stop location.

**3. Trip Table**:

- Central table linking both breadcrumb and stop tables.
- **Columns**:
  - trip_id: Primary key, uniquely identifying each trip.
  - route_id: Identifies the route of the trip.
  - vehicle_id: Identifier for the vehicle used on the trip.
  - service_key, direction: Descriptors categorizing the trip type and direction.

# Exploring Transit Data

- **Total number of days the data was collected: 38 days**
- **Total data collected everyday was around: ~ 75 MB**
- **Breadcrumbs records collected everyday was: ~ 50 MB**
- **Rows added to Trip table:** Each day, approximately **1,623 new trip entries** were added to the trip table,
- **Rows added to Breadcrumb table:** Each day, approximately **294,248 new entries** were added to the table,

# Data Visualization - 1

**Question:** Late Night Trips, showing bus trips running between 10 PM and 12 AM

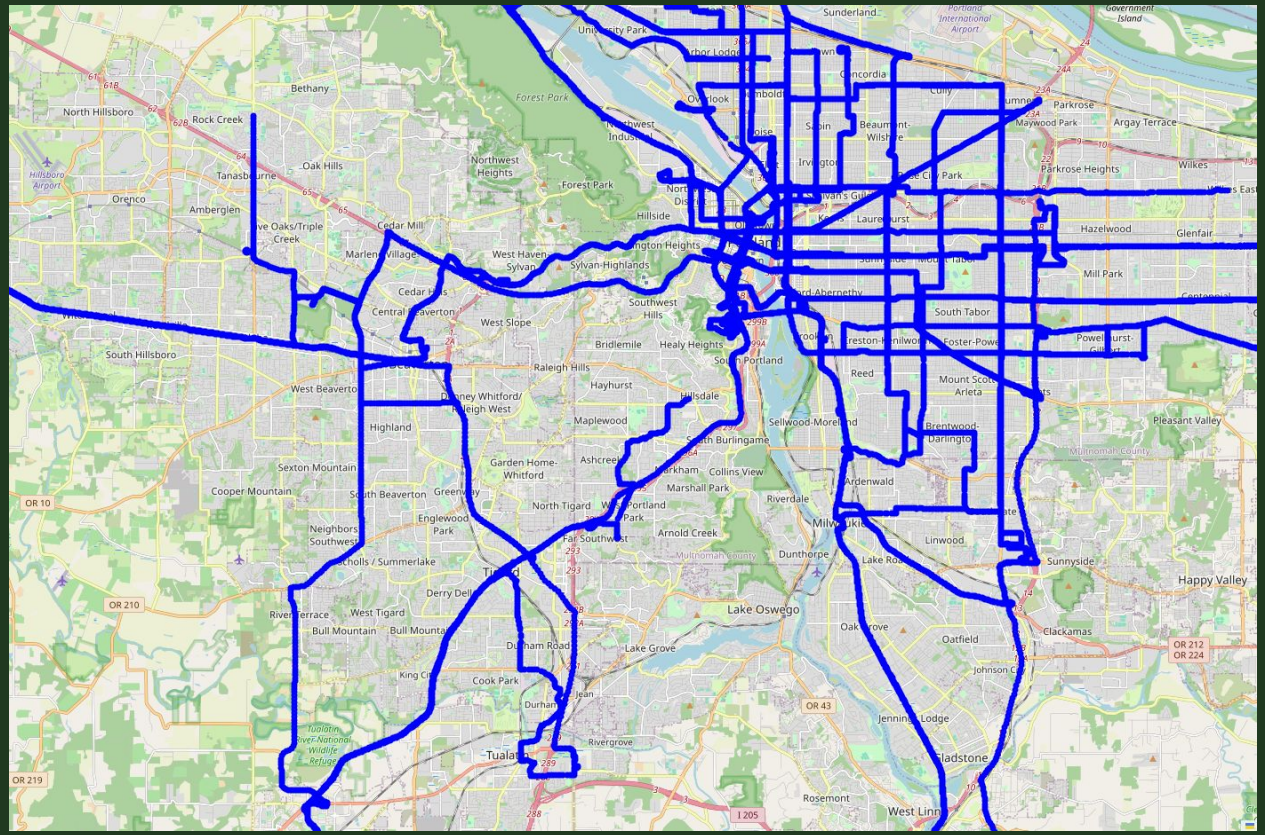
**Query:**

```
SELECT b.latitude, b.longitude, b.speed
FROM breadcrumb b
JOIN trip t ON b.trip_id = t.trip_id
WHERE EXTRACT(HOUR FROM b.tstamp) BETWEEN 22 AND 23
  AND EXTRACT(DOW FROM b.tstamp) = 0
  AND b.latitude IS NOT NULL
  AND b.longitude IS NOT NULL
LIMIT 500;
```

**Purpose:**

- Identifies bus trips between 10 PM and 12 AM
- Filters data specifically for Sundays
- Retrieves latitude, longitude, and speed of buses
- Ensures only records with valid GPS data are included

# Data Visualization - 1 - Output



# Data Visualization - 2

**Question:** All trips that traveled within 1km of Ladd Circle Park (45.508537, -122.649434) on 15th Jan, 2023 before 11am

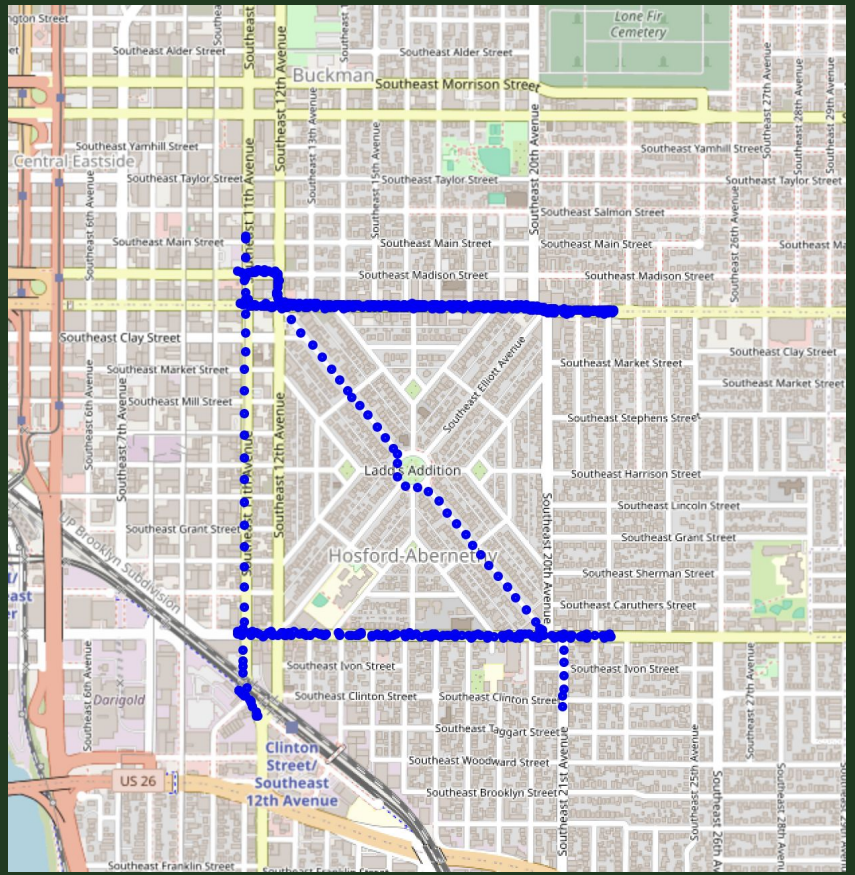
**Query:**

```
SELECT latitude, longitude, speed
FROM breadcrumb
WHERE DATE(tstamp) = '2023-01-15'
  AND EXTRACT(HOUR FROM tstamp) < 11
  AND latitude BETWEEN 45.503 AND 45.514
  AND longitude BETWEEN -122.655 AND -122.643
  AND latitude IS NOT NULL
  AND longitude IS NOT NULL;
```

**Purpose:**

- Retrieves bus trip data from January 15, 2023
- Focuses on trips recorded before 11 AM
- Targets area within ~1 km of Ladd Circle Park using latitude and longitude bounds
- Provides location and speed details for spatial analysis

# Data Visualization - 2 - Output



# Conclusion

- **Successfully processed and visualized GPS data from TriMet.**
- **Collected and managed real-time data by using Data Engineering techniques.**
- **Ensured data integrity through validations and transformations.**
- **Created visualizations to analyze traffic patterns and performance of the trip.**

# Challenges

- **Lesser Data Collected Than Expected:**
  Initially, the data collected was much lower than expected. This was due to misconfigured filters and delays in the source data, causing some messages to be missed.

- **Subscriber Collected Invalid Speed Data:**
  The subscriber was capturing all messages, including those with zero or invalid speeds. This affected data quality until validation checks were added to filter them out.

- **Cron Job Not Running on VM Initially:**
  The cron job wasn't running at first because it wasn't properly set up inside the VM. After configuring it correctly, the data pipeline automation started working as intended.

“Special thanks to **TriMet** for sharing their valuable real-world data, which was instrumental in this project”.

“We Would also like to express our sincere gratitude to **Professor Bruce Irvin** for his guidance and support throughout this project”.

"A big thank you to both the TA’s, **Alex Harris** and **Kira Klingenberg**, for their assistance and valuable feedback."

# THANK YOU!