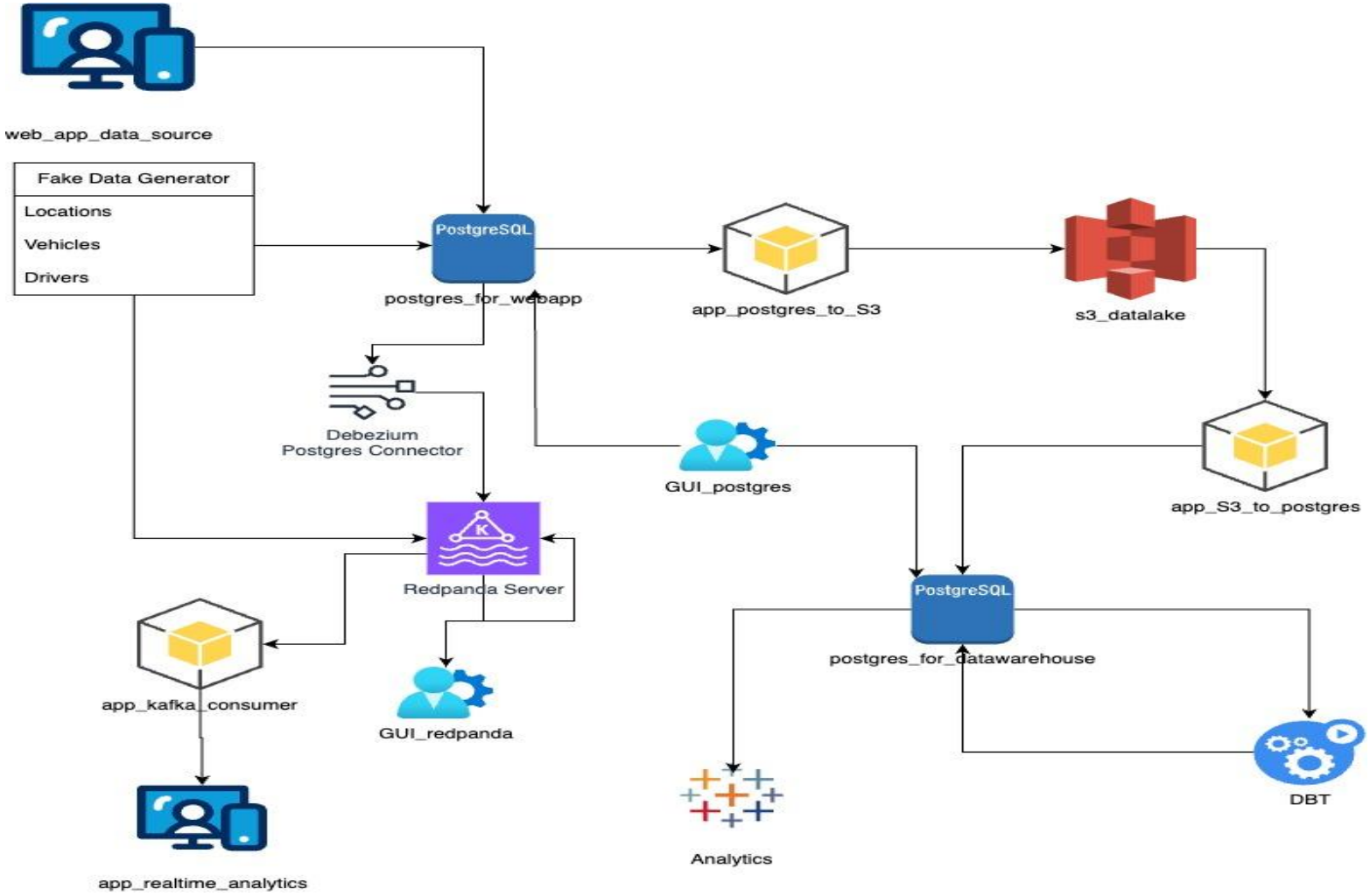
Datapipeline Tech Stack Demo

April 2024

Caleb Bak
Data Engineer, KeyLogic


web_app_data_source
Fake Data Generator
Locations
Vehicles
Drivers
PostgreSQL
postgres_for_webapp
app_postgres_to_S3
s3_datalake
Debezium
Postgres Connector
GUI_postgres
app_S3_to_postgres
Redpanda Server
PostgreSQL
postgres_for_datawarehouse
app_kafka_consumer
GUI_redpanda
DBT
Analytics
app_realtime_analytics

0. Data Generator

- Functionality: Generates trip data for PostgreSQL and Kafka.
- Features:
  - Reads trip data from a CSV file and posts it to the database and Kafka server.
  - Simulates real-time data generation by posting data at regular intervals.
- Database Interaction:
  - Connects to PostgreSQL using the DatabaseHandler class.
  - Adds trip data to the `public.trips` table.
  - Creates a view for real-time location applications.
- Kafka Integration:
  - Utilizes KafkaProducer class to send messages to the Kafka server.
- Use Cases:
  - Facilitates simulated trip data generation for testing and development.
  - Supports database population and real-time data streaming.
  - Suitable for testing real-time analytics, monitoring systems, and demonstrating application functionality.

0. Data Generator

post_data_to_postgres_and_kafka.py

app_data_generator > post_data_to_postgres_and_kafka.py > ...

Run Cell | Run Above | Debug Cell

```python
#%%
#Post data every 1 second
try:

    for i in range(len(df_trip)):

        #create the trip row data
        trip_data = json.loads(df_trip.iloc[i].to_json())

        #podt data to postgres
        db.add_trip(trip_data)

        #alternatively we can post data to redpanda / kafka
        # producer.send('latest_locations', {"row": str(trip_data)})
        producer.send_messages({ str(trip_data)})

        time.sleep(data_refresh_interval)

except Exception as e:
        logger.error(f"Error: {str(e)}.")
        logger.error(traceback.format_exc())
```

DBeaver 24.0.2 - trips

SQL | Commit | Rollback | Auto

postgres_webapp | public@postgres

Data | Proje

Enter a part of object

- postgres_dw - localhost:5
- postgres_webapp - localh
  - Databases
    - postgres
      - Schemas
        - public
          - Tables
            - trips
          - Foreign Tables
          - Views
          - Materialized Vie
          - Indexes

trips | Properties | Data | ER Diagram

postgres_webapp | Databases | postgres | Schemas

trips | Enter a SQL expression to filter

	id	vehicle_id	trip_id
1	1	1	1
2	2	1	1
3	3	3	1
4	4	5	1
5	5	4	1
6	6	2	1
7	7	4	1
8	8	3	1

localhost:5001

Learning | TEK-SYSTEMS | TRADER | Udemy Business | vs-code shortcuts

Latest Trips

Vehicle ID	Trip ID	Measurement Sequence	Measurement Time	Latitude	Longit
7	2	3	2024-04-15 23:20:00	56.9268048001	56.926804
7	2	3	2024-04-15 23:20:00	56.9268048001	56.926804
7	2	3	2024-04-15 23:20:00	56.9268048001	56.926804
11	2	3	2024-04-15 23:20:00	56.9268048001	56.926804
12	2	3	2024-04-15 23:20:00	56.9268048001	56.926804
13	2	3	2024-04-15 23:20:00	56.9268048001	56.926804
5	1	120	2024-04-15 20:45:03	41.1313444054	-136.5522
2	1	120	2024-04-15 20:45:03	43.0895298073	-117.3775
4	1	160	2024-04-16 00:05:03	58.2356368075	-64.00386
1	1	160	2024-04-16 00:05:03	57.9459209802	-63.76319
3	1	241	2024-04-16 06:50:03	67.8835508496	-112.3508

Add a New Trip

Vehicle ID:

Trip ID:

Redpanda

- Overview
- Topics
- Schema Registry
- Consumer Groups
- Security
- Quotas
- Connectors
- Reassign Partitions

Cluster > Topics > latest_locations

857 kiB	911	delete	7 days	Infinite
Size	Messages	cleanup.policy	retention.ms	retention.bytes

Messages | Consumers | Partitions | Configuration

PARTITION: All | START OFFSET: Newest-50 | MAX RESULTS: 50 | FILTER

Offset	Partition	Timestamp	Key	Value
3 354	0	4/18/2024, 2:10:29 PM		"{'vehicle_id': 4, 'trip_id': 1, 'measurement_sequence':

Key: Null | Value: Text (288 B) | Headers: No headers set | Compression: uncompressed | Transactional: false

Key | Value | Headers

"{'vehicle_id': 4, 'trip_id': 1, 'measurement_sequence': 86, 'measurement_time': '2024-04-15 17:55:03', 'latitude': 49.72224
'distance_traveled': 1075, 'fuel_remaining_percent': 64, 'fuel_remaining_gallon': 19.2, 'color': 'yellow'}"

Kafka Consumers

1. Postgres for Webapp

- Functionality: Database server for the web application's data warehouse.
- Dockerfile: Extends `debezium/postgres:16` image and includes initialization SQL script.
- Initialization Script:
  - Defines tables for drivers, vehicles, and trips.
  - Specifies views and constraints.
- Tables:
  - Stores driver information.
  - Stores vehicle information.
  - Stores trip data.
- Views:
  - Defines a view to retrieve the latest locations of vehicles.

1. Postgres for Webapp

EXPLORER

- DATAPIPELINE_V7
  - .config
  - .venv_v6
  - .vscode
  - app_configure_kafkaconnectors
  - app_data_generator
  - app_postgres_to_S3
  - app_realtime_location
  - app_S3_to_postgres
  - app_webapp
  - data_db_dw_postgres
  - data_db_webapp_postgres
  - data_pg_admin
  - data_redpanda
  - data_s3_raw
  - Documentation
    - datapipeline_architecture_diagram.jpg
    - datapipeline_architecture.xml
    - README.md
  - draft_app_kafka_consumer_producer
  - draft_dbt
  - .dockerignore
  - .gitignore
  - docker-compose.yml
  - dockerfile.postgres_datawarehouse
  - dockerfile.postgres_webapp
  - init_postgres_datawarehouse.sql
  - init_postgres_webapp.sql
  - requirements.txt

init_postgres_webapp.sql

```sql
CREATE TABLE trips (
    id SERIAL PRIMARY KEY,
    vehicle_id INTEGER NOT NULL,
    trip_id INTEGER NOT NULL,
    measurement_sequence INTEGER NOT NULL,
    measurement_time TIMESTAMP NOT NULL,
    latitude FLOAT NOT NULL,
    longitude FLOAT NOT NULL,
    fuel_capacity FLOAT NOT NULL,
    distance_traveled FLOAT NOT NULL,
    fuel_remaining_percent FLOAT NOT NULL,
    fuel_remaining_gallon FLOAT NOT NULL,
    color VARCHAR(50) NOT NULL
);


ALTER TABLE public."trips" REPLICA IDENTITY FULL;


drop view if exists latest_locations;
CREATE VIEW latest_locations AS
SELECT *
FROM trips AS t
WHERE (t.vehicle_id, t.trip_id, t.measurement_sequence) IN (
    SELECT vehicle_id, trip_id, MAX(measurement_sequence)
    FROM trips
    GROUP BY vehicle_id, trip_id
);



ALTER TABLE public."drivers" REPLICA IDENTITY FULL; -- This is necessary for kafka streaming to work
ALTER TABLE public."vehicles" REPLICA IDENTITY FULL;
```

2. Pg Admin

- Functionality: Web-based interface for managing PostgreSQL databases.
- Usage: Docker container runs the latest version of pgAdmin.
- Configuration:
  - Default email and password set via environment variables.
- Access:
  - Accessible via web browser at port 5050.
- Persistence:
  - Data stored in a volume mounted to `/var/lib/pgadmin`.
- Use Cases:
  - Database administrators and developers can perform various database tasks.
  - Simplifies monitoring and management of PostgreSQL databases.

3. Web App

- Functionality: Web app for collecting and storing data in the `postgres_for_webapp` database.
- Features:
  - Allows submission of trip data (vehicle info, location, fuel details).
  - Displays latest trip data for each vehicle.
- Database Interaction:
  - Connects to PostgreSQL specified in environment variables.
  - Defines SQLAlchemy model for Trips table.
- Routes:
  - `/`: Displays latest trip data for each vehicle.
  - `/submit`: Handles form submission to insert new trip data.
- Importance:
  - Facilitates collection and storage of real-time transportation data.
  - Supports efficient data retrieval and analysis.
- Use Cases:
  - Real-time monitoring and management of vehicle trips.
  - Logistics tracking, fleet management, transportation analytics platforms.

3. Web App

Latest Trips

Vehicle ID	Trip ID	Measurement Sequence	Measurement Time	Latitude	Longitude	Fuel Capacity	Distance Traveled	Fuel Remaining (%)	Fuel Remaining (gallon)	Color
7	2	3	2024-04-15 23:20:00	56.9268048001	56.9268048001	2.0	2.0	2.0	2.0	red
7	2	3	2024-04-15 23:20:00	56.9268048001	56.9268048001	2.0	2.0	2.0	2.0	red
7	2	3	2024-04-15 23:20:00	56.9268048001	56.9268048001	2.0	2.0	2.0	2.0	red
11	2	3	2024-04-15 23:20:00	56.9268048001	56.9268048001	2.0	2.0	2.0	2.0	red
12	2	3	2024-04-15 23:20:00	56.9268048001	56.9268048001	2.0	2.0	2.0	2.0	red
13	2	3	2024-04-15 23:20:00	56.9268048001	56.9268048001	2.0	2.0	2.0	2.0	red
5	1	120	2024-04-15 20:45:03	41.1313444054	-136.5522584619	50.0	2000.0	86.0	43.3	green
2	1	120	2024-04-15 20:45:03	43.0895298073	-117.3775881257	10.0	1000.0	0.0	0.0	red
4	1	160	2024-04-16 00:05:03	58.2356368075	-64.0038619396	30.0	2000.0	33.0	10.0	orange
1	1	160	2024-04-16 00:05:03	57.9459209802	-63.7631951125	50.0	2000.0	73.0	36.7	yellow
3	1	241	2024-04-16 06:50:03	67.8835508496	-112.3508989575	30.0	2000.0	33.0	9.9	orange

Add a New Trip

Vehicle ID: 12
Trip ID: 2
Measurement Sequence: 3
Measurement Time: 2024-04-15 23:20:0
Latitude: 56.9268048001
Longitude: 56.9268048001
Fuel Capacity: 2
Distance Traveled: 2
Fuel Remaining (%): 2
Fuel Remaining (gallon): 2
Color: red
Submit

4. App: Postgres to S3

- Functionality: Transfers data from PostgreSQL to an S3 bucket.
- Features:
  - Fetches environment variables for configuration.
  - Connects to PostgreSQL and S3.
  - Creates S3 bucket if not existent.
  - Tracks last processed ID and updates it after processing.
  - Retrieves new entries from the database.
  - Uploads data to S3 in JSON format.
- Importance:
  - Enables data transfer from database to cloud storage.
  - Supports data backup, archiving, and analysis.
- Use Cases:
  - Periodic backups of database data.
  - Data sharing between systems.
  - Data storage for analytics and reporting.

4. App: Postgres to S3

EXPLORER

DATAPIPELINE_V7

- .config
- .venv_v6
- .vscode
- app_configure_kafkaconnectors
- app_data_generator
- app_postgres_to_S3
  - app.py
  - dockerfile.postgres_to_s3
  - entrypoint.sh
  - last_id.txt
- app_realtime_location
- app_S3_to_postgres
- app_webapp
- data_db_dw_postgres
- data_db_webapp_postgres
- data_pg_admin
- data_redpanda
- data_s3_raw
- Documentation
  - datapipeline_architecture_diagram.jpg
  - datapipeline_architecture.xml
- README.md

app.py

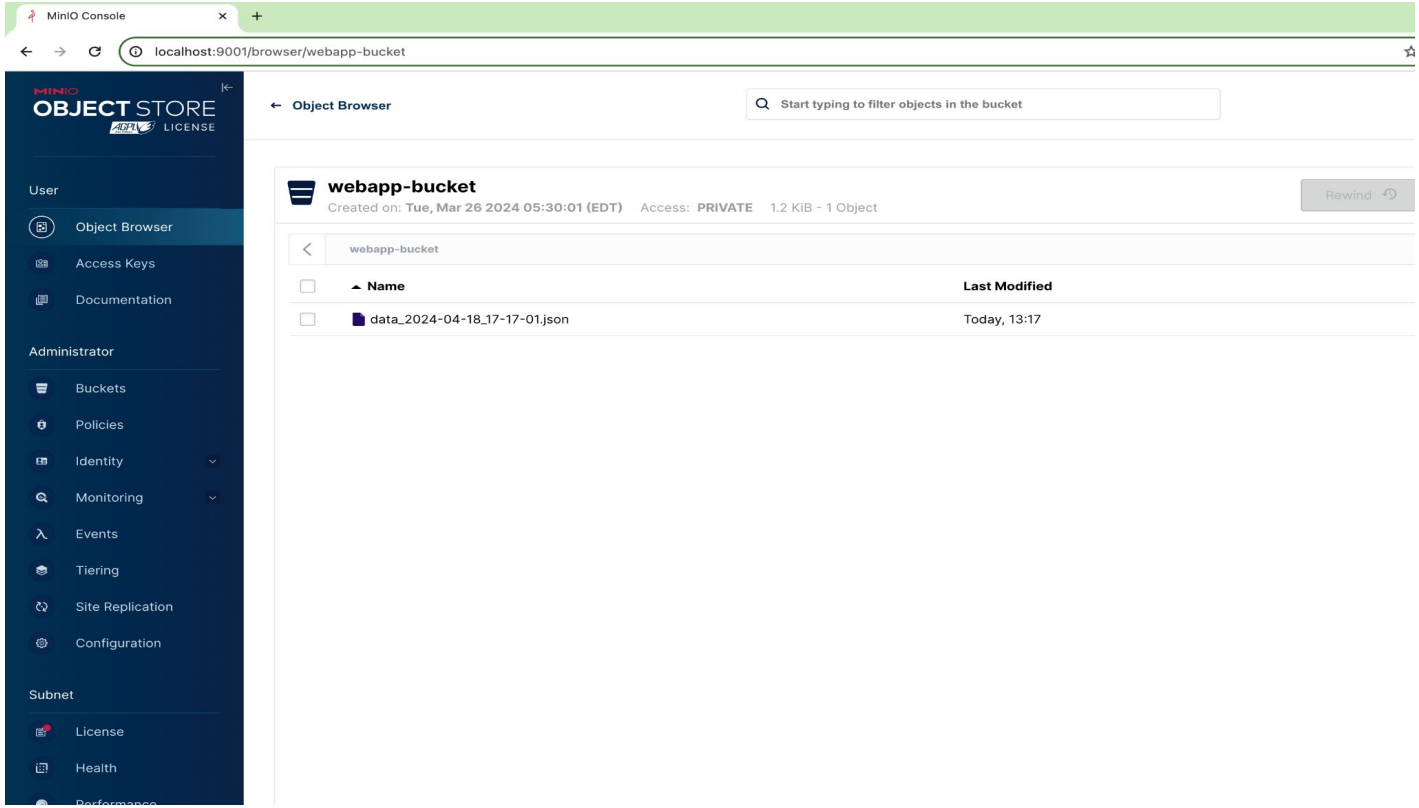
app_postgres_to_S3 > app.py > update_last_processed_id > f

```python
def upload_to_s3(data, file_name):
    logging.info("Uploading file to S3.")
    s3_client.put_object(
        Bucket=S3_BUCKET_NAME,
        Key=file_name,
        Body=json.dumps(data, default=str)  # Convert data to JSON, handling dates etc
    )
    logging.info(f"File '{file_name}' uploaded to S3.")

def main():
    """Main function to fetch new entries and upload them to S3."""
    logging.info("Starting main function...")
    last_id = get_last_processed_id()
    new_entries = fetch_new_entries(last_id)
    if new_entries:
        # Update the last processed ID based on the entries fetched
        last_processed_id = new_entries[-1]['id']  # Assuming 'id' is the name of your
        update_last_processed_id(last_processed_id)
        timestamp = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
        file_name = f"data_{timestamp}.json"
        upload_to_s3(new_entries, file_name)
    else:
        logging.info("No new entries to upload.")

if __name__ == "__main__":
    main()
```

5. Minio / S3 as Landing Zone

- Functionality: Serves as an S3 bucket for storing raw data files.
- Features:
  - Utilizes MinIO Docker image for S3-compatible storage.
  - Exposes ports 9000 and 9001 for S3 operations and MinIO console access.
  - Depends on `postgres_for_webapp` and `postgres_for_datawarehouse`.
  - Sets environment variables for access keys.
  - Mounts volumes for data storage and configuration.
  - Defines command to start MinIO server with console access.
  - Includes health check for server status.
- Importance:
  - Centralizes storage for raw data files.
  - Facilitates data ingestion and processing.
- Use Cases:
  - Secure storage for raw data files.
  - Data access by multiple services for processing or analysis.

5. Minio / S3 as Landing Zone



6. App: S3 to Postgres

- Functionality: Lambda function transferring data from S3 to PostgreSQL.
- Features:
  - Connects to PostgreSQL and S3 using provided credentials.
  - Retrieves JSON files from specified S3 bucket.
  - Parses JSON data and inserts into designated PostgreSQL table.
  - Ensures data integrity by avoiding duplicate entries.
  - Logs information for monitoring and debugging.
- Importance:
  - Facilitates ingestion of S3 data into structured database format.
- Use Cases:
  - Regular processing and analysis of data stored in S3.
  - Data enrichment through SQL queries or other operations.

6. App: S3 to Postgres

EXPLORER

DATAPIPELINE_V7

- .config
- .venv_v6
- .vscode
- app_configure_kafkaconnectors
- app_data_generator
- app_postgres_to_S3
- app_realtime_location
- app_S3_to_postgres
  - app.py
  - dockerfile.s3_to_postgres
  - entrypoint.sh
- app_webapp
- data_db_dw_postgres
- data_db_webapp_postgres
- data_pg_admin
- data_redpanda
- data_s3_raw
- Documentation
- draft_app_kafka_consumer_producer
- draft_dbt
- .dockerignore
- .gitignore
- docker-compose.yml
- dockerfile.postgres_datawarehouse
- dockerfile.postgres_webapp
- init_postgres_datawarehouse.sql
- init_postgres_webapp.sql
- requirements.txt

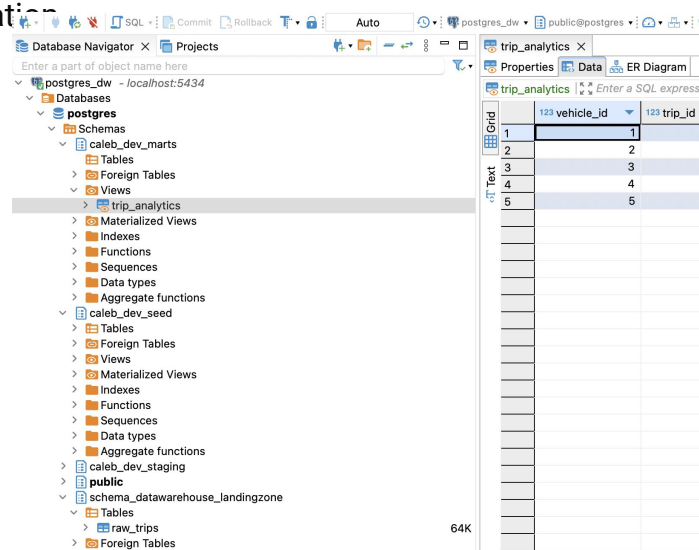
app.py

app_S3_to_postgres > app.py > ...

```python
def insert_data_to_postgres(data, table_name="raw_trips", table_schema="schema_datawarehouse_landingzone"):
                    query = f"""
                        INSERT INTO {table_schema}.{table_name} ({column_names})
                        VALUES ({placeholders})
                        ON CONFLICT DO NOTHING
                    """
                    cursor.execute(query, column_values)
                conn.commit()
                logging.info("Data inserted into PostgreSQL successfully.")
        except psycopg2.Error as e:
            logging.error(f"Failed to insert data into PostgreSQL: {e}")


# Main function to process S3 data and insert it into PostgreSQL
def process_s3_data():
    """Retrieve data from S3 and insert it into PostgreSQL."""
    files = retrieve_files_from_s3()
    for file_name in files:
        data = retrieve_data_from_file(file_name)
        if data:
            insert_data_to_postgres(data, table_name, table_schema)
            # Optionally, delete the file from S3 after processing
            s3_client.delete_object(Bucket=S3_BUCKET_NAME, Key=file_name)
            logging.info(f"Deleted file: {file_name}")


if __name__ == "__main__":
    process_s3_data()

Run Cell | Run Above | Debug Cell
# %%
```

7. PostGres for Data Warehouse

- Functionality: PostgreSQL server for enterprise data warehouse.
- Features:
  - Utilizes `debezium/postgres:16` Docker image.
  - Container named `postgres_for_datawarehouse`.
  - Exposes port 5434 for PostgreSQL connections.
  - Sets environment variables for user, password, and database name.
  - Mounts `init_postgres_datawarehouse.sql` for database initialization.
  - Defines `raw_trips` table for trip data storage.
  - Creates `latest_locations` view for retrieving latest location data.
- Use Case:
  - Dedicated database server for enterprise data warehousing.
  - Supports analytics and reporting.



8. Redpanda Console

- Functionality: GUI for managing and monitoring Redpanda clusters.
- Features:
  - Provides graphical interface for administering and monitoring Redpanda.
  - Allows configuration of Kafka brokers and enables schema registry.
  - Accessed via port mapping.
  - Depends on `redpanda_server` service for communication.
- Importance:
  - Facilitates easy administration, configuration, and monitoring of Redpanda clusters.
  - Enhances operational efficiency and management of streaming data workflows.
  - Essential for organizations managing data-intensive applications or streaming platforms.

8. Redpanda Console

Overview - Redpanda Consol

localhost:8080/overview

Redpanda

- Overview
- Topics
- Schema Registry
- Consumer Groups
- Security
- Quotas
- Connectors
- Reassign Partitions

Cluster > **Overview**

Running	13.4 MiB	Redpanda v23.2.6	1 of 1	8	38
Cluster Status	Cluster Storage Size	Cluster Version	Brokers Online	Topics	Replicas

BROKER DETAILS

ID	Status	Size	
0	Running	13.4 MiB	View

RESOURCES AND UPDATES

- Documentation
- CLI Tools
- Unable to fetch news, please try again later

CLUSTER DE

SERVICES

Kafka Conne

Schema Reg

STORAGE

Total Bytes

Primary

Replicated

SECURITY

Service Acc

ACLs

Licensing

webapp.public.trips - Redpan

localhost:8080/topics/webapp.public.trips?p=-1&s=50&o=-1#messages

Redpanda

- Overview
- Topics
- Schema Registry
- Consumer Groups
- Security
- Quotas
- Connectors
- Reassign Partitions

Cluster > Topics > **webapp.public.trips**

11.7 MiB	2,861	delete	7 days	Infinite
Size	Messages	cleanup.policy	retention.ms	retention.bytes

Messages | Consumers | Partitions | Configuration

PARTITION: All | START OFFSET: Newest-50 | MAX RESULTS: 50 | FILTER

Offset	Partition	Timestamp	Key	Value Preview
3 182	0	4/18/2024, 4:52:49 PM	{"schema":{"type":"struct","fields":[{"ty...	{"schema":{"type":"struct",...

Key	Value	Headers	Compression	Transactional
Json (170 B)	Json (3.55 kiB)	No headers set	uncompressed	false

Key | Value | Headers


```
{ 2 items
  "schema" : { 5 items
    "type" : "struct"
    "fields" : [...] 6 items
    "optional" : false
    "name" : "webapp.public.trips.Envelope"
    "version" : 1
  }
  "payload" : { 6 items
    "before" : NULL
    "after" : { 12 items
      "id" : 3181
      "vehicle_id" : 5
      "trip_id" : 1
      "measurement_sequence" : 60
      "measurement_time" : 1713195903000000
      "latitude" : 40.4798222027
      "longitude" : -117.4737610337
      "fuel_capacity" : 50
      "distance_traveled" : 999
      "fuel_remaining_percent" : 93
      "fuel_remaining_gallon" : 46.7
      "color" : "green"
    }
```

9. Redpanda Server

- Functionality: Redpanda server providing Kafka-compatible streaming service.
- Features:
  - Hosts Redpanda server container for Kafka-compatible streaming.
  - Maps ports for external access to Redpanda services.
  - Executes startup commands for Redpanda server configuration.
- Importance:
  - Core component of Redpanda platform for high-performance, real-time data streaming.
  - Enables scalability and reliability for streaming applications and data pipelines.
  - Ideal for event-driven architectures, real-time analytics, log aggregation, and distributed messaging systems.

9. Redpanda Server

EXPLORER

DATAPIPELINE_V7

- .config
- .venv_v6
- .vscode
- app_configure_kafkaconnectors
- app_data_generator
- app_postgres_to_S3
- app_realtime_location
- app_S3_to_postgres
- app_webapp
- data_db_dw_postgres
- data_db_webapp_postgres
- data_pg_admin
- data_redpanda
- data_s3_raw
- Documentation
- draft_app_kafka_consumer_producer
- draft_dbt
- .dockerignore
- .gitignore
- docker-compose.yml
- dockerfile.postgres_datawarehouse
- dockerfile.postgres_webapp
- init_postgres_datawarehouse.sql
- init_postgres_webapp.sql
- requirements.txt

docker-compose.yml

docker-compose.yml > {} services > {} app_s3_to_postgres > [ ] environment > 5

```yaml
services:


  redpanda_server: #This is the Redpanda server as a kafka compatible streaming service
    container_name: redpanda_server
    image: docker.redpanda.com/redpandadata/redpanda:v23.2.6
    volumes:
      - ./data_redpanda:/var/lib/redpanda/data
    ports:
      - 18081:18081
      - 18082:18082
      - 19092:19092
      - 19644:9644
    healthcheck:
      test:
        ["CMD-SHELL", "rpk cluster health | grep -E 'Healthy:.+true' || exit 1"]
      interval: 15s
      timeout: 3s
      retries: 5
      start_period: 5s
    command:
      - redpanda
      - start
      - --kafka-addr internal://0.0.0.0:9092,external://0.0.0.0:19092
      # Address the broker advertises to clients that connect to the Kafka API.
      # Use the internal addresses to connect to the Redpanda brokers'
      # from inside the same Docker network.
      # Use the external addresses to connect to the Redpanda brokers'
      # from outside the Docker network.
      - --advertise-kafka-addr internal://redpanda_server:9092,external://localhost:19092
      - --pandaproxy-addr internal://0.0.0.0:8082,external://0.0.0.0:18082
      # Address the broker advertises to clients that connect to the HTTP Proxy.
      - --advertise-pandaproxy-addr internal://redpanda_server:8082,external://localhost:18082
      - --schema-registry-addr internal://0.0.0.0:8081,external://0.0.0.0:18081
      # Redpanda brokers use the RPC API to communicate with eachother internally.
      - --rpc-addr redpanda_server:33145
      - --advertise-rpc-addr redpanda_server:33145
      # Tells Seastar (the framework Redpanda uses under the hood) to use 1 core on the system.
      - --smp 1
      # The amount of memory to make available to Redpanda.
      - --memory 1G
      # Mode dev-container uses well-known configuration properties for development in containers.
      - --mode dev-container
      # enable logs for debugging.
      - --default-log-level=debug
```

10. Redpanda Init

- Functionality: Starts Kafka service and creates a topic, exiting upon execution.
- Features:
  - Hosts the initialization container for Redpanda.
  - Depends on the health status of the redpanda_server service before execution.
  - Executes command to create a Kafka topic with specified replication factor and partition count.
  - Specifies brokers for topic creation.
- Importance:
  - Crucial role in setting up Redpanda environment by initializing Kafka service and configuring essential components.
  - Ensures required Kafka infrastructure is ready for use.
- Use Cases:
  - Deployment scenarios requiring automated initialization of Redpanda services, including Kafka topics.

11. Debezium

- Functionality: Facilitates integration between data sources and Apache Kafka for change data capture (CDC).
- Features:
  - Utilizes Debezium Connect Docker image version 2.4.
  - Configures environment variables for connecting to Redpanda Kafka server, specifying bootstrap servers, group ID, and storage topics.
  - Depends on availability of redpanda_server and postgres_for_webapp services before starting.
  - Exposes port 8083 for communication with external systems.
  - Supports monitoring and management of connectors through REST API endpoints.
- Importance:
  - Critical role in enabling real-time data streaming and synchronization between data sources (e.g., PostgreSQL) and Apache Kafka.
  - Captures and propagates database changes to downstream systems for analytics, data warehousing, and event-driven architectures.
- Use Cases:
  - Enables downstream systems to react to data changes instantly.
  - Commonly used in microservices architectures, event sourcing, data integration, and replication scenarios.

11. Debezium

EXPLORER

DATAPIPELINE_V7

- .config
- .venv_v6
- .vscode
- app_configure_kafkaconnectors
- app_data_generator
- app_postgres_to_S3
- app_realtime_location
- app_S3_to_postgres
- app_webapp
- data_db_dw_postgres
- data_db_webapp_postgres
- data_pg_admin
- data_redpanda
- data_s3_raw
- Documentation
- draft_app_kafka_consumer_producer
- draft_dbt
- .dockerignore
- .gitignore
- docker-compose.yml
- dockerfile.postgres_datawarehouse
- dockerfile.postgres_webapp
- init_postgres_datawarehouse.sql
- init_postgres_webapp.sql
- requirements.txt

docker-compose.yml

docker-compose.yml > {} services > {} app_s3_to_postgres > [ ] envi

```yaml
services:
  debezium:
    image: debezium/connect:2.4
    container_name: debezium
    environment:
      BOOTSTRAP_SERVERS: redpanda_server:9092
      GROUP_ID: 1
      CONFIG_STORAGE_TOPIC: connect_configs
      OFFSET_STORAGE_TOPIC: connect_offsets
      # KEY_CONVERTER: io.confluent.connect.avro.AvroCon
      # VALUE_CONVERTER: io.confluent.connect.avro.AvroC
    depends_on: [redpanda_server, postgres_for_webapp]
    ports:
      - 8083:8083
    healthcheck:
      test:
        [
          'CMD',
          'curl',
          '--silent',
          '--fail',
          '-X',
          'GET',
          'http://localhost:8083/connectors',
        ]
      start_period: 10s
      interval: 10s
      timeout: 5s
      retries: 5
```

12. Debezium Init

- Functionality:
  - Initializes the Debezium connector to capture changes from PostgreSQL database and publish them to Kafka topics.
- Features:
  - Dependencies on `debezium`, `postgres_for_webapp`, and `redpanda_server` services.
  - Executes a shell script `config_debezium_postgres_connector.sh`.
  - Mounts `./app_configure_kafkaconnectors` to `/app` inside the container.

```
curl -i -X POST -H "Accept:application/json" \
-H "Content-Type:application/json" \
"http://debezium:8083/connectors/" \
--data-raw '{
    "config": {
        "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
        "plugin.name": "pgoutput",
        "database.hostname": "postgres_for_webapp",
        "database.port": "5432",
        "database.dbname": "postgres",
        "database.user": "admin",
        "database.password": "password",
        "database.server.name": "postgres-webapp",
        "table.include.list": "public.trips",
        "topic.prefix": "webapp"
    },
    "name": "webapp_postgres"
}'
```

13. Debezium Console

- Functionality:
  - Provides a graphical user interface (GUI) for monitoring and managing Debezium connectors.
- Features:
  - Utilizes the Debezium UI Docker image hosted on Quay.io.
  - Automatically restarts in case of failures to ensure continuous availability.
  - Depends on the availability of `debezium`, `redpanda_server`, and `postgres_for_webapp` services.
  - Exposes port 8081 for external access to the UI.
  - Configures environment variables for specifying Kafka Connect URIs to connect to the Debezium connector REST API.
- Importance:
  - Enables visual monitoring and management of Debezium connectors in real-time.
  - Enhances the usability and accessibility of Debezium for administrators and developers.
  - Simplifies the setup and management of data integration pipelines.

13. Debezium Console

Debezium UI

localhost:8081

debezium http://debezium:8083

Connectors

This list shows all connectors that have been created. You can create a new connector by clicking the 'Create

Name

Create a connector

Name
webapp_postgres

Connector name

webapp_postgres RUNNING

Table view JSON view

connector.class	io.debezium.connector.postgresql.PostgresConnector
connector.displayName	PostgreSQL
database.dbname	postgres
database.user	admin
database.server.name	postgres-webapp
connector.id	postgres
plugin.name	pgoutput
database.port	5432
topic.prefix	webapp
database.hostname	postgres_for_webapp
database.password	********
name	webapp_postgres
table.include.list	public.trips

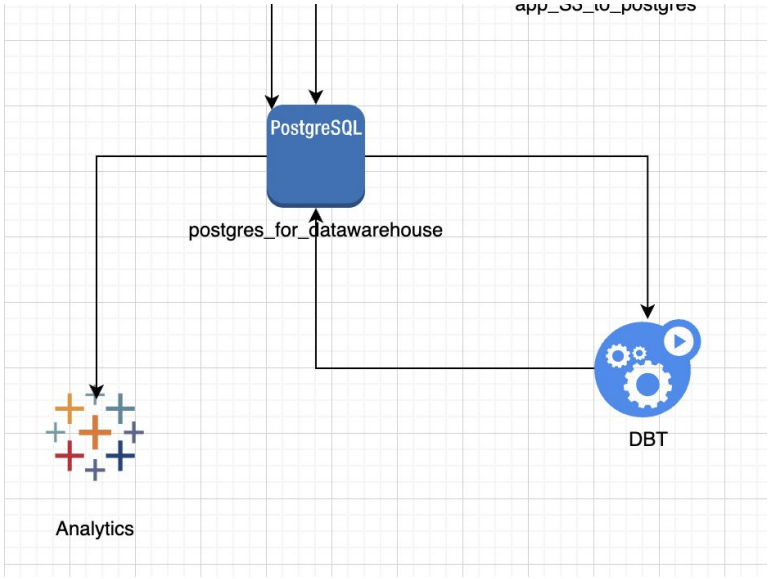
15. dbt

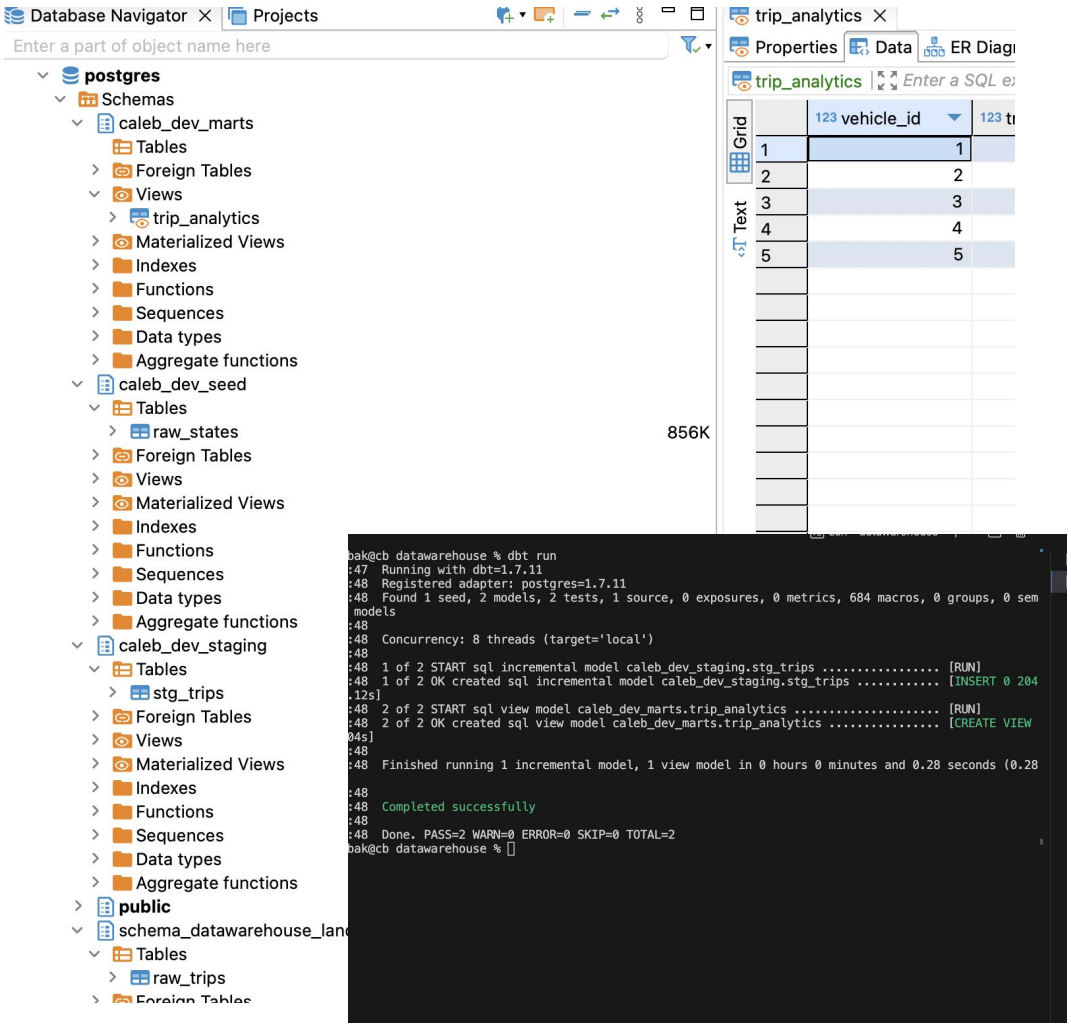
Functionality:
Represents a Dockerized dbt (data build tool) environment for running analytics workflows, including modeling, testing, and documentation generation.

Use case:
Suitable for data teams working on data modeling, transformation, and analytics projects. It can be used in data warehouses, analytics platforms, and ETL pipelines to automate data transformation processes, perform data quality checks, and generate documentation for data models.

The Dockerized dbt environment simplifies project setup and management, making it ideal for agile data teams operating in dynamic and collaborative environments.

15. dbt


PostgreSQL
postgres_for_datawarehouse
DBT
Analytics


Database Navigator
Projects
Enter a part of object name here
postgres
Schemas
caleb_dev_marts
Tables
Foreign Tables
Views
trip_analytics
Materialized Views
Indexes
Functions
Sequences
Data types
Aggregate functions
caleb_dev_seed
Tables
raw_states
856K
Foreign Tables
Views
Materialized Views
Indexes
Functions
Sequences
Data types
Aggregate functions
caleb_dev_staging
Tables
stg_trips
Foreign Tables
Views
Materialized Views
Indexes
Functions
Sequences
Data types
Aggregate functions
public
schema_datawarehouse_lan
Tables
raw_trips
trip_analytics
Properties
Data
ER Diag
trip_analytics
Enter a SQL ex
Grid
Text
123 vehicle_id
1
2
3
4
5
bak@cb datawarehouse % dbt run
:47 Running with dbt=1.7.11
:48 Registered adapter: postgres=1.7.11
:48 Found 1 seed, 2 models, 2 tests, 1 source, 0 exposures, 0 metrics, 684 macros, 0 groups, 0 sem
models
:48
:48 Concurrency: 8 threads (target='local')
:48
:48 1 of 2 START sql incremental model caleb_dev_staging.stg_trips ..... [RUN]
:48 1 of 2 OK created sql incremental model caleb_dev_staging.stg_trips ..... [INSERT 0 204
.12s]
:48 2 of 2 START sql view model caleb_dev_marts.trip_analytics ..... [RUN]
:48 2 of 2 OK created sql view model caleb_dev_marts.trip_analytics ..... [CREATE VIEW
04s]
:48
:48 Finished running 1 incremental model, 1 view model in 0 hours 0 minutes and 0.28 seconds (0.28
:48
:48 Completed successfully
:48
:48 Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2
bak@cb datawarehouse %

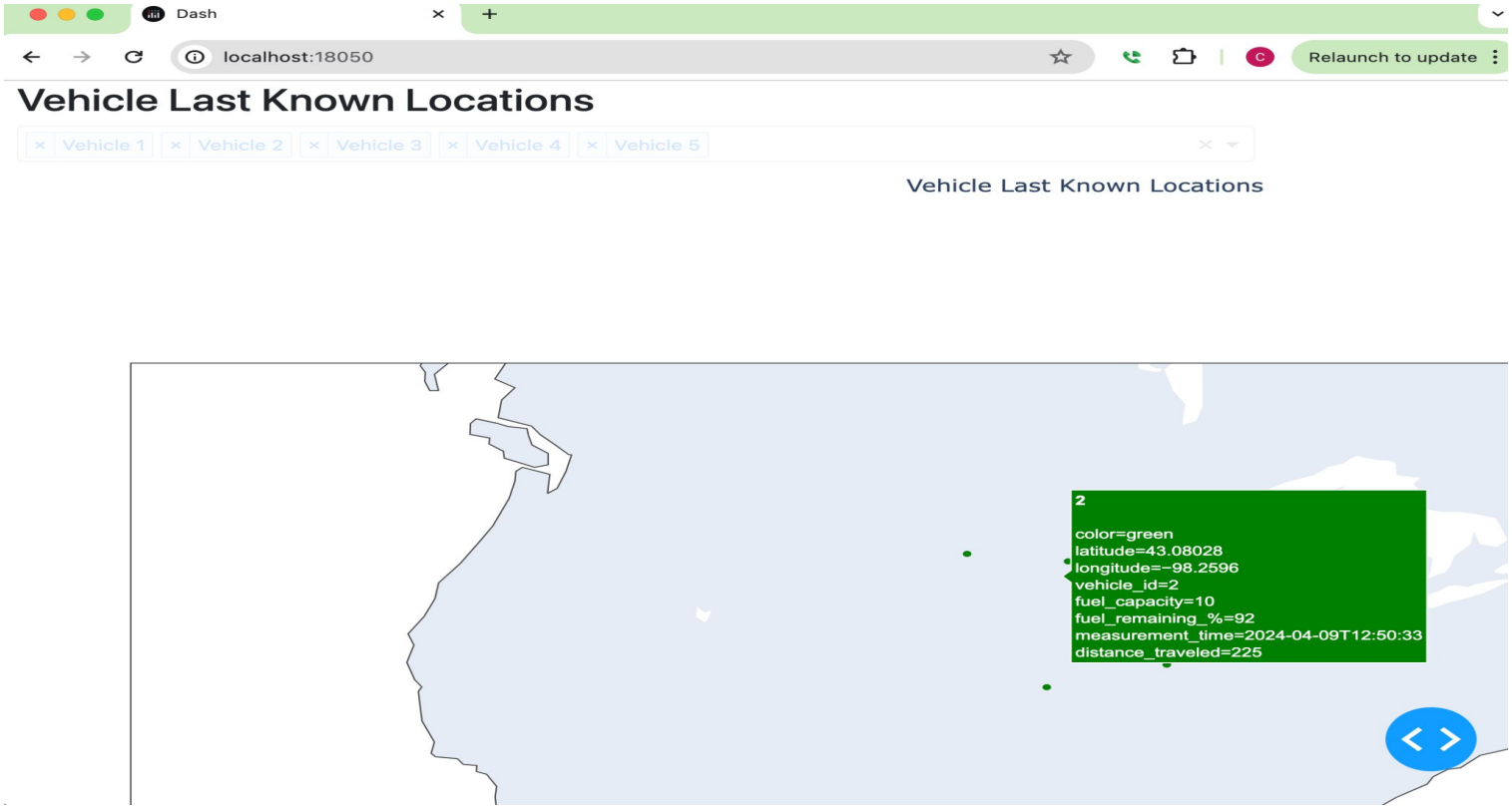
16. Real Time Analytics

- Functionality:
  - Represents a real-time location tracking application using Dash, a Python web framework.
- Features:
  - Utilizes Dash framework along with Plotly for creating interactive data visualizations.
  - Loads vehicle data from a CSV file (`trips.csv`) containing various parameters such as vehicle ID, measurement time, latitude, longitude, fuel capacity, fuel remaining percentage, distance traveled, and color.
  - Provides a dropdown menu to select individual vehicles or all vehicles at once.
- Use Case:
  - Simulates a scenario for real-time monitoring of multiple vehicles' locations, enabling users to track their movements, analyze fuel levels, and monitor distances traveled.

TO-DO:

Adopt the Kafka-Consumer App (under the draft_app_kafka_consumer_producer) to consume real-time data .

16. Real Time Analytics



Next

- As an orchestration tool evaluate Mage vs Airflow
- Improve logging
- Replace cron jobs with the orchestration tool : Mage
- Implement Kafka Consumer module in real time analytics application
- Issue: docker cannot resolve minIO service name as DNS. AWS endpoint url needs to be manually modified in the docker compose file
- Implement real data models in dbt and demonstrate full capability
  - Tests
  - Lineage
  - Dictionary
- Evaluate other open source analytics tools, ex: Lightdash

Questions?