# ETL END TO END IMPLEMENTATION

CPSC 531

Aviya Singh & Kavitha Dasaratha

## Table of Contents

## Introduction

In our bustling city, there's a ton of information buzzing around every second. From traffic updates and weather changes to emergencies happening in real-time, keeping track of everything can be a challenge. That's where our project comes in!

We are simulating data to gather information about the vehicle; weather report like sunny, rainy, snowy; traffic related data if there is an emergency, if there is an acciedent happened on the road or traffic police arrest happened; GPS data showing the movement of the vehicle based on the longitude and latitude of the vehicle.

We are using Apache Kafka and Apache Spark to perform data streaming on the AWS , we're able to process and analyze this data as it comes in, giving us valuable insights into what's happening in our city at any given moment.

Whether it's monitoring traffic flow, predicting weather patterns, or responding to emergencies swiftly, our Smart City Data Streaming Project helps city officials make informed decisions to keep our city running smoothly and safely.

Join us as we take a closer look at how our system works and the impact it's making on our city's daily life. Let's dive into the world of real-time data streaming and discover the secrets behind our smart city!

## Architecture

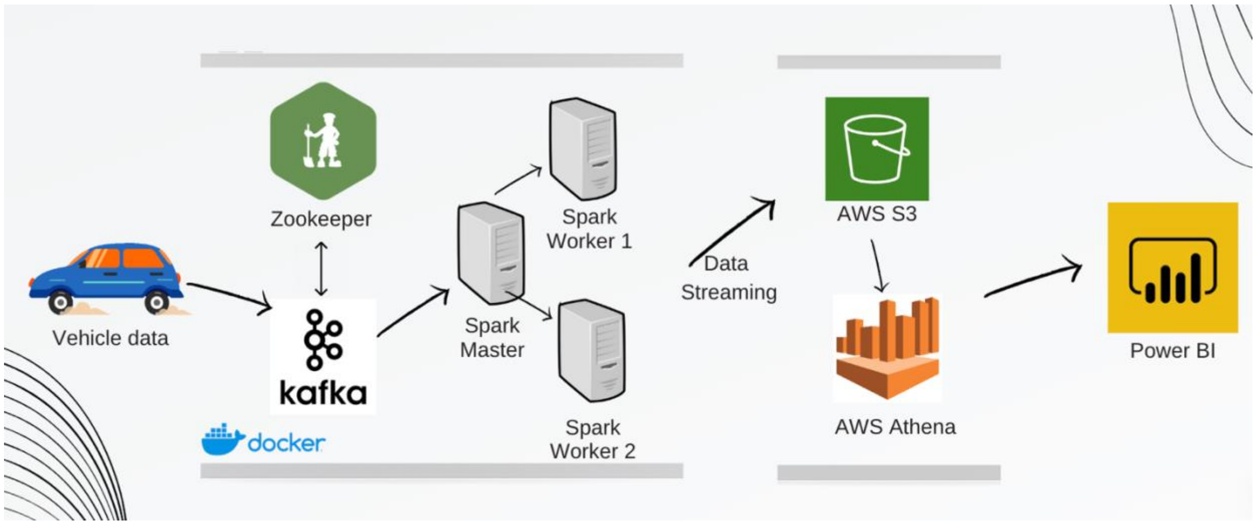
The below architecture flow involves ingesting real-time data from various sources, processing and analyzing it using Apache Kafka and Spark, and storing the processed data in AWS S3 for further analysis and visualization using tools like Power BI. This architecture enables the creation of a scalable, real-time data processing pipeline capable of handling large volumes of streaming data.

Data Extraction, Transformation and Loading Flow

- **Data Ingestion and Processing**:
  - Data Sources: Various data sources such as sensors, cameras, and emergency services generate real-time data.
  - Dockerized Environment: The entire system runs within Docker containers, ensuring consistency and portability.
  - Apache Kafka: Acts as a central hub for streaming data, where different types of data are published to Kafka topics.
  - Producers: Each Python script acts as a producer, generating and publishing data to specific Kafka topics.

- Consumers: Other components of the system, such as Spark applications, can subscribe to these Kafka topics to consume and process the data.

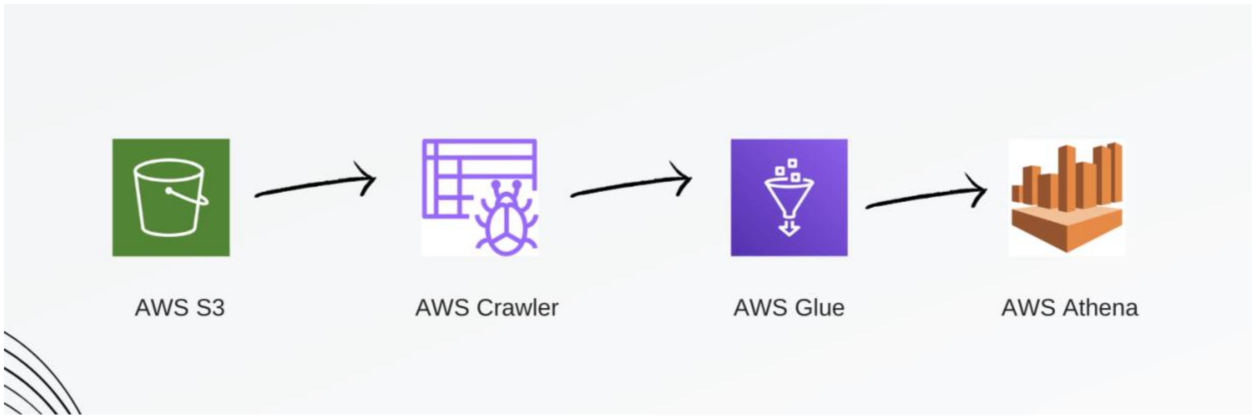
- **Data Processing and Analysis:**
  - Apache Spark: Spark applications are responsible for processing and analyzing the streaming data received from Kafka topics.
  - Spark Streaming: Utilizes Spark's streaming capabilities to process data in real-time, applying transformations and aggregations as needed.
  - Schema Definition: Data schemas are defined for each type of data (e.g., vehicle data, GPS data) to structure and interpret the incoming data streams accurately.
  - Data Processing Logic: Python functions within Spark applications define the logic for processing and enriching the incoming data streams, generating insights or performing actions based on the processed data.



- **Data Storage and Visualization:**
  - AWS S3 Bucket: Processed data is stored in an AWS S3 bucket, providing a scalable and durable storage solution.
  - Parquet Format: Data is stored in Parquet format, optimized for efficient storage and retrieval.
  - Streaming Writers: Spark applications write the processed data streams to Parquet files in the S3 bucket, ensuring data persistence and enabling further analysis.
  - Data Visualization Tools: Tools like Power BI can connect to the S3 bucket to visualize and analyze the stored data, providing insights and actionable information to end-users.

### AWS Flow

Firstly, we are simulating the vehicle data such as vehicle device id, location, timestamp, GPS location, and how the vehicle moves it will also talk about the weather conditions like if it is sunny, raining, or snowy in different GPS location, if there are any accidents, fire happened, or the vehicle is caught by the traffic police.



The above architecture gives an illustrative flow of how things are working in the AWS Dashboard.

AWS Crawler is being used to read the data from the AWS S3 Bucket to infer the schema and publish the data to AWS Glue to view the data in tabular form. Once we have the data on the AWS Glue, we can use AWS Athena to perform query on the data. We will see more in the Implementation, and how it looks.

## Tools

The tools that we are using in building the project are:

- **Docker Engine**: Docker is a containerization platform that allows us to package our applications and their dependencies into lightweight containers, ensuring consistency between development, testing, and production environments.

- **Zookeeper**: Zookeeper is a distributed coordination service that provides a centralized repository for configuration information and helps maintain consensus across nodes in a distributed system, essential for tasks like leader election and distributed synchronization.

- **Apache Kafka (Broker):** Kafka is a distributed streaming platform that allows us to publish, subscribe to, store, and process streams of records in real-time, acting as a highly scalable and fault-tolerant messaging system.

- **Apache Spark (Master and Worker nodes**): Spark is a distributed data processing engine that provides high-level APIs for building parallel applications, enabling efficient processing of large-scale data sets across a cluster of machines, with a master coordinating tasks and one or more workers executing computations.

- **AWS S3 Bucket**: Amazon Simple Storage Service (S3) is an object storage service that provides scalable storage for data objects in the cloud, offering high availability, durability, and low latency access to data, making it suitable for storing large volumes of structured and unstructured data.

- **AWS Crawler**: AWS Glue Crawler is an automated tool that scans and catalogs data stored in various data sources, including S3 buckets and databases, extracting schema information and creating metadata tables that can be used for querying and analysis.

- **AWS Glue**: AWS Glue is a fully managed extract, transform, and load (ETL) service that makes it easy to prepare and transform data for analytics, offering features like job scheduling, dependency management, and data cataloging, facilitating data integration and data processing tasks.

- **AWS Athena**: Amazon Athena is an interactive query service that allows users to analyze data directly from S3 using standard SQL queries, eliminating the need for managing infrastructure and providing fast query performance for ad-hoc and exploratory analysis tasks.

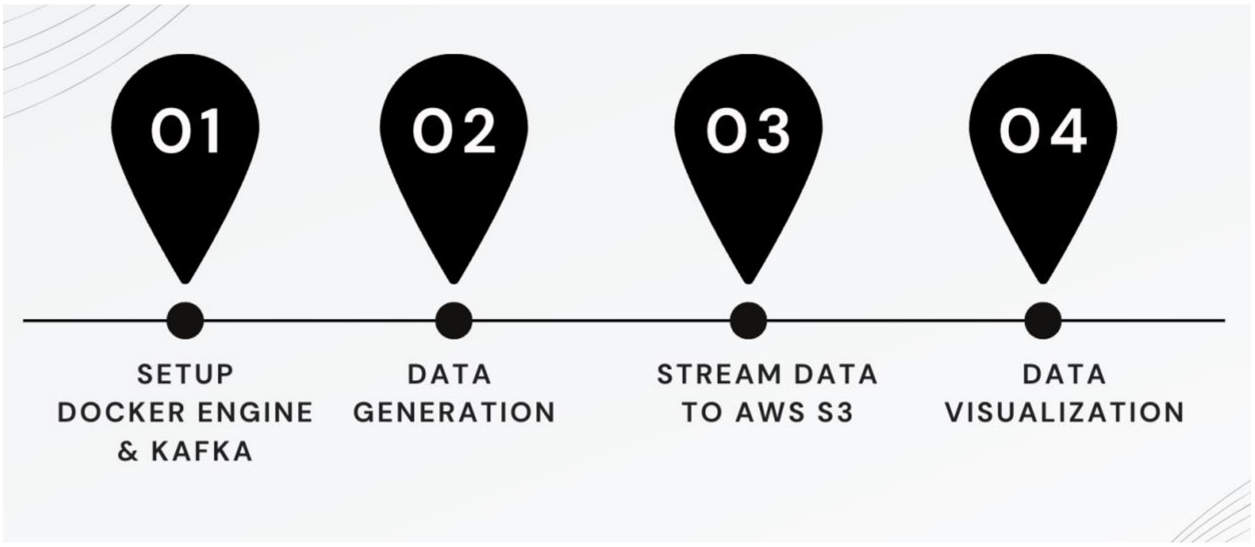
- **Power BI**: Power BI is a business analytics service provided by Microsoft that enables users to visualize and analyze data through interactive dashboards and reports, offering features like data visualization, data modeling, and collaboration tools for sharing insights across organizations.

## Implementation

The implementation was broken down into 4 phases:

Phase 1: Setup docker engine and Kafka
Phase 2: Data Generation
Phase 3: Stream data to the AWS S3 Bucket and perform transformation using AWS Crawler and query processing using AWS Glue

Phase 4: Data Visualization using Power BI



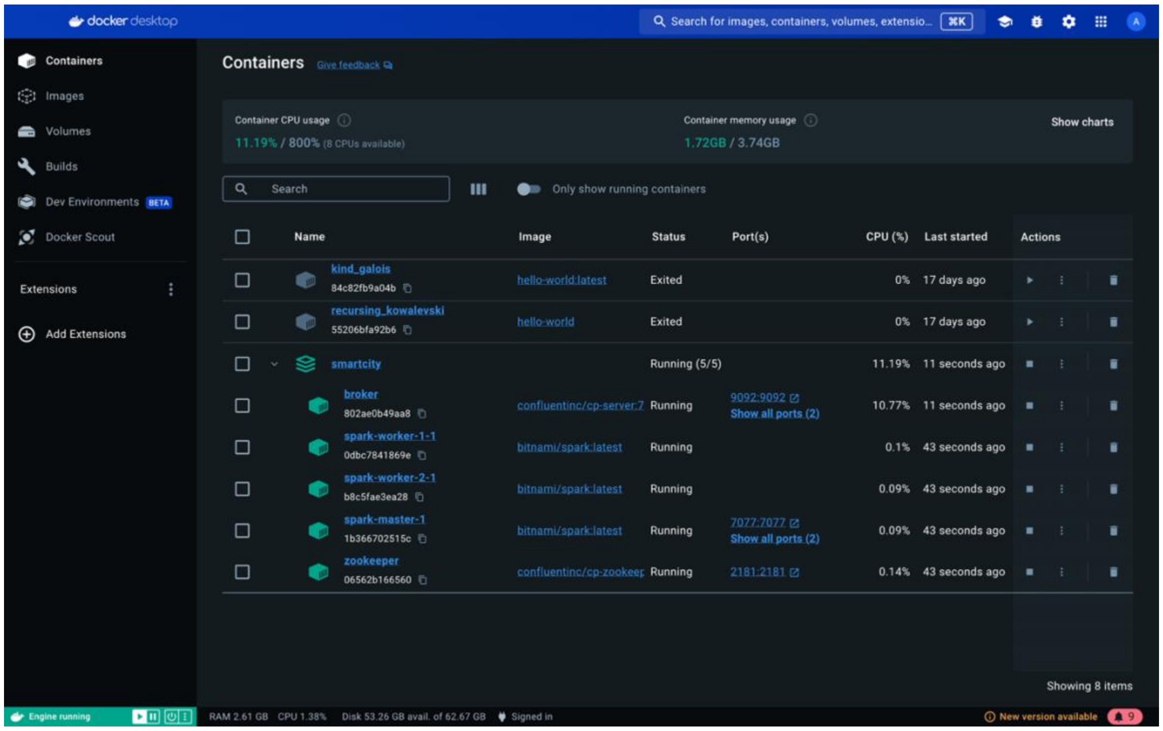
**Docker Installation**

- Download the installer using the download buttons at the top of the page, or from the release notes.
- Double-click Docker.dmg to open the installer, then drag the Docker icon to the **Applications** folder. By default, Docker Desktop is installed at /Applications/Docker.app.
- Double-click Docker.app in the **Applications** folder to start Docker.
- The Docker menu displays the Docker Subscription Service Agreement.

We are using docker-compose.yml to configure multi-container application for running Apache Kafka, Apache Spark, and Zookeeper services.

Let's break it down section by section:

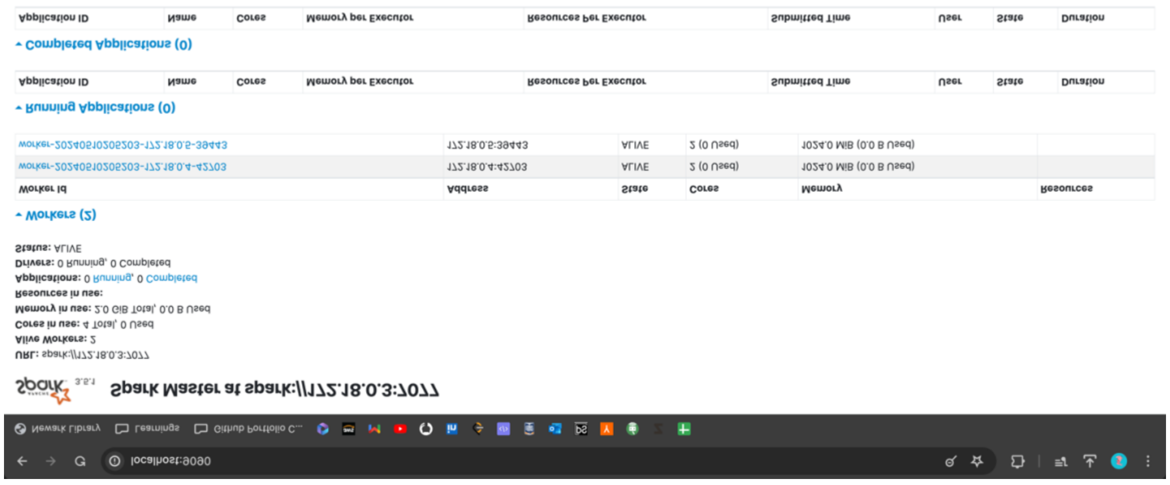
- **Version and Services**:
    - **Setting up Services**: We define several services within the docker-compose file, including Zookeeper, Kafka broker, Spark master, and multiple Spark worker nodes. These services are essential components of our distributed system.
    - **Container Configuration**: For each service, we specify the Docker image to use, any volumes to mount, and environment variables to configure the service.
    - **Network Configuration**: We define a custom network called "datamasterylab" to facilitate communication between the different services. This ensures that our services can interact with each other seamlessly.


Containers

| Name | Image | Status | Port(s) | CPU (%) | Last started |
|---|---|---|---|---|---|
| kind_galois 84c82fb9a04b | hello-world:latest | Exited | | 0% | 17 days ago |
| recursing_kowalevski 55206bfa92b6 | hello-world | Exited | | 0% | 17 days ago |
| smartcity | | Running (5/5) | | 11.19% | 11 seconds ago |
| broker 802ae0b49aa8 | confluentinc/cp-server:7 | Running | 9092:9092 Show all ports (2) | 10.77% | 11 seconds ago |
| spark-worker-1-1 0dbc7841869e | bitnami/spark:latest | Running | | 0.1% | 43 seconds ago |
| spark-worker-2-1 b8c5fae3ea28 | bitnami/spark:latest | Running | | 0.09% | 43 seconds ago |
| spark-master-1 1b366702515c | bitnami/spark:latest | Running | 7077:7077 Show all ports (2) | 0.09% | 43 seconds ago |
| zookeeper 06562b166560 | confluentinc/cp-zookeer | Running | 2181:2181 | 0.14% | 43 seconds ago |

Container CPU usage 11.19% / 800% (8 CPUs available)
Container memory usage 1.72GB / 3.74GB
Showing 8 items
RAM 2.61 GB CPU 1.38% Disk 53.26 GB avail. of 62.67 GB

- **Dependency Management**: We specify dependencies between services using the "depends_on" directive. For example, Spark workers depend on the Spark master, ensuring that the workers only start once the master is up and running.
- **Port Mapping**: We map container ports to host ports, allowing us to access services running inside Docker containers from outside the Docker environment. This is essential for interacting with our services and accessing web interfaces like Spark's UI.


Spark Master at spark://172.18.0.3:7077

**URL:** spark://172.18.0.3:7077
**Alive Workers:** 2
**Cores in use:** 4 Total, 0 Used
**Memory in use:** 2.0 GiB Total, 0.0 B Used
**Resources in use:**
**Applications:** 0 Running, 0 Completed
**Drivers:** 0 Running, 0 Completed
**Status:** ALIVE

Workers (2)

| Worker Id | Address | State | Cores | Memory | Resources |
|---|---|---|---|---|---|
| worker-20240510205203-172.18.0.4-42703 | 172.18.0.4:42703 | ALIVE | 2 (0 Used) | 1024.0 MiB (0.0 B Used) | |
| worker-20240510205203-172.18.0.5-39443 | 172.18.0.5:39443 | ALIVE | 2 (0 Used) | 1024.0 MiB (0.0 B Used) | |

Running Applications (0)

| Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|---|

Completed Applications (0)

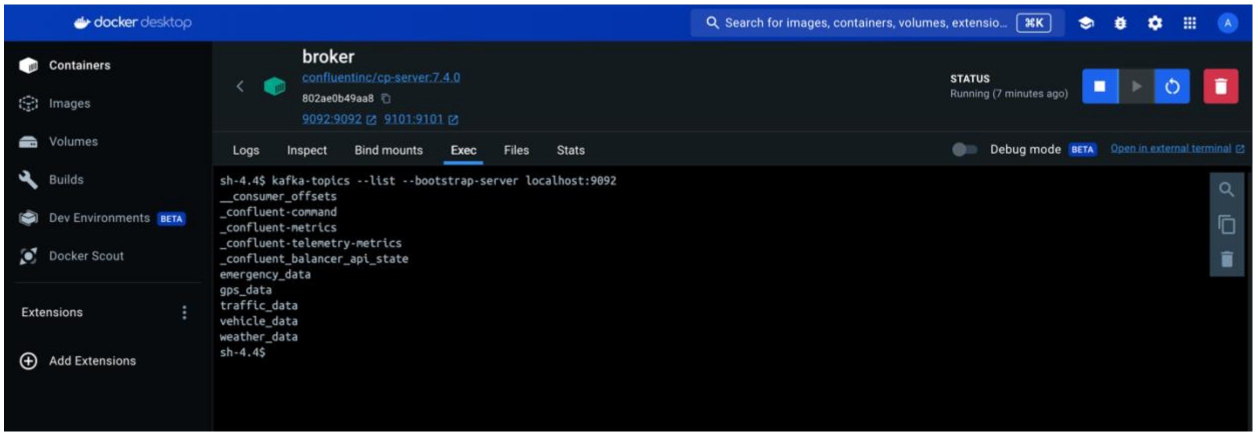
| Application ID | Name | Cores | Memory per Executor | Resources Per Executor | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|---|

## Data Generation

The main.py file is responsible for simulating the generation of various types of data that mimic real-world scenarios. This simulated data serves as a placeholder for the actual data that would be received from sensors, cameras, and other sources in a real smart city environment.

**Data Generation Logic**:

- The main.py file contains Python functions that generate simulated data for different aspects of smart city operations, including vehicle movement, GPS coordinates, traffic camera snapshots, weather conditions, and emergency incidents.
- Each function within main.py utilizes Python's random module to generate realistic values for attributes such as speed, temperature, and location.



```
sh-4.4$ kafka-topics --list --bootstrap-server localhost:9092
__consumer_offsets
_confluent-command
_confluent-metrics
_confluent-telemetry-metrics
_confluent_balancer_api_state
emergency_data
gps_data
traffic_data
vehicle_data
weather_data
sh-4.4$
```

- **Simulated Data Types**:
  - **Vehicle Data**: Simulates the movement of vehicles within the city, including attributes such as speed, location, and vehicle type.
  - **GPS Data**: Generates GPS coordinates for vehicles, along with speed and direction information.
  - **Traffic Camera Data**: Simulates snapshots from traffic cameras positioned across the city, providing visual insights into traffic conditions.
  - **Weather Data**: Generates weather conditions such as temperature, precipitation, wind speed, and humidity at different locations within the city.
  - **Emergency Incident Data**: Simulates emergency incidents such as accidents, fires, and medical emergencies, including details such as incident type, location, and status.

- **Realism and Randomness**:
    - While the data generated by main.py is simulated, efforts have been made to ensure realism by incorporating randomness and variability into the generated values.
    - Randomization techniques such as random.uniform() and random.choice() are used to introduce variability in attributes such as speed, temperature, and incident type, reflecting the unpredictable nature of real-world events.
    - **Usage and Integration**:
    - The simulated data generated by main.py serves as input for downstream processes within the smart city data streaming pipeline.
    - This simulated data allows developers and data engineers to test and validate the functionality of the data processing and analysis components without relying on real-time data sources.

Data Streaming to AWS S3 Bucket

The spark-city.py file is responsible for orchestrating the streaming of data from various sources to AWS (Amazon Web Services) for further processing and analysis. This file utilizes Apache Spark's capabilities to ingest, transform, and store streaming data in AWS S3, enabling scalable and efficient data processing in a distributed environment.

- **Apache Spark Streaming**:
    - The spark-city.py file leverages Apache Spark's streaming capabilities to ingest real-time data streams from Kafka topics.
    - Spark Streaming allows for the processing of data in micro-batches, enabling near-real-time analysis and aggregation of streaming data.

- **Data Schema Definition**:
    - Data schemas are defined within spark-city.py to structure and interpret the incoming data streams accurately.
    - These schemas define the fields and data types for each type of data being streamed, ensuring consistency and compatibility with downstream processing logic.

- **Kafka Integration**:
    - Spark-city.py integrates with Apache Kafka to consume streaming data from Kafka topics.
    - Kafka topics act as intermediate data channels, facilitating the seamless transfer of data between producers and consumers within the distributed system.

- **Data Processing and Enrichment**:
    - Spark-city.py applies data processing and enrichment logic to the incoming data streams, transforming raw data into structured, actionable insights.
    - This processing may include filtering, aggregation, joining, and other data manipulation operations to extract relevant information from the streaming data.

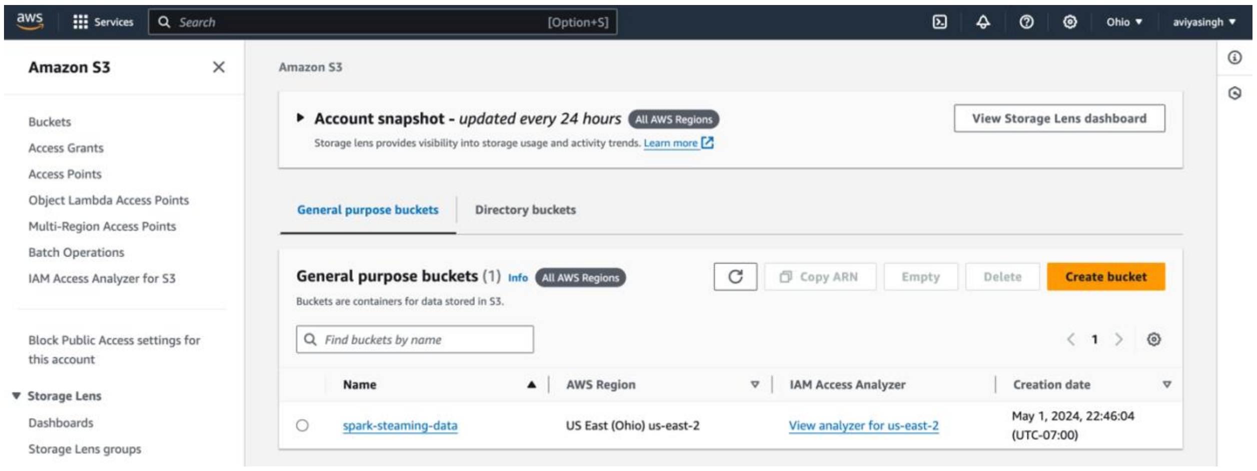
- **Data Storage in AWS S3**:
    - Processed data streams are stored in AWS S3 buckets, providing a scalable and durable storage solution for large volumes of streaming data.
    - Spark-city.py writes the processed data streams to Parquet files in the S3 bucket, leveraging the optimized storage format for efficient storage and retrieval.

- **Checkpoints and Reliability**:
    - Checkpointing mechanisms are employed within spark-city.py to ensure fault tolerance and data reliability in case of failures or system crashes.
    - Checkpoints enable Spark to resume streaming from the last known state, ensuring data integrity and consistency in the event of disruptions.

## Data Visualization

AWS Crawler scans the contents of an AWS S3 bucket, infers the schema of the data files, and creates metadata tables in the AWS Glue Data Catalog. AWS Glue uses these metadata tables to perform ETL tasks on the data, while AWS Athena allows users to query the data stored in S3 using standard SQL queries, leveraging the inferred schema and metadata information. This workflow enables seamless data discovery, processing, and querying on data stored in AWS S3, making it accessible for analysis and insights using Athena.

- **AWS S3 Bucket**:
    - The AWS S3 bucket serves as a storage repository for various types of data, including structured and semi-structured data files stored in formats like CSV, JSON, or Parquet.



Amazon S3

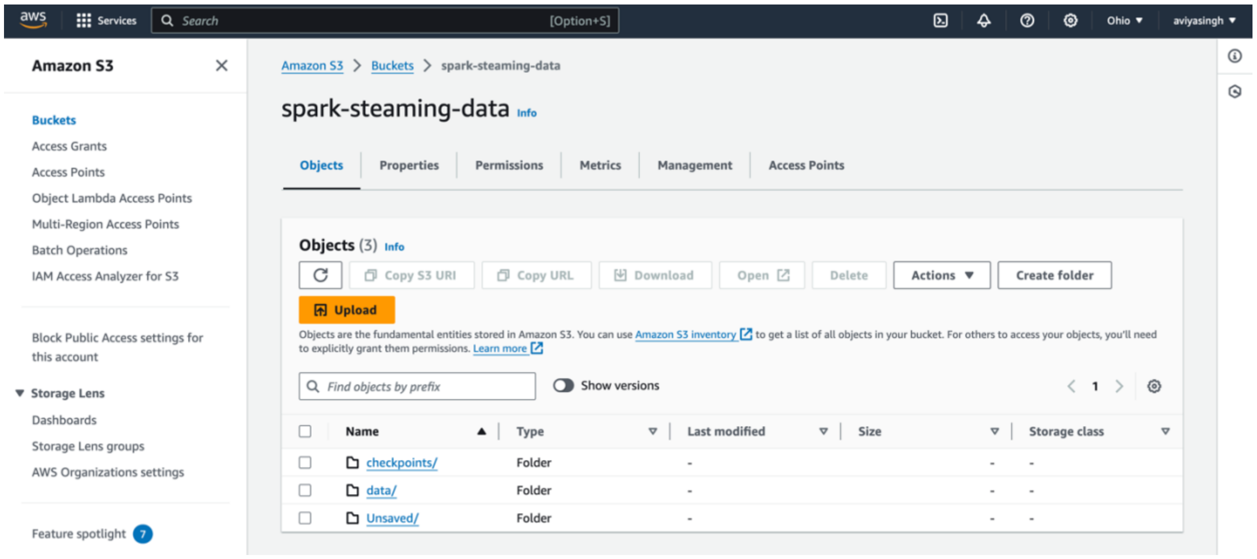
**Account snapshot - *updated every 24 hours*** All AWS Regions

Storage lens provides visibility into storage usage and activity trends. Learn more

**General purpose buckets** (1) Info All AWS Regions

Buckets are containers for data stored in S3.

| Name | AWS Region | IAM Access Analyzer | Creation date |
|---|---|---|---|
| spark-steaming-data | US East (Ohio) us-east-2 | View analyzer for us-east-2 | May 1, 2024, 22:46:04 (UTC-07:00) |



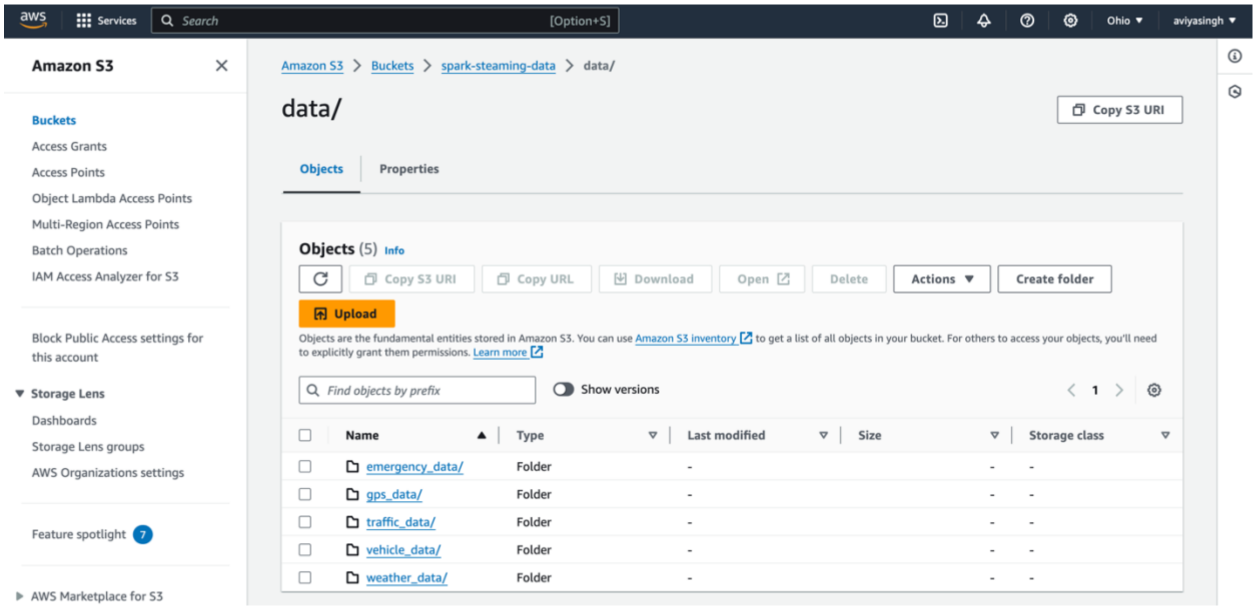
Amazon S3 > Buckets > spark-steaming-data

**spark-steaming-data** Info

**Objects** (3) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

| Name | Type | Last modified | Size | Storage class |
|---|---|---|---|---|
| checkpoints/ | Folder | - | - | - |
| data/ | Folder | - | - | - |
| Unsaved/ | Folder | - | - | - |



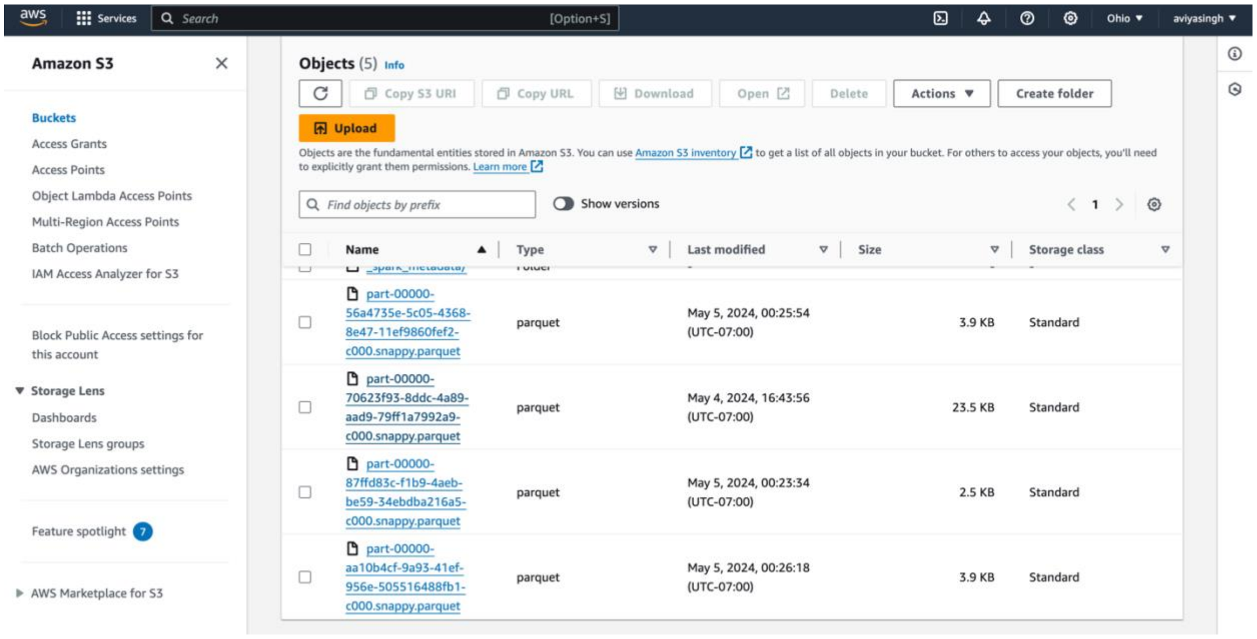
Amazon S3 > Buckets > spark-steaming-data > data/

**data/**

**Objects** (5) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

| Name | Type | Last modified | Size | Storage class |
|---|---|---|---|---|
| emergency_data/ | Folder | - | - | - |
| gps_data/ | Folder | - | - | - |
| traffic_data/ | Folder | - | - | - |
| vehicle_data/ | Folder | - | - | - |
| weather_data/ | Folder | - | - | - |

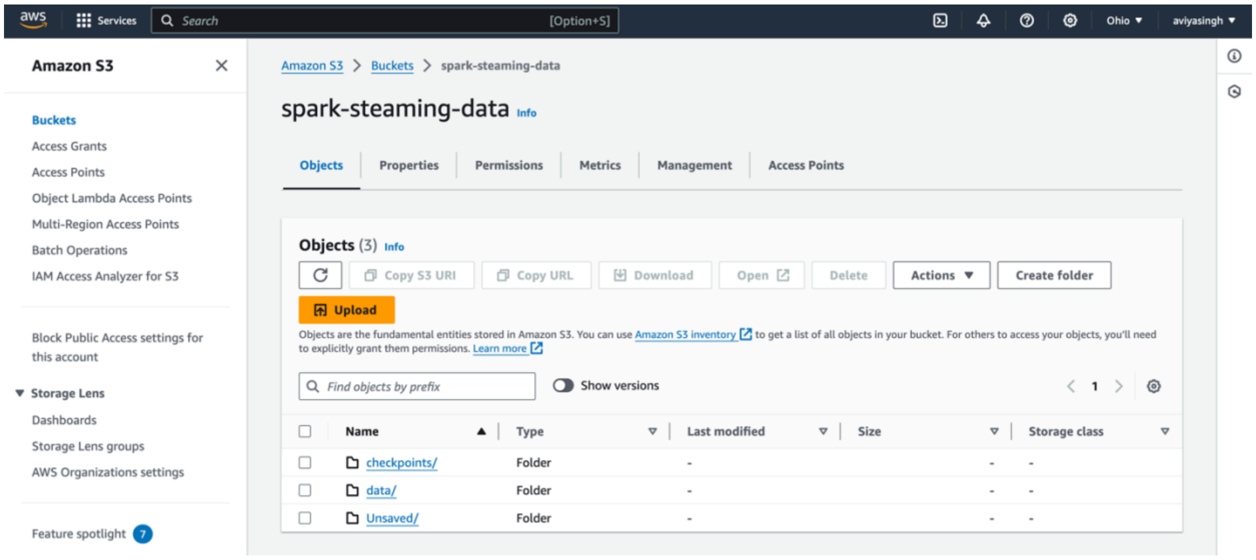


Objects (5) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

| Name | Type | Last modified | Size | Storage class |
|---|---|---|---|---|
| _spark_metadata/ | Folder | - | - | - |
| part-00000-56a4735e-5c05-4368-8e47-11ef9860fef2-c000.snappy.parquet | parquet | May 5, 2024, 00:25:54 (UTC-07:00) | 3.9 KB | Standard |
| part-00000-70623f93-8ddc-4a89-aad9-79ff1a7992a9-c000.snappy.parquet | parquet | May 4, 2024, 16:43:56 (UTC-07:00) | 23.5 KB | Standard |
| part-00000-87ffd83c-f1b9-4aeb-be59-34ebdba216a5-c000.snappy.parquet | parquet | May 5, 2024, 00:23:34 (UTC-07:00) | 2.5 KB | Standard |
| part-00000-aa10b4cf-9a93-41ef-956e-505516488fb1-c000.snappy.parquet | parquet | May 5, 2024, 00:26:18 (UTC-07:00) | 3.9 KB | Standard |

- **AWS Crawler**:
  - The AWS Glue Crawler is configured to periodically scan the contents of the AWS S3 bucket.
  - When triggered, the AWS Crawler automatically detects new data files or changes to existing files in the S3 bucket.
  - The crawler analyzes the data files and infers the schema (i.e., the structure and data types) based on the contents of the files.



Amazon S3 > Buckets > spark-steaming-data

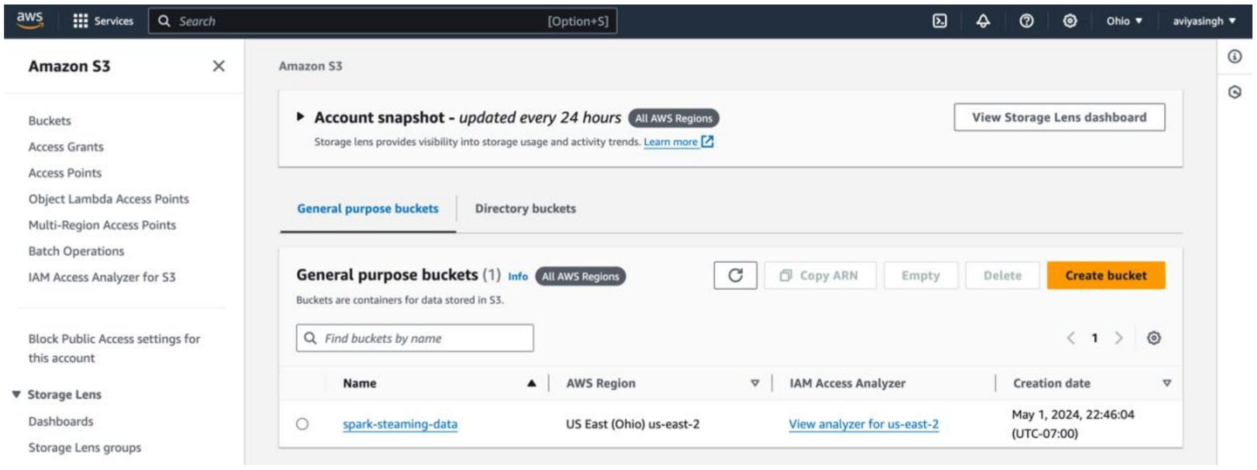
spark-steaming-data Info

Objects (3) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

| Name | Type | Last modified | Size | Storage class |
|---|---|---|---|---|
| checkpoints/ | Folder | - | - | - |
| data/ | Folder | - | - | - |
| Unsaved/ | Folder | - | - | - |

- **Inferred Schema**:
  - Once the AWS Crawler completes its analysis, it creates metadata tables in the AWS Glue Data Catalog.
  - These metadata tables contain information about the inferred schema of the data files, including column names, data types, and partitioning keys.



Account snapshot - *updated every 24 hours* All AWS Regions

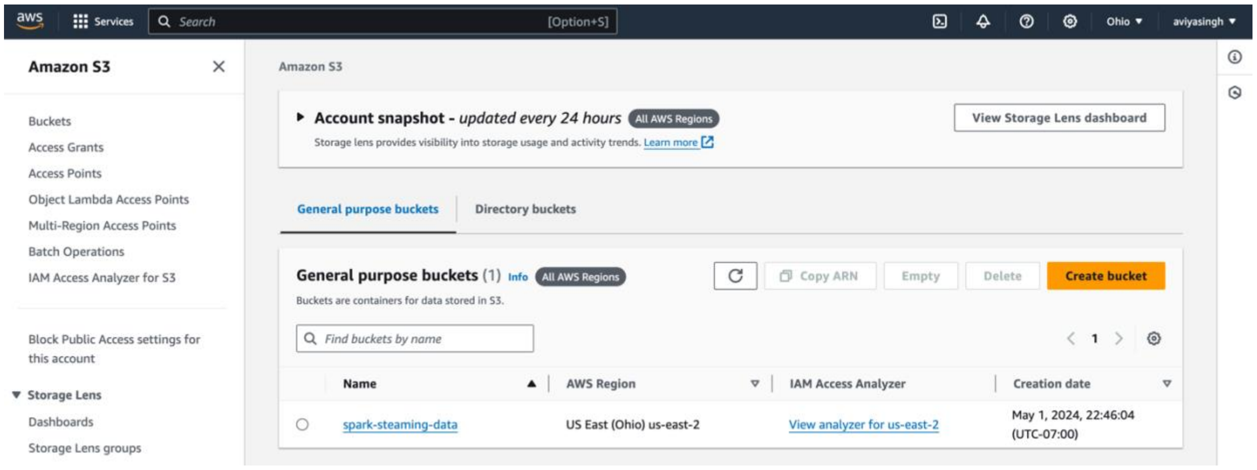
Storage lens provides visibility into storage usage and activity trends. Learn more

General purpose buckets (1) Info All AWS Regions

Buckets are containers for data stored in S3.

| Name | AWS Region | IAM Access Analyzer | Creation date |
|---|---|---|---|
| spark-steaming-data | US East (Ohio) us-east-2 | View analyzer for us-east-2 | May 1, 2024, 22:46:04 (UTC-07:00) |

- **AWS Glue**:
  - AWS Glue is a fully managed extract, transform, and load (ETL) service that uses the metadata tables created by the AWS Crawler to perform data processing tasks.
  - AWS Glue can run ETL jobs to clean, transform, and enrich the data as needed before storing it in a target data store or format.



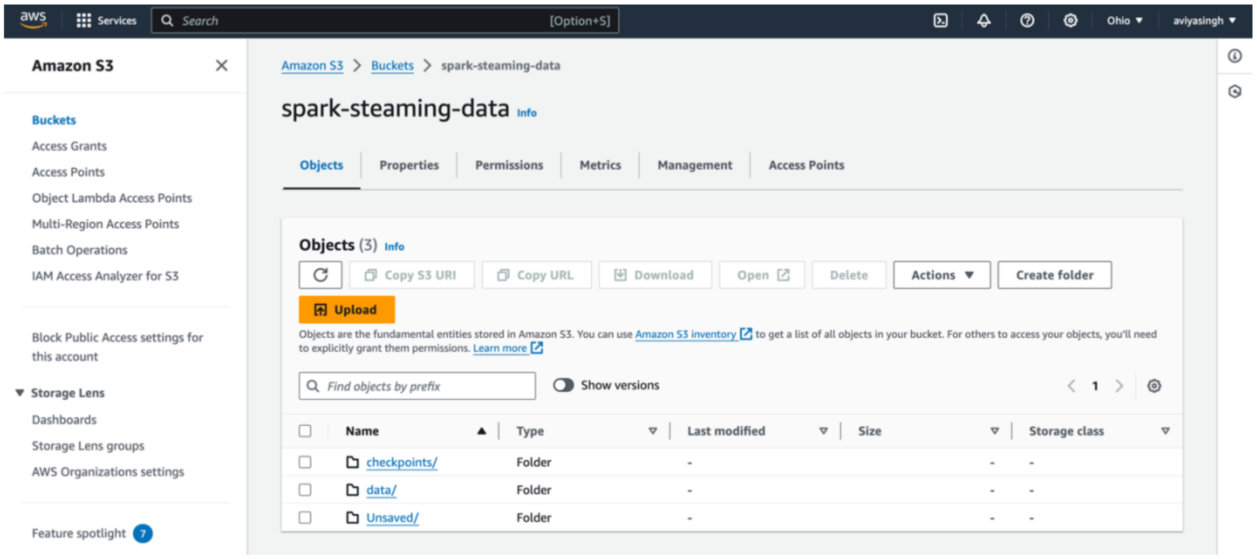
Account snapshot - *updated every 24 hours* All AWS Regions

Storage lens provides visibility into storage usage and activity trends. Learn more

General purpose buckets (1) Info All AWS Regions

Buckets are containers for data stored in S3.

| Name | AWS Region | IAM Access Analyzer | Creation date |
|---|---|---|---|
| spark-steaming-data | US East (Ohio) us-east-2 | View analyzer for us-east-2 | May 1, 2024, 22:46:04 (UTC-07:00) |



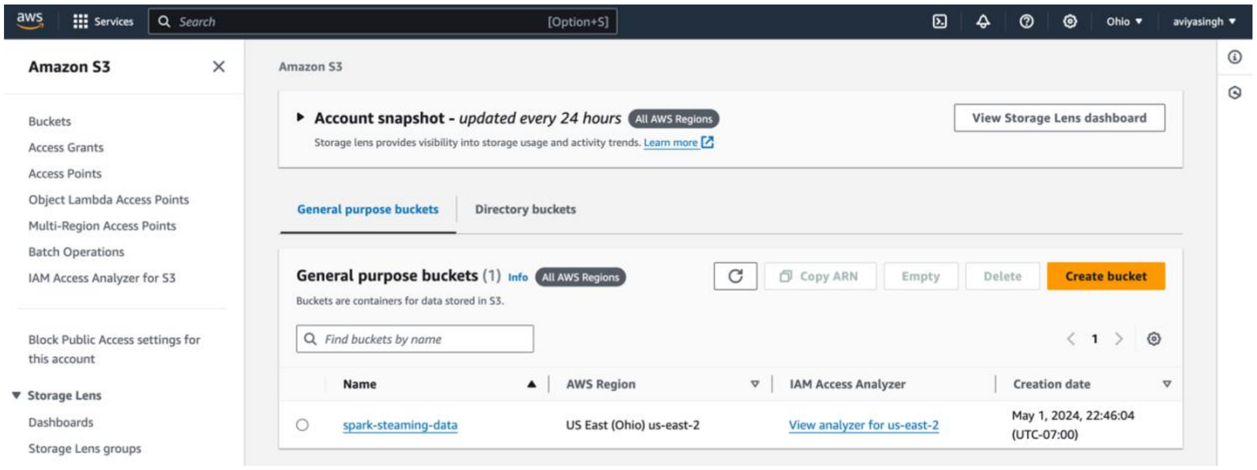
spark-steaming-data Info

Objects (3) Info

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

| Name | Type | Last modified | Size | Storage class |
|---|---|---|---|---|
| checkpoints/ | Folder | - | - | - |
| data/ | Folder | - | - | - |
| Unsaved/ | Folder | - | - | - |

- **Data Querying with AWS Athena**:
  - AWS Athena is a serverless interactive query service that allows users to query data stored in AWS S3 using standard SQL.
  - Athena uses the metadata tables created by AWS Glue to understand the schema of the data and perform SQL queries on it.
  - Users can execute ad-hoc SQL queries using the Athena console or through API calls, without the need to provision or manage any infrastructure.



Account snapshot - *updated every 24 hours* All AWS Regions

Storage lens provides visibility into storage usage and activity trends. Learn more

General purpose buckets (1) Info All AWS Regions

Buckets are containers for data stored in S3.

| Name | AWS Region | IAM Access Analyzer | Creation date |
|---|---|---|---|
| spark-steaming-data | US East (Ohio) us-east-2 | View analyzer for us-east-2 | May 1, 2024, 22:46:04 (UTC-07:00) |

## Execution Steps

- Firstly setup docker engine, run *docker compose up* command to setup multi-container application for running Apache Kafka, Apache Spark, and Zookeeper services.
- Secondly, execute the main.py file to simulate the vehicle data

- Lastly, execute the spark-city.py file to inject the data to aws s3 bucket using:

```
docker exec -it smartcity-spark-master-1 spark-submit \
--master spark://spark-master:7077 \
--packages org.apache.spark:spark-sql-kafka-0-
10_2.13:3.5.1,org.apache.hadoop:hadoop-aws:3.3.6,com.amazonaws:aws-java-sdk-
s3:1.11.469 \
jobs.spark_city.MainClass
```

## Github Location of the Project

https://github.com/Aviya-Singh/SmartCity

## Data Visualization

After processing and transforming data in AWS Glue, we exported it to CSV format, making it accessible for analysis in Power BI. Utilizing Power BI's data connectivity features, we established a connection to the CSV file stored locally or on a cloud-based storage service like AWS S3.

Once connected, Power BI's intuitive interface allowed us to import the CSV data and create visualizations, including bar graphs, with ease. We selected the relevant fields from the CSV data to populate the bar graph's axes, such as categories for the X-axis and numerical values for the Y-axis.

Using Power BI's drag-and-drop functionality, we quickly designed the bar graph, customizing its appearance with color schemes, labels, and titles to enhance readability and visual appeal. By configuring filters and slicers, we provided interactivity to the bar graph, allowing users to dynamically explore the data and drill down into specific segments.

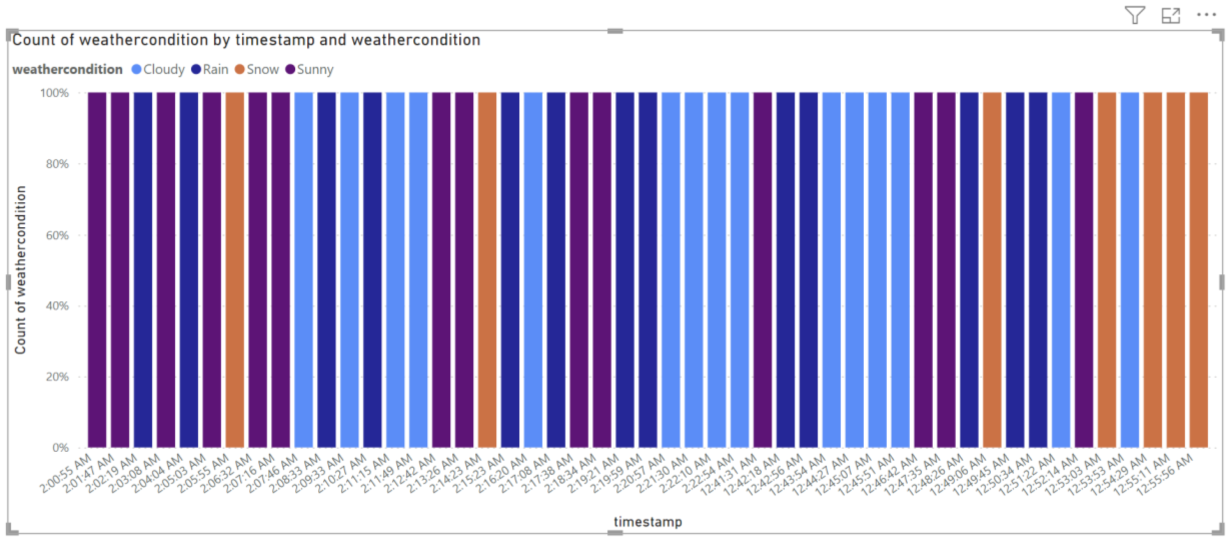
Upon completion, we published the Power BI report containing the bar graph to a secure workspace or shared it with stakeholders via email or a web link. This enabled decision-makers to gain actionable insights from the data represented in the bar graph, facilitating informed decision-making and driving business value.

We have projected three visualization which illustrate:
- Weather report
- Vehicle status and timestamp
- Vehicle status if there is any accident, fire, medical, police and if has been resolved or active

**Visual Report I**: The weather report at different intervals of the vehicle, if it is cloudy, snowy, rainy, or sunny. It also shows the average precipitation and the median of the temperature.

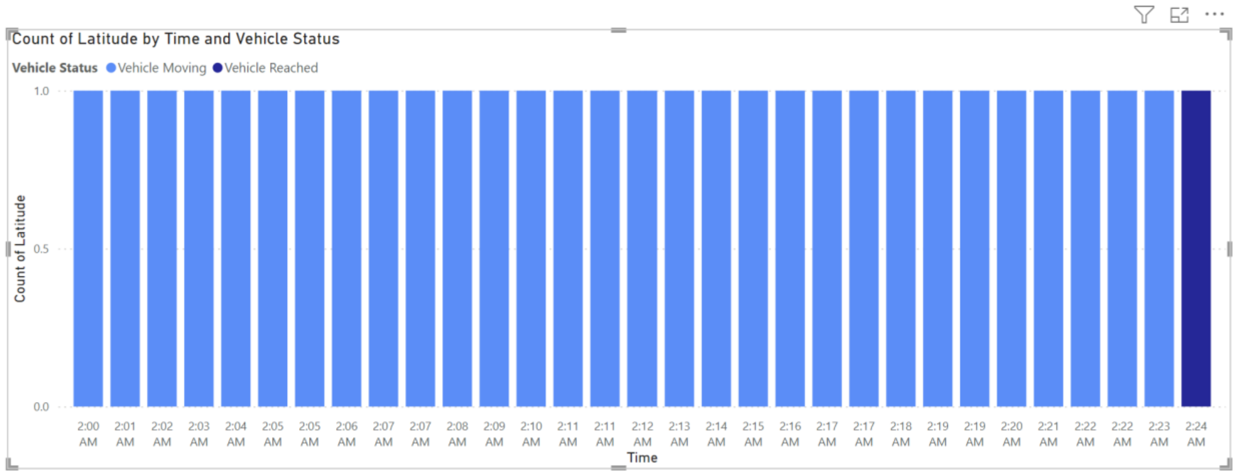
10.82
Average of precipitation

12.86
Median of temperature

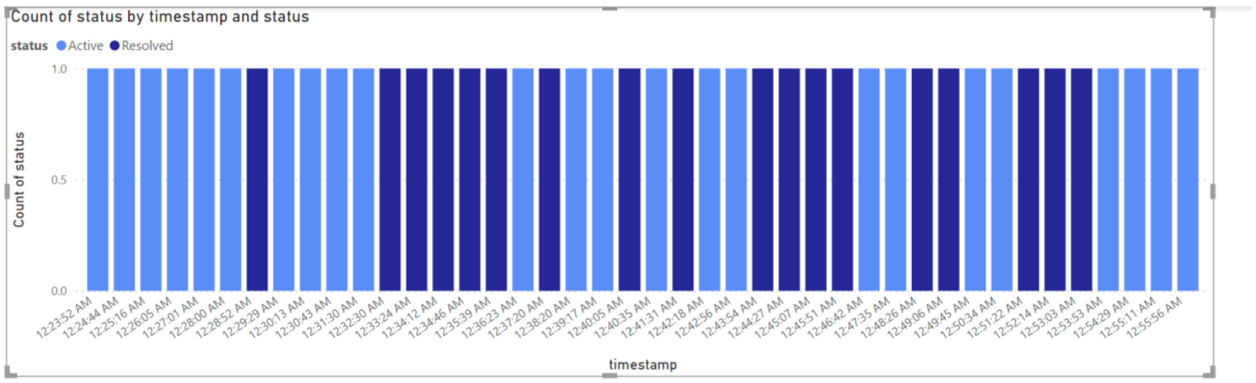


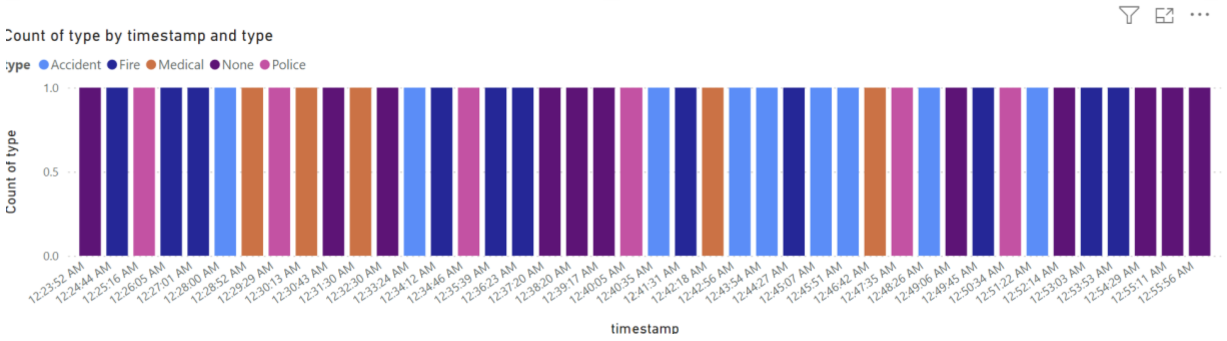
**Visual Report II**: The status of the vehicle at different timestamps. The light blue bar shows that the vehicle is still moving and the dark blue bar shows that the vehicle reached the destination at 2:24 AM.

24.21
Average of speed



**Visual Report III:** The later visualization shows if that if there was an accident or fire or a medical emergency or if police is chasing. All these are being displayed in different colors like lightblue for accidents, dark blue for fire, orange for medical, pink for police and purple for none. And the first bar graph show if type of issue has been resolved or still active.





## Conclusion

In conclusion, the Smart City Data Streaming and Analysis project successfully demonstrated the integration of cutting-edge technologies to create a scalable and efficient solution for processing real-time data in smart city environments. By leveraging Docker, Apache Kafka, Apache Spark, and AWS services, we established a robust data pipeline capable of handling diverse data streams and empowering stakeholders with actionable insights. This project underscores the importance of data-driven decision-making in modern urban management, paving the way for smarter and more sustainable cities of the future.