UrbanFlow - Project plan and description

Michael Mayrhofer (12122564), Julian Zeilinger (12122540), Andrej Kapusta (12427660), Dániel Hajós (12432109), and Sebastian Klaus (12120487)

TU Wien - 2025W

1 Project overview

The goal of the project is to create a system that ingests real-time vehicle traffic information. This data is then used to calculate various metrics. The data will be sourced from simulated IoT sensors that report traffic information for a given location. A simple web interface will also be provided to visualize the results.

2 Reasons for a Serverless Architecture

A serverless architecture is ideal for this scenario, as traffic data is typically very irregular and difficult to predict.

- **Irregular workloads (bursty workloads):** Traffic volume is not constant. There are clear peaks (e.g. during rush hour) and long periods of very low activity (e.g. at night) [2]. With a traditional, server-based architecture, we would have to provision server capacity for the maximum peak load. This would mean that expensive resources would remain unused most of the time. Serverless platforms avoid this problem by automatically adjusting computing power to actual demand in real time. This ensures performance during peak loads without paying for idle time.
- **Scale-to-zero economics:** A key economic advantage is the 'scale-to-zero' capability [1]. During periods without data traffic, the system does not process any data, and computing resources scale down to zero. With the pay-per-use model, this means that there are no costs for unused computing time. This makes this approach a highly efficient solution for event-driven workloads such as ours [1].

3 Intended Usage and Main Stakeholders

3.1 Intended Usage

The primary purpose of the system is to provide real-time insight into urban traffic conditions based on data streamed from IoT sensors. The software is intended to support real-time monitoring and enable traffic analytics.

- **Real-Time Traffic Monitoring**
  Users interact with a dashboard that visualizes the current traffic conditions on a city map. The sensor locations are displayed with a congestion indicator, allowing users to assess traffic densities, identify bottlenecks, or track emerging incidents.
- **Traffic Analytics and Metrics Evaluation**
  The system is intended to compute derived metrics, for example, the *Congestion Index*, based on incoming sensor data. Analysts can use these metrics to study traffic patterns and evaluate the performance of the infrastructure.

3.2 Main Stakeholders

The system involves several stakeholders:

- **Traffic Operators** rely on accurate, low-latency traffic insights to monitor road conditions and optimize traffic strategies. Their primary concern is the reliability, availability, and accuracy of the data presented.
- **Traffic Analysts and Planners** use the processed traffic metrics to study long-term congestion patterns, evaluate the effectiveness of mobility projects, and plan infrastructure improvements. They value data quality, historical trend access, and the interpretability of metrics such as the *Congestion Index*.
- **Software Engineers** are responsible for maintaining and further developing the system. Their focus is on maintainability, scalability, cost efficiency, throughput, and the ease with which new features or data sources can be integrated.
- **End Users / The General Public** might not directly interact with the system; however, they benefit indirectly through improved traffic management and reduced congestion. The insights derived from the system can inform navigation apps, city planning, and public information services.

4 Architecture and Data Flow

As displayed in Figure 1 the cloud infrastructure is simulated entirely within a LocalStack environment and provisioned using Terraform to replicate a native AWS architecture without incurring public cloud costs. Thereby, the system ingests simulated sensor data into Amazon Kinesis Data Streams, which serves as a durable buffer to manage bursty traffic loads. This stream triggers stateless AWS Lambda functions (Python) that process batch records to calculate the Congestion Index and persist the results (with configured expiration) in Amazon DynamoDB for low-latency access. The visualization layer retrieves this data via Amazon API Gateway, serving a Angular frontend hosted on Amazon S3 and triggering a reader lambda to fetch the latest state from the database. The IoT sensor network is emulated by Python scripts that generate synthetic telemetry data, such as vehicle speed and count, and push these records to the ingestion layer at controlled, variable rates to realistically mimic bursty traffic patterns and rush hour spikes.

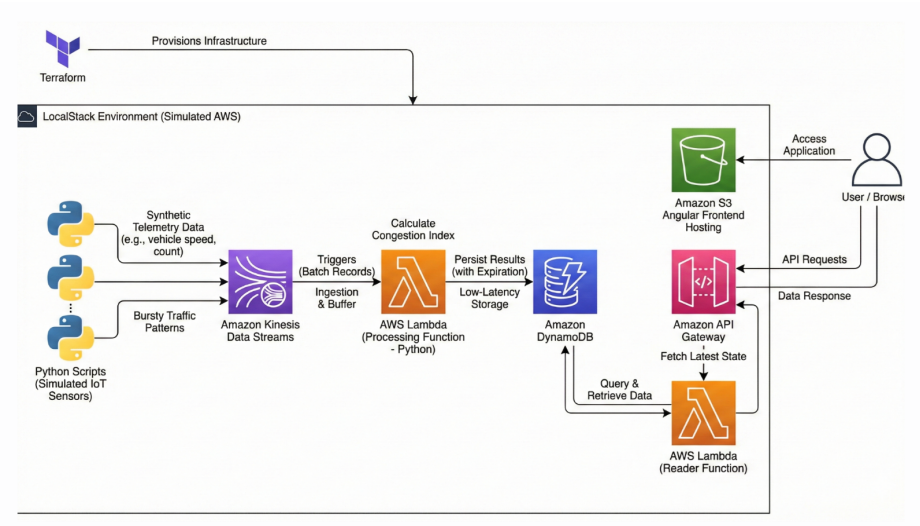

Fig. 1. Data flow visualization

Figure 2 illustrates the project's data flow. Sensors publish raw data to a durable stream buffer that decouples ingestion from data processing. The Congestion Index calculation (see appendix A) will be performed by a serverless function which also stores the results in a database. The output function will provide an API for the dashboards to receive and visualize data.

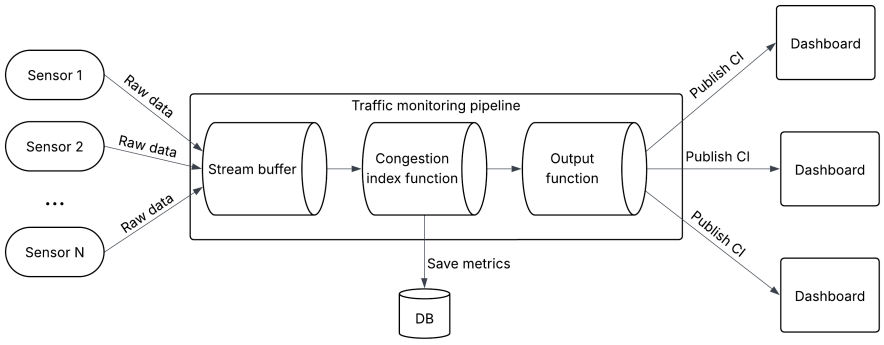

Fig. 2. Data flow visualization

5 Key Performance Indicators (KPIs)

The project's success will be measured against defined technical and business performance indicators.

Technical KPIs

- **Latency:** Since the project involves real-time traffic monitoring of the traffic, it is a critical objective to keep the latency as low as possible [3][4]. Otherwise, the actual traffic situation will not be reflected in time in the visualization. Although some latency is expected, it should not exceed a certain threshold making the application feel unresponsive.
  - **Measurement:** After a traffic-measuring function has been deployed in AWS's locally running LocalStack, the system can efficiently measure the invocation duration (e.g. through logs).
  - **Threshold:** To ensure that the system has low latency, an average latency is measured, and it should be less than $500\,ms$ under normal load. In addition, another measure is the 95^{th} percentile latency, which should be less than $1\,second$ in peak times.
- **System Availability:** To maintain real-time visibility of road conditions, we rely on uninterrupted data aggregation and processing. In native architectures, automated resilience and self-recovery are fundamental design principles, making system availability a critical indicator of overall reliability [3].
  - **Measurement:** Since LocalStack does not come with CloudWatch metrics like normal AWS does, this can only be measured by locally logging containers and checking if they can recover automatically and finish the assigned job. This number can then be aggregated to see how many lambda function invocations have succeeded. The total percentage can be obtained by calculating:

$$Availability = \frac{Successful\,Invocations}{Total\,Invocations} \tag{1}$$

  - **Threshold:** $Availability$ must be greater than 99% over 24 hours.
- **Data Freshness:** Similar to the latency metric, we want to ensure that users are not seeing stale traffic data. This metric is crucial in real-time analytics, where the value of information decreases as it becomes outdated [4].
  - **Measurement:** The easiest approach to ensure that the data are fresh is by introducing two timestamps. The first will be a timestamp of sensor ingestion (i.e., timestamp in Amazon Kinesis, automatically provided under the name *ApproximateArrivalTimestamp*[10]), and the second will be when the data are displayed on the dashboard for visualization.
  - **Threshold:** To ensure that the data are fresh enough, the difference between these two stamps should not exceed 30 seconds.

- **Scalability / Throughput:** Since traffic volume can be quite bursty, the workload can increase quickly during rush hour. Assessing throughput and scaling behavior of the application shows whether the system can remain responsive and reliable under fluctuating load conditions [3][4].
  - **Measurement:** To check if the system behaves correctly under load, a stress test with bursty workloads will be conducted. The test will simulate rush hour. The processing of these events is then measured.
  - **Threshold:** To ensure that the system can withstand rush hour, at least $1\,000\,events/sec$ should be processed.
- **Utilization Efficiency:** Inefficient event batching, excessively frequent function invocations, or redundant processing steps increase operational overhead without adding value. Tracking utilization efficiency allows the system to avoid unnecessary cost and wasted computation, aligning with the pay-as-you-go model [3][4].
  - **Measurement:** To check if batching is setup satisfactory, the ratio between the configured batch size and the real average records per invocation is proposed:

$$Batch\,Utilisation = \frac{Average\,Records\,per\,Invocation}{Configured\,Batch\,Size} \times 100 \tag{2}$$

  - **Threshold:** To ensure that the system batches the data efficiently, the average number of events processed should be close to 70% of the set batch size.

Business KPIs

- **Cost Optimisation:** The serverless solution offers a pay-as-you-go pricing model, which makes it more scalable, not necessitating any upfront capital to set up hardware for such a project. Also, because of spikes in traffic, which may occur during rush hours or public holidays, the necessity of scaling the app up and down makes it more cost-affordable, and thus fits it to the current demand. [5]
  - **Measurement:** In AWS the *Cost Explorer* can be used to check the billing metrics. However, as we use locally deployed stack (i.e. LocalStack), the costs are not taken into consideration. They can be, however, simulated using a formula:

$$Cost\,per\,1000\,events = \frac{Total\,Cost}{Events\,Processed/1000} \tag{3}$$

  - **Threshold:** For this measurement, we did not state a cost threshold.
- **Time-to-Insight:** The system must identify traffic incidents swiftly, since the success of the business case relies on real-time awareness [7].
  - **Measurement:** This implies also the data freshness technical KPI, so it can be measured equally.
  - **Threshold:** See the Data Freshness Technical KPI.

– **Business Continuity:** When evaluating traffic jams in real time, it is necessary to process events as soon as possible, even after a crash occurred [6].
  – **Measurement:** The measurement is done by several technical KPIs in the previous section, e.g. Data Freshness, System Availability and Scalability.
  – **Threshold:** Consists of several thresholds stated in technical KPIs.

5.1 Evaluating KPIs with Experiment

To verify that all defined thresholds are met and to ensure that the system operates as intended, the following experiment has been designed. A concise summary of the acceptance criteria (thresholds) and corresponding test types is provided in Table 1. It should be noted that this experiment does not evaluate the cost optimisation criterium, as such measurements are difficult to obtain in a LocalStack environment that relies on local hardware rather than a rented cloud infrastructure.

Setup

1. **Environment:** Set up the same topology in LocalStack with the help of Terraform to ensure similar conditions for validation.
2. **Function Configuration:** Configure the event source mapping by setting the batch size to 500.
3. **Generator:** Publish sensor events at controlled rates.

Steps

1. **Warm-up:** Run at a baseline rate (e.g. 500 events/sec) for 3 minutes to stabilise concurrency.
2. **Ramp-up:** Increase the events' rate to 10000 events/sec for another 5 minutes.
3. **Metrics Measurement:** Take the measured metrics such as timestamps, logging, and failure ratios.
   – **Latency:** Lambda processing completion (make an average out of the times).
   – **Throughput:** Processed events and the duration of the test (processed events/sec).
   – **Errors:** Failed vs. total invocations.
   – **Batch efficiency:** Average records per invocation.
   – **Data Freshness:** Take a sample of several events and their timestamps from the storage (ingestion time) and compare them with the visualisation timestamp.
4. **Acceptance Criteria Evaluation:** Evaluate the measured data with the set thresholds.

KPI	Measurement Method	Threshold
Latency	Invocation duration logs	<500 ms avg
System Availability	Success vs. failure ratio	>99% uptime
Data Freshness	Ingestion vs. visualisation timestamps	<30 s
Scalability/Throughput	Concurrent execution of, events/sec	>1000 events/sec
Utilization Efficiency	Batch size and redundant invocations	>70% efficiency

Table 1. Suggested Experiment Setup

A Congestion Index

To compute a Congestion Index the system uses a speed-based approach [9]. The index is based on comparing the actual traffic speed with free-flow (ideal) speed, providing an intuitive measure of congestion. It is calculated as follows:

$$CI = \frac{v_{free} - v_{avg}}{v_{avg}} \tag{4}$$

, where v_{free} is the ideal speed (speed limit) and v_{avg} the average speed of vehicles over a time interval (eg. 30-60 seconds). This calculation is made for defined segments and interpreted as follows:

- $CI \approx 0$: Free-flow traffic
- $CI > 0.5$: Noticeable congestion
- $CI > 2.0$: Severe congestion.

To visualize the congestion of each road segment the system maps the speed ratio to the colors shown in table 2.

Color	Condition	Speed Criterion
green	free	$>$ 80% of v_{free}
yellow	dense	50-80% of v_{free}
orange	near capacity	30-50% of v_{free}
red	congested	$<$ 30% of v_{free}

Table 2. Speed ratio to color mapping.

References

1. Castro, P., Ishakian, V., Muthusamy, V., & Slominski, A. (2019). *The rise of serverless computing.* Communications of the ACM, 62(12), 44-54.
2. Raith, P., Nastic, S., & Dustdar, S. (2023). *Serverless Edge Computing—Where We Are and What Lies Ahead.* IEEE Internet Computing, 27(3), 50-64.
3. Frey, S., Reich, C., & Lüthje, C. (2013, September). *Key performance indicators for cloud computing SLAs.* In The fifth international conference on emerging network intelligence, EMERGING (pp. 60-64).

4. Deng, S., Zhao, H., Huang, B., Zhang, C., Chen, F., Deng, Y., ... & Zomaya, A. Y. (2024). *Cloud-native computing: A survey from the perspective of services.* Proceedings of the IEEE, 112(1), 12-46.
5. Ugwueze, V. U. (2024) *Serverless Computing: Redefining Scalability And Cost Optimization In Cloud Services.*
6. Emmanuel, F. V. (2025) *Serverless Computing for Adaptive Scalability in Disaster Recovery Systems: Designing scalable, serverless infrastructures to enhance disaster recovery protocols in critical applications.*
7. Holt-Nguyen, C. (2023) *Time to Insight: A Critical Metric in Data Analytics.* Medium
8. Oakley, Joe, et al. *Foresight plus: serverless spatio-temporal traffic forecasting.* GeoInformatica 28.4 (2024): 649-677.
9. Richardson, A. J., and M. A. P. Taylor. *Travel time variability on commuter journeys.* High Speed Ground Transportation Journal 12.1 (1978).
10. Amazon Web Services. *Amazon Kinesis Data Streams API Reference: Record.* AWS Documentation, 2025.