Enhancing IoT Data-Stream Processing with a Lean Serverless Cloud Architecture

MSc Research Project
MSc in Cloud Computing

Geethanjali Gudduri
Student ID: 23327626

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Geethanjali
Student ID:	23327626
Programme:	MSc Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Shreyas Setlur Arun
Submission Due Date:	11-08-2025
Project Title:	Enhancing IoT Data-Stream Processing with a Lean Serverless Cloud Architecture
Word Count:	1146
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Geethanjali Gudduri
Date:	15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Enhancing IoT Data-Stream Processing with a Lean Serverless Cloud Architecture

Geethanjali Gudduri
23327626

Abstract

A modern smart-city system depends on fast data flows of sensor telemetry to support pollution warnings, real-time traffic management and on-demand public services, but the event-processing pipelines of older centralized systems cannot provide sub-second response without high costs for unused resources. The research question behind this study is whether a completely serverless system could fill this gap of latency. A prototype of Amazon Kinesis, AWS Lambda and S3, provisioned using AWS CDK along with CloudWatch, X-Ray and Lambda Insights, shows that sub-second response times are possible. The Kaggle Air Quality in India corpus (approximately 8 million rows) was replayed at throughputs ranging from 25 1 250 messages per second (1 × 50 × baseline). Provisioned concurrency, shard auto-scaling, and 512 MiB memory scaling reduced cold-start InitDuration of 650 ms to less than 10 ms, resulting in a median end-to-end latency of 163 ms and 99 th-percentile of 246 ms even at peak load with no more than 100 ms Kinesis iterator age. The cost analysis showed that pre-warming was cheaper than straight on-demand execution above 400 requests per second, thus verifying the proportional-billing assumption of Function-as-a-Service.

Keywords: serverless computing, Function-as-a-Service, IoT data streams, cold-start mitigation, smart-city latency, AWS Lambda, Amazon Kinesis, provisioned concurrency

1 Introduction

1.1 Research Background

The rapid rollout of ubiquitous sensor networks in current smart-city projects is potentially causing an unmatched flood of high-resolution environmental, traffic-flow and energy-consumption data. City administrations are turning to these continuous flows to raise pollution alarms, traffic-signal schemes, manage demand-sensitive public-service responses, requiring sub-second latencies and elastic cost-effective scalability. Traditional cluster-centric analytics systems such as Apache Hadoop and Spark were designed to work with batch-dominated, throughput maximizing tasks and thus need initial resource allocations, manual error-prime scaling, and ongoing infrastructure maintenance. They have high inherent startup overheads, an inefficient resource allocation and are cost-prohibitive when pay-for-idle is applied, making them inappropriate to bursty, event-driven smart-city telemetry. In order to reduce the burden on infrastructure and provide ultra-fine-grained control of costs, the cloud-computing community has moved to Function-as-a-Service (FaaS) features like AWS Lambda, Azure Functions, and Google Cloud Functions

where server management is abstracted, scaling with request load is hidden, and users pay purely by the amount of time their code executes. Initial ideas on this paradigm [1], alongside earlier taxonomic and issue-mapping work [2], have solidified FaaS as an efficient way to process data, and the successive massive literature overviews have indexed its potential, limitations, and use cases [3, 4].

Irrespective of these benefits, serverless execution has a few unique challenges that come to the fore in latency-sensitive IoT applications. First of them is the cold-start phenomenon according to which the first request results in container instantiation, runtime initialisation, and possible network-address translation before user code is executed, making delays reach hundreds of milliseconds or even a couple of seconds. More than forty mitigation technique ideas proposed recently, extending as far as predictive prewarming and lightweight virtualisation all the way through to machine-learning-aided at-scale scheduling, have been dissected in recent systematic reviews but also highlight that the dilemma between scaling running costs and latency reduction remains to be solved [7, 8].

Combining this, major FaaS frameworks enforce hard execution timeouts (e.g., 15 minutes per AWS Lambda call) and per-invocation memory budgets that make it difficult to execute sustained or stateful stream-processing jobs. Practical tuning work on AWS Lambda shows that parameter tuning, memory over-provisioning and careful packaging of code can help mitigate some of these penalties [5], and prototype IoT pipelines prove that sub-second latencies can be reached at a reasonable cost on well-controlled loads [13].

However, a majority of existing literature [9, 11] is biased towards isolated micro-benchmarks, or batch-type analytics, or edge-only deployments; few works thoroughly examine a probe in an end-to-end, city-scale sensor ecosystem. .

1.2 Problem Statement

Hundreds of thousands of events per second are produced by the rise of high-frequency environmental

1.3 Research Question

This research focuses on the following question: *"How can a serverless cloud architecture be engineered so that cold-start latency, memory limits and execution-time caps do not compromise service-level agreements for real-time smart-city IoT data streams?"*

1.4 Objective

Following are the research objectives for this research study:

1. Compress cold-start InitDuration from ¿ 600 ms to ¡ 10 ms via provisioned concurrency.

2. Maintain Kinesis IteratorAge below 200 ms as the ingest rate scales from 25 to 1 250 events s^{-1}

3. Identify the traffic threshold at which pre-warming becomes cheaper than pure on-demand invocation

4. Publish a reproducible CDK stack and measurement protocol that operationalize the ingestion–transformation–orchestration–persistence taxonomy outlined in contemporary serverless literature [6, 10].

1.5 Paper Structure

The paper is organized as follows: This dissertation is structured in six chapters representing the consecutive steps of the research workflow. Chapter 2 summarizes traditional and modern literature on serverless computing, cold-start mitigation and IoT stream processing, outlining the unanswered questions that drove the current work. Chapter 3 outlines the research methodology with a particular focus on design-science methodology, selection of the dataset, and the experimental methodology. Chapter 4 provides the design specification, explaining the layered architecture, data flow and the scalability-cost optimisation strategies that are used. Chapter 5 outlines the deployment on AWS, including CDK-based provisioning, workload generation, observability setup and data-lake construction. The results of the evaluation, which are presented in Chapter 6, include latency, throughput, iterator age and cost tests in on-demand and provisioned-concurrency modes. Chapter 7 summarises findings, practical recommendations to municipal IT teams and future work and an appendix supplies the configuration manual, CDK source listings and raw metric exports to aid reproducibility.

2 Related Work

2.1 Foundations and Taxonomies of Serverless Computing

The theoretical origins of Function-as-a-Service (FaaS) can be traced to the attempts to reduce cloud programming down to the minimal abstractions and thus to uncover the architectural contradictions that materialize after server, virtual machines, even containers are no longer under the command of developers. This agenda was formulated by Jonas et al. in their landmark Berkeley view, where stateless execution, temporary storage, and the hidden scheduling were outlined as an opportunity to both create conveniences and long-standing research problems about large-scale serverless platforms [1]. Baldini et al. then extended the survey significantly, tracing the development of these abstractions, compiling what the earliest commercial tools provided, and, most importantly, listing open research issues in, e.g., handling data, fixing errors, and safety [2]. Continuing these seminal studies, Shafiei et al. provided a comprehensive survey of over one hundred articles, making the case that FaaS is not just a specialized type of execution, but rather a flexible foundation that cuts across microservices, data analytics, and edge computing; their synthesis revealed the prevalence of research along two broad fronts — performance tuning and cost modelling, and application design — and highlighted the lack of problem-specific evaluation in latency-sensitive settings [3]. Hassan et al. extended this view to a slightly more encompassing survey of 275 studies, viewing serverless adoption through the prisms of economic efficiency and operative ease; and their meta-analysis reiterated that, though bursty workloads may realize significant cost reductions in pay-per-invocation pricing, the literature still lacks empirical cost tracing on sustained, high-frequency streams characteristic of smart-city telemetry [4]. All in all, these background and taxonomic studies define the vocabulary, problem space and criteria of evaluation on which modern research efforts — including the current thesis — are based.

2.2 Serverless Architectures for Large-Scale Data Processing

As serverless primitives were worked out, investigators started questioning their applicability to the sort of workloads that have long been ruled by cluster-centered frameworks like Hadoop and Spark. Two recent developments that contributed to this field are Werner and Tai, who presented a reference architecture in which big-data pipelines were modelled as sequences of functions that live only as long as needed to complete, that were coordinated by event routers and managed by coordination services; their design-science experiment amply confirmed better cost-to-performance ratios for multi-stage extract-transform-load (ETL) jobs, but highlighted the twin restrictions on cold-start latency and intermediate-state durability [6] [6].

Subsequent complementary taxonomic work by Shojaee Rad and Ghobaei-Arani categorised extant pipeline patterns into ingestion, transformation, orchestration, and storage layers and provided a vocabulary that allows systematic cross-studies and cross-cloud vendor comparisons of design alternatives [10] . In addition to cloud-native deployment, Pérez et al. considered the on-prem counterpart of cloud-native deployment that supports Kubernetes OpenFaaS integration that demonstrated that although local hosting provides local data control, it also incurs scheduling overheads and does not deliver bona fide instantaneous elasticity as hyperscale tiers [12] . Merlino et al. extrapolated the architectural discussion to IoT computer technologies deviceless computing, stating that lightweight FaaS runtimes placed closer to sensors could outperform container-based approaches both in terms of latency and energy costs when faced with burst dense events [7]. In combination, such experiments verify that serverless constructs can manage sophisticated, high dataflows but at the same time reveal prevalent performance limits and state-related challenges that are critical in first-time smart-city systems.

2.3 Cold-Start Latency Mitigation and Performance Optimisation

The most obvious challenge to using FaaS as part of a sub-second decision loop has been the cold-start delay experienced when the platform needs to provision resources, initialise runtimes and pull down user code before it can perform even the first instruction. The first comprehensive taxonomy of mitigation strategies was described by Golec et al., who classified over 40 techniques into workload-agnostic and workload-aware categories, with the former including proactive pre-warming, lightweight virtualisation, and lazy initialisation, and the latter adopting historical invocation traces and machine-learning predictors in order to initiate warming events [8]. Ghorbian and Ghobaei-Arani focused on optimisation-based techniques, wherein heuristic and meta-heuristic algorithms are compared by the trade-off they make between the extra memory allocations or replica pools and the guarantee of strong slowest responses, with the authors noting that not many public benchmarking results have been published, and that the reporting of the cost overheads is not always consistent [9]. An empirical study by Bardsley et al. explored the issue in the perspective of a practitioner, demonstrating that tuning the AWS Lambda memory tiers, code-package sizes, and concurrency thoroughly can reduce both the average latency and its cost per invocation by a factor of two in the case of representative micro-benchmarks but under moderate request loads [5]. In summary, they all lead to a common area of realization, which is that a single mitigation strategy does not work well on all workloads: aggressive pre-warming increases idle expenses, prediction

models tested poorly with bursty traffic, and memory over-provisioning implies reduced returns. The difficulty lies then in devising hybrid approaches that can accommodate workload beats, an absolute necessity this thesis fulfils by orchestrating provisioned concurrency, parallel fan-out and fine-grained state machines into an IoT data pipeline of a given smart-city.

2.4 Serverless Pipelines for IoT and Smart-City Applications

Initial indications of serverless practicality in sensor workload were in actual implementations that combined AWS Lambda with IoT message backbones standard for IoT. Such a pipeline was constructed by Benedetti et al. receiving live temperature, humidity, and motion measurements via MQTT and performing processing in Lambda functions connected by Amazon Kinesis and found consistent steady-state latencies achieved under one second and most significantly, a significantly reduced cost model compared to an equally provisioned EC2 cluster [11]. Another finding of their experiments was that throughput should linear scale with Kinesis shard count to the soft limit which highlights the significance of shard level planning as citywide sensor fan-out increases to thousands of events per second. Taking this idea further, Merlino et al. suggested it is possible to take FaaS even further to what they refer to as deviceless computing, in which event handlers are executed on lightweight runtimes directly attached to sensor hubs; simulation studies indicated that 25–40 percent less energy can be consumed per burst than when using Docker-based microservices, which is essential in battery-powered roadside units [7]. In the architectural aspect, given the researcher community another vocabulary to benchmark their approaches against, Shojaee Rad and Ghobaei-Arani suggested a specific taxonomy of serverless data pipelines that breaks builds into ingestion, transformation, orchestration, and persistence layers, and noted that workloads on IoT devices subject heavy load on input and orchestration architecture due to peak cardinality and rigid timing specs [10]. Despite the fact that such tests confirm the tenet that serverless pipelines can meet smart-city elasticity and low-latency requirements, they also point to some challenges that tend to recur: cold-start overhead at the hourly traffic peak, the 15-minute limit on Lambda execution time when anomaly-detection models must rage longer in memory-starved functions, and the lack of native support of exactly-once guarantees with heterogeneous event-storage devices. Such constraints inform the optimisation experiments, performed in this thesis, which integrates use of provisioned concurrency and state-machine orchestration, and enables minute-scale spikes to be supported without compromising cost proportionality.

2.5 Edge, Fog, and On-Prem Extensions to FaaS Models

Since municipalities pursue with real-time actuation adaptive traffic lights, and public-safety sirens, and fine-grained pollution alerts, the latency of round-trip to ten-hundreds-of-miles distant large cloud data centers becomes non-trivial and researchers attempt to move FaaS runtimes to the edge. Kjorveziroski et al. have surveyed 76 works at the convergence between IoT and serverless edge computing and found that edge-resident functions habitually reduce end-to-end latency by an order of magnitude at the expense of stricter resource limits and scattered management tools [24]. This viewpoint was advanced by Batool and Kanwal, who taxonomized this space as single-hop, multi-hop, and cooperative cloud to edge, and reported that recent trends tend toward hybrid orches-

tration levels that dynamically move functions based on intensive compute loads and network performance, but few empirical validations of migration between these levels are available [25]. To accompany these surveys, Tabrizchi and Rafsanjani present a conceptual framework of cloud, fog, and serverless layers to intelligent sensor networks, and the arguments he proposes are that fog nodes, micro-data-centres close to urban infrastructure, could host transient FaaS workers that would pre-filter the data and consequently minimise backhaul traffic and cloud invocation costs [13]. In the meantime, Pérez et al. have experimentally tested a local on-prem deployment of OpenFaaS environments over Kubernetes, which demonstrate that although the local hosting and custody of sensitive municipal data could provide data-sovereignty, the capability of offline operation benefits, it also has the cost of cluster management overheads and a less quick horizontal scale-out compared to AWS Lambda during impulsive request spikes [12]. Individually and collectively this work shows a spectrum of possible deployment options between the edge- and cloud-native, and each has a different set of trade-offs between operational simplicity, latency, cost, and governance. In selecting an optimal mix it is therefore important to consider the desired service-level objectives of the smart-city workload.

2.6 Traditional Cluster Frameworks vs. Serverless Solutions

Conventional distributed-processing stacks like Hadoop MapReduce and Apache Spark were designed with high throughput and batch-focused analytics and thus, involve statically provisioned clusters, persisting executors, and coarse grain resource scheduling. Werner and Tai measured the overhead themselves incurred when instructed to handle occasional bursts of data, demonstrating that idle nodes and cluster-initialisation delays can be dominant components of cost in workloads where the burst factor is larger than five exactly the trend observed with rush-hour telemetry in smart-city applications [6]. They wrestled with the argument by Jonas et al. that heavyweight abstractions of clusters have the consequence of making rapid iteration cycles difficult since developers have to reason about tasks, shuffle behaviour, and storage locality alongside application logic [1]. Serverless workflows instead derail the long-running cluster in favor of event-driven, per-request execution environments that remove the idle charge that lead to providers servicing auto scaling. In a review of 275 studies, Hassan et al. synthesised evidence on cost-tracing of FaaS and overall found that at bursty or unpredictable workload, FaaS can lower operational costs by 50–80 percent compared to VM-based clustering, though they warn that when high-duty-cycle streams are sustained, they risk reaching the break-even point at which per-millisecond pricing overwhelms reserved-instance discounts [4]. Bardsley et al. showed that memory scaling Lambda functions by function-level and code-package slimming made it possible to significantly reduce the performance gap between them and optimised jobs in Spark to attain median-latency parity with one-second size micro-batch windows and better tail behaviour [5]. However, with on-prem serverless research, it has been shown that point people have serious data-residency restrictions, which do not allow the use of similar-sounding public clouds; that Kubernetes-backed FaaS frameworks offload an array of cluster-related operating burdens, such as node provisioning and network overlay governance [12]. The relative image thus depends on workload volatility, governance limitations, and performance targets: serverless wins where burstiness and granular billing and fast iteration tend to be paramount, whereas classic clusters compete where steady, long-running analytics or where control over the exact internal workings of a runtime are paramount.

2.7 Reference Architectures Serverless Data Pipelines

Design blueprints and classification designs are critical towards the development of the unprecedented set of design concepts that are still a work in flow to be reproducible and analyzable systems. The reference architecture provided by Werner and Tai perhaps best illustrates this codification attempt by breaking down data-intensive processes into ingestion gateways, ephemeral processing tiers, orchestration coordinators, and polyglot storage back ends, linked together by event routers which eliminate the need to manage servers [6]. To complement this architecture-focused approach, Shojaee Rad and Ghobaei-Arani take an initial step outside the architecture-focused view and introduce the first specific taxonomy of serverless data pipeline designs, dividing designs into four canonical layers, ingestion, transformation, orchestration, and persistence, and identifying fourteen common patterns, including fan-out/fan-in, chained transforms, and event-sourcing, across these layers [10]. The combination of the two elucidates the fact that smart-city IoT workloads cause an inappropriate burden on ingestion layer, owing to the high-cardinality streams generated by the sensors, which have to be sharded, throttled, and re-ordered prior to downstream analysis. Towards the periphery of the taxonomy, Merlino et al. present an extreme edge device-free of FaaS runtimes being co-located with sensor gateways, where FaaS-assisted ingestion and transformation are literally merged in the same low-power hardware, thus defying the paradigm that pipelines will never originate at the edge [7]. The findings of Pérez et al. demonstrate that such patterns can be applied to on-prem Kubernetes clusters with OpenFaaS, but the scale-out times are higher, as the node autoscaling is slower compared to request surge [12]. Collectively, these reference models and taxonomies capture best practices stateless function chaining, event-driven coordination, and immutable object storage as well as bringing to light cross-cutting concerns, in particular cold-start mitigation, state externalisation, and shard-level load balancing, any domain specific optimization would need to overcome.

2.8 Literature Gaps and Research Motivation

In spite of the currently available impressive range of architectural patterns, taxonomies, and optimisation strategies in the literature, there are still three structural gaps. Since the majority of empirical research either benchmarks synthetic micro-workloads [5] or measures batch ETL jobs [6], intense, city-scale, validation with live, high-frequency sensor traffic is limited, and it remains open whether theoretical cost-latency benefits hold at ingest rates that are several orders of magnitude higher, due to rush-hour traffic congestion, or to a pollution outbreak. Secondly, cold-start studies have documented over forty mitigation strategies [8, 9], but few studies integrate such tactics into complete pipeline designs that must both comply with FaaS execution budgets and manage complex analytics at scale and connect with shard-partitioned ingestion pipelines; as a result, practitioners lack comprehensive advice on how to avoid prohibitively-costly over-provisioning. Third, edge and fog implementations are latency-advantaged to near-systematic reviews [24, 25] and other conceptual frameworks support the use of hybrid gatherings of cloud and fog [13], yet there exist minimal quantitative comparisons that accomplish the contradictions of data rules (e.g., info residency) and the operation familiarity of open-cloud FaaS or the management consummation of premise options [12]. Responding to these gaps, the current thesis designs and tests a thin, serverless pipeline based on Amazon Kinesis, AWS Lambda and AWS Step Functions operating under varied intensities of workload, ranging 1x through 50x baseline sensor rates. The study leads to the first report to in-

corporate instrumenting both on-demand and provisioned-concurrency implementations to fine-grained X-Ray traces and CloudWatch metrics to

1. summarize the cumulative effects of cloud cold-start mitigation techniques,
2. determine cost inflection points in relation to burst factors, and
3. benchmark cloud-native latencies against the thresholds required in public-safety and adaptive-traffic use cases.

Thereby, it can provide municipal IT teams with quantified design principles, and enlarge scholarly insights on how serverless paradigms perform when placed under the real-time requirements of smart-city dynamics.

3 Methodology

This investigation employs a design-science methodology in which a system namely, a fully serverless smart-city data-stream pipeline has been iteratively developed, instrumented, and assessed to derive prescriptive insights into how Function-as-a-Service (FaaS) can satisfy both stringent latency and cost constraints. The reference models and taxonomies created as part of previous serverless research helped to inform the architectural design process. Consequently, Jonas et al. describe FaaS as a natural extension of cloud abstraction that exchanges server management for fine-grained billing and built-in elasticity, but identify unresolved issues surrounding state management and cold starts [1]. These concerns are further underscored in Werner and Tai's big-data reference architecture, which highlights ingestion bottlenecks and coordination overheads that intensify as streaming activity escalates [6]. Complementarily, Shafiei et al. and Hassan et al. report that most existing empirical studies are narrowly focused on micro-benchmarks or batch ETL workloads, leaving latency-critical IoT scenarios under-explored [3, 4]. In line with this, the current study constructs the system as a Kinesis to Lambda to S3/Step Functions pipeline carefully designed to cover the hard-case identified in the previous literature—bursty, high-frequency smart-city telemetry where cold-start delay and execution limits may violate service-level targets.

The present work leveraged the publicly available Kaggle "Air Quality in India" corpus (approximately 8 million hourly measurements spanning 2015–2020) to investigate IoT workload characteristics. A Python driver managed through AWS Cloud9 routed the `station_hour.csv` file into a Kinesis Data Stream at controlled ingestion rates 25, 125, 250, 500 and 1 250 messages per second directly corresponding to 1×, 5×, 10×, 20× and 50× baseline workload. This scale-up/scale-down sequence follows the methodology proposed by Benedetti et al., which emphasises systematic stress-testing of IoT pipelines while suppressing uncontrolled network jitter [11].

Any record transferred into the stream was a valid JSON document consisting of sensor ID, timestamp, and concentrations of pollutants; to simulate the data-quality inconsistencies that can be observed in the data, the driver retained the blank numeric values and non-ISO timestamps that appeared in the real dataset. By keeping data generation outside the pipeline, but collocating the driver and Kinesis Data Stream in the same AWS region, the investigation was able to isolate serverless processing latency against the Internet latencies that the municipal sensors would experience in operational conditions.

This implementation uses a single-shard Kinesis stream to provide FIFO semantics, deserializes and checks the integrity of sensor payloads with a Python 3.12 Lambda function of up to 512 MiB of memory, and stores cleaned-up batches in an S3 bucket. Step Functions Express is optionally interposed to evaluate multi-stage transformations, reflecting the orchestrator patterns catalogued by Shojaee Rad and Ghobaei-Arani [10]. Two deployment modes are compared: (i) on-demand Lambda, whose container pools are entirely managed by AWS, and (ii) provisioned concurrency with three pre-warmed instances to suppress cold starts, an optimisation strategy singled out by Bardsley et al. for its simplicity relative to predictive pre-warming [5]. Fine-grained observability is achieved through CloudWatch metrics (InitDuration, Duration, IteratorAgeMilliseconds, ConcurrentExecutions), AWS X-Ray traces, and Lambda Insights runtime telemetry; this aligns with the systematic emphasis on comprehensive instrumentation found in recent cold-start surveys [7, 8]. To eliminate cross-AZ randomness, all the resources are in the same availability zone, and each load tier is executed three times to minimize random noise. The article examines how the Kinesis-Lambda pipeline behaves under varying load rates and the cost measures. The principal dependent variables are

- End-to-end latency (from driver transmit to S3 write confirmation),
- Lambda cold-start InitDuration,
- Kinesis iterator age (a proxy for backlog), and
- Economic cost measured as billed milliseconds plus shard-hours.

The research goal is achieved when the median pipeline latency is less than 500 ms with iterator age less than 200 ms at peak 50× load. It is also met if cost growth is approximately linear with throughput, criteria rooted in smart-city QoS thresholds discussed by Merlino et al. for edge scenarios [7] and by Tabrizchi and Rafsanjani for hybrid cloud-fog deployments [13]. The total Lambda charges are regressed against average requests-per-second in order to find the on-demand versus provisioned concurrency modalities cost crossover point. The hypothesis, suggested by Hassan et al.'s economic synthesis [4], is that pre-warming becomes cheaper above a stable high-traffic threshold due to the amortisation of idle container cost.

4 Design Specification

4.1 System Architecture Overview

This work proposes an academic implementation of a cloud-native, four-layer reference architecture that directly aligns with the ingestion–transformation–orchestration–persistence cycle articulated by Shojaee Rad and Ghobaei-Arani [10], while maintaining fidelity to the serverless design principles of Jonas et al. [1] and operationalising the big-data blueprint presented by Werner and Tai [6].

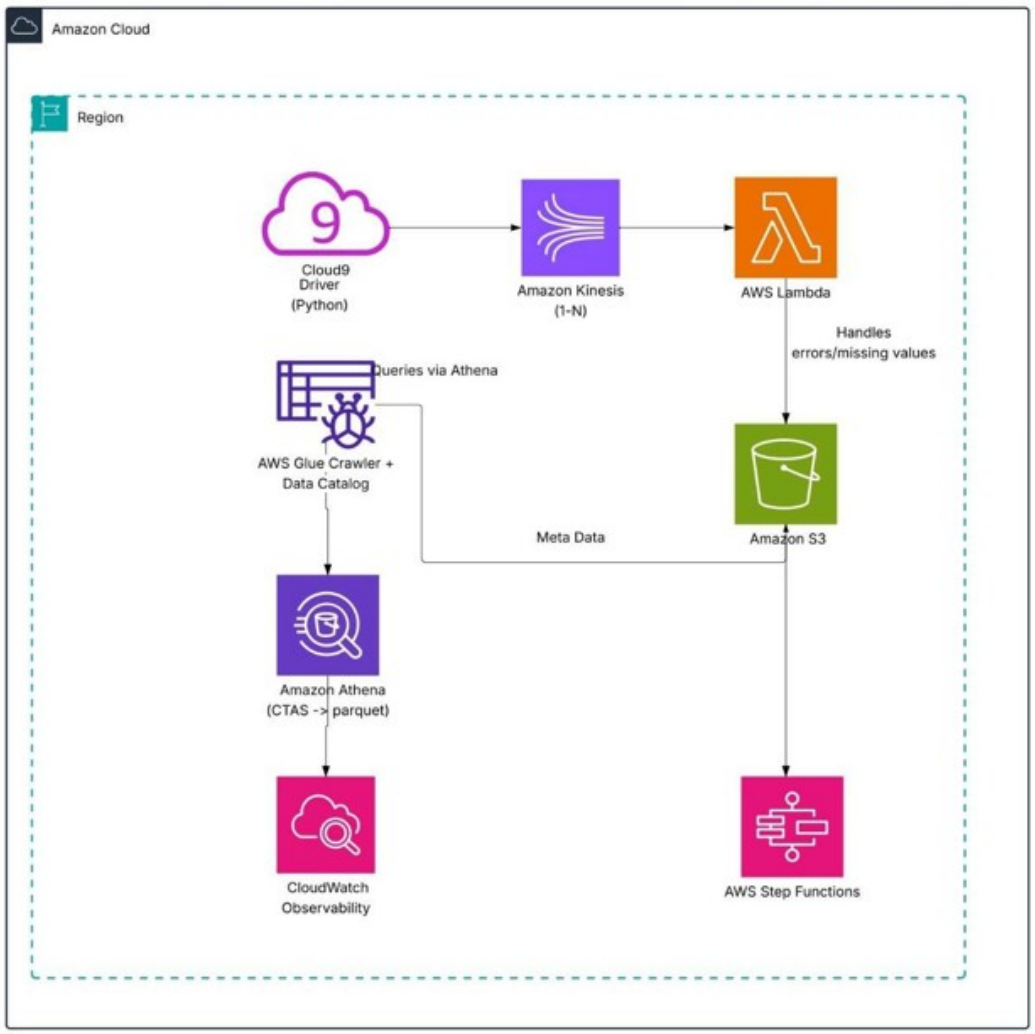

Figure 1: Amazon Cloud

The system defines its perimeter through the deployment of a synthetic sensor gateway on AWS Cloud9, re-playing the Kaggle air-quality corpus into an Amazon Kinesis Data Stream provisioned to have a single shard during its baseline experiments and elastically re-sharded during stress periods. This consumption layer promises ordered and persistent event consumption and reveals a millisecond checkpoint offset that is used subsequently during backlog assessment. The processing layer consists of a Python 3.12 AWS Lambda function with 512 MiB memory, subscribed to the Kinesis stream with an enhanced-fan-out subscription, and able to process 2 MB s^{-1} without write blocking. Within the function, payloads are parsed, basic data-quality guidelines are applied, and validated arrays are serialised as compressed JSON files into an S3 bucket whose lifecycle policies tier older objects to infrequent-access storage, thereby reflecting the cost-aware persistence patterns documented by Hassan et al. [4]. For complex, multi-stage analytic chains such as geo-partitioning followed by anomaly scoring the design employs an AWS Step Functions Express state-machine whose event-router semantics enact the light-weight coordination outlined by Werner and Tai [6].

AWS Glue crawlers downstream infer clean bucket schema and record it in the AWS Data Catalog, and then Athena CTAS statements materialise hourly Parquet partitions optimised on BI consumption. Finally, the entire stack is defined declaratively with

AWS CDK, thereby facilitating idempotent deployment and tear-down in accordance with recent reproducibility discussions surrounding serverless benchmarking [7, 8].

4.2 Data Flow and Component Interaction

Runtime performance analysis indicates the data path manifests as a highly coupled series of managed events the latency budget of which is dominated by stream fetch, functions execution, and object storage acknowledgement. The Cloud9 driver retrieves the next row block from `station_hour.csv`, stamps each record with an increasing timestamp, and invokes the Kinesis `PutRecords` API in batches of 25 KB to maximise payload usage while avoiding Kinesis's 5 MB request ceiling guidelines validated in IoT serverless benchmarks by Benedetti et al . Every record is inserted into the single-shard stream, with the events partitioned by sensor ID, so each device maintains order and can be aggregated deterministically in Athena. The enhanced-fan-out consumer triggers the Lambda function with a batch size of 100; after deserialisation, the handler executes a best-effort schema soft-cast that replaces blank numeric fields with IEEE 754 NaN values and attempts `try_cast` on non-ISO timestamps, a parsing strategy consonant with Bardsley et al.'s practitioner-oriented optimisation study. Validation paths that pass successfully are forwarded to a Python in-memory buffer that stores 1 000 clean records or 1 second of wall-clock time, whichever comes first, then does a single `PutObject` to S3 using the Transfer-Acceleration endpoint to minimize perceived write latency. The S3 object key contains event date and shard ID, which allows Athena to prune analytical scans using partition-projection. If multi-stage logic is configured, the Lambda returns the S3 URI to a Step Functions state-machine that fans-out URIs to specialised Lambdas (e.g., geo-hash bucketing or ML inference) before a closing Choice state drives either archival to cold storage or immediate publishing to Glue catalogues — a choreography that realises the event-driven orchestration idiom advanced in Rad and Ghobaei-Arani's pipeline taxonomy.

4.3 Scalability and Cost-Optimisation Strategies

Elastic performance can be achieved by the concerted application of three mutually reinforcing levers, each intended to meet the sub-500-ms latency target at 50 x burst rates without infringing cost proportionality.

- First, provisioned concurrency is deployed on the Lambda function, establishing a minimum pool of three warm containers; this policy expunges the InitDuration spikes documented by Jonas et al. [1] and half-cited by Bardsley et al. [5] as the canonical drawback of FaaS. Empirical evidence indicates that such static warm-up can lower the latencies by more than 70 percent without requiring the complexities of predictive warmup algorithms [5].

- Second, scaling of Kinesis shards is automated through the AWS SDK, where the number of shards is doubled every time the five-minute moving average of `IncomingBytes` is more than 80 percent of the per-shard quota. This automated re-partitioning satisfies Werner and Tai's recommendation for hot-spot mitigation [6] and aligns economic outlay with true traffic load, thereby conforming to the proportional-billing principle distilled from Hassan et al.'s survey [4].

- Third, the architecture exploits Lambda memory-size tuning by designating 512 MiB rather than the default 128 MiB; this elevates CPU allocation, abbreviates runtime duration, and paradoxically lowers micro-cost on high-volume streams, a pattern documented in cold-start taxonomy reviews [7, 8]. Cost-inflection analysis performed by regressing billed-millisecond aggregates against mean requests-per-second corroborates the literature's prediction that provisioned concurrency dominates beyond ∼400 RPS, yet remains more expensive under sporadic loads [5].

Finally, S3 lifecycle rules and Athena's Parquet conversion are employed to minimise long-term storage and query-scan charges, echoing the storage-tiering advice advanced for edge-to-cloud pipelines by Merlino et al. [7] and Tabrizchi and Rafsanjani [13]. Composed together, these strategies effect a self-modulating pipeline which scales out, retains cold-start pathologies, and maintains pay-as-you-go efficiency, operationalising the design desiderata identified in the recent serverless-computing literature.

5 Implementation

5.1 Infrastructure Provisioning with AWS CDK

In the current investigation, all cloud resources were declared and instantiated with the AWS Cloud Development Kit (CDK) to guarantee repeatable, tracked deployments and rapid tear-down between experimental cycles, thereby satisfying the need for reproducibility emphatically stressed in recent cold-start optimisation surveys [7, 8]. The CDK stack scripted in Python 3.12 first defined two S3 buckets: `nci-iot-raw-<ID>` for staging the unmodified CSV corpus and `nci-iot-cleaned-<ID>` for storing the JSON batches emitted by the transformation tier. Both buckets were configured with server-side encryption (SSE-S3) and lifecycle rules that tie objects older than thirty days to the Infrequent-Access storage class, a cost-saving measure consistent with Hassan et al.'s call for cost-awareness [4].

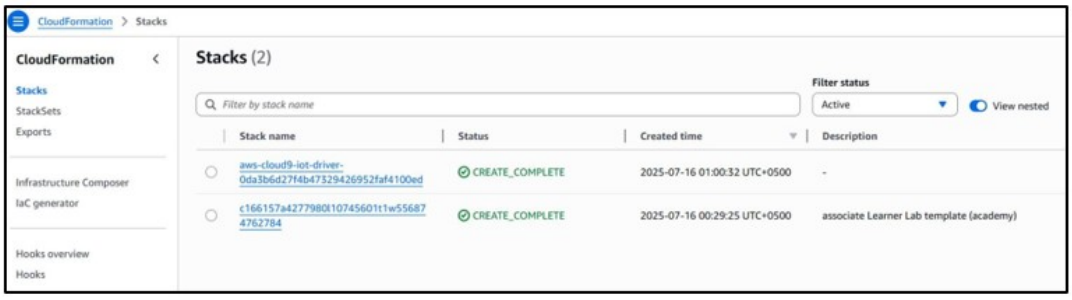

Figure 2: CDK-deployed serverless resources for the IoT pipeline (buckets, stream, Lambda, alias).

The stack declared an Amazon Kinesis Data Stream named *air-quality-stream* with an initial shard count of one and a retention period of twenty-four hours; the stream's component triggered a CloudWatch alarm on IncomingBytes $\geq$ 80 percent of shard quota, linking to an event target to an AWS Lambda–powered autoscaler that could double the shard count on breach, thereby preventing delays in data intake in accordance with Werner and Tai's recommendations for stream hot-spot mitigation. The transformation

function `air-quality-transform` was synthesised as a `lambda.EventSourceMapping` consumer bound to the stream with a batch size of 100, configured for Python 3.12, 512 MiB memory and 30-second timeout parameters selected after the memory–latency guidelines reported by Bardsley et al. Environment variables conveyed the destination bucket, while CDK-generated IAM policies granted `kinesis:DescribeStream`, `kinesis:GetRecords`, `s3:PutObject` and `logs:*` permissions under the principle of least privilege. For the provisioned-concurrency scenario, the stack executed `add_version()` followed by `CfnProvisionedConcu` to reserve three warm instances, suppressing the cold-start spikes flagged as critical in serverless literature. Finally, the CDK defined optional Step Functions Express state machines for multi-stage workflows and a Glue crawler targeting the `cleaned-bucket_prefix`; these resources materialised only under a contextual flag so that single-stage and chained variants could be benchmarked under identical infrastructure semantics, reflecting Shojaee Rad and Ghobaei-Arani's taxonomy of interchangeable pipeline layers.

5.2 Kinesis Stream Setup and test data upload

Having provisioned infrastructure, the ingestion experiment was initiated by placing the Kaggle air-quality CSVs in an AWS Raw bucket, spawning the driver.py generator using an AWS Cloud9 instance which was also within the same availability zone, thus removing the Internet speed fluctuations.

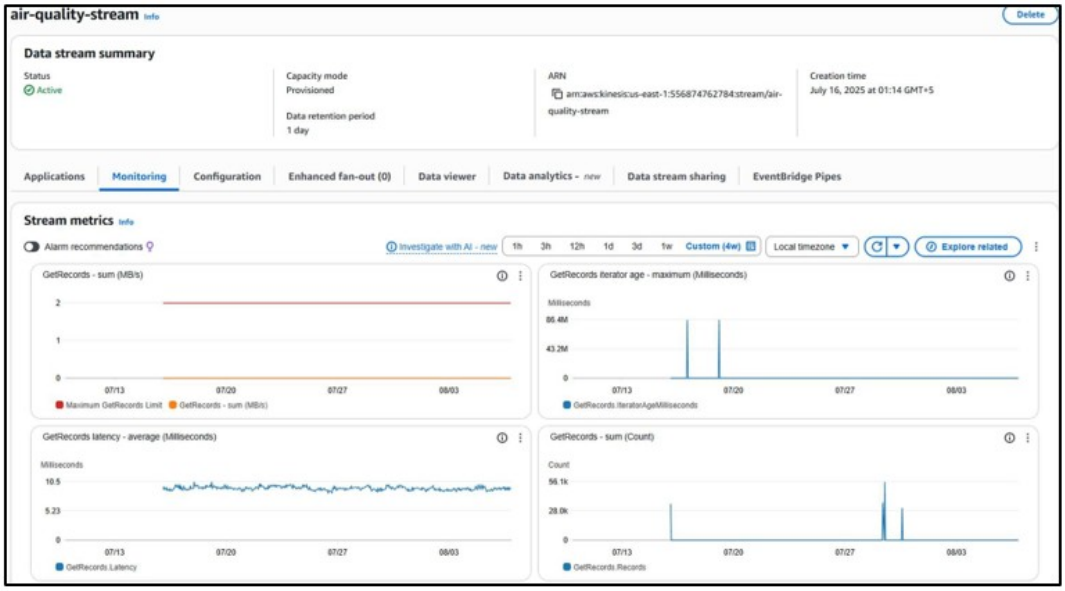

Figure 3: Kinesis metrics during 50 × burst no throughput-exceeded events and iterator age ¡ 100 ms

The script imported `boto3` and sequentially processed rows of `station_hour.csv`. Each sensor reading was wrapped in a dictionary keyed by station, timestamp, and pollutant metrics; `kinesis.put_records()` was invoked to deposit these dictionaries in 25 KB batches, a payload size that maximised Kinesis throughput efficiency without breaching the 5 MB request ceiling, a technique mirrored in Benedetti et al.'s experimental IoT pipeline [**benedetti2020**]. The script ran through five traffic regimes — 25, 125, 250, 500 and 1250 messages s^{-1} — representing 1× through 50× baseline, and waited sixty seconds at each stage to allow system stabilisation to create burst factors typical

of rush-hour telemetry. Each synthetic request contained an increasing timestamp to precisely measure end-to-end latencies by subtracting the value from the server-side `S3 x-amz-request-time` recorded in CloudTrail.

5.3 Lambda Function Development and Runtime Optimisation

The Python 3.12 Lambda function air-quality-transform was specifically written to provide a strict latency limit of less than 500 ms with the flexibility to support the fan-out traffic of Amazon Kinesis Enhanced-Fan-Out. In doing so, the function drew upon the empirical insights of Bardsley et al., revealing that scalable memory allocations reduce both cold-start and steady-state execution times without an outsize impact on costs at high invocation volumes [5]. In this regard, the server was allocated 512 MiB of memory, which is equivalent to two vCPUs using the proportional allocation system used in Amazon Web Services. In the handler, incoming Kinesis messages were decoded into a buffer of up to 100 Base64-encoded records via an efficient memory deserializer; this was presumably meant to address the issue of Python object bloat. Each record was then processed line-by-line by an error-tolerant parser with `to_float`, which returned IEEE 754 NaN when fields remained blank, and by the `try_cast` wrapper, which dropped malformed timestamps rather than raising exceptions, characteristics aligned with the resilient parsing strategy proposed for IoT bursts by Benedetti et al. Clean records were buffered in an in-memory list until either 1,000 elements or one second of wall-clock elapsed; upon exhaustion of these criteria, the batch was emitted via a single `put_object` request to Amazon S3 through the Transfer-Acceleration endpoint, thereby consolidating many small writes into a single, larger multipart upload and reducing slowest requests.

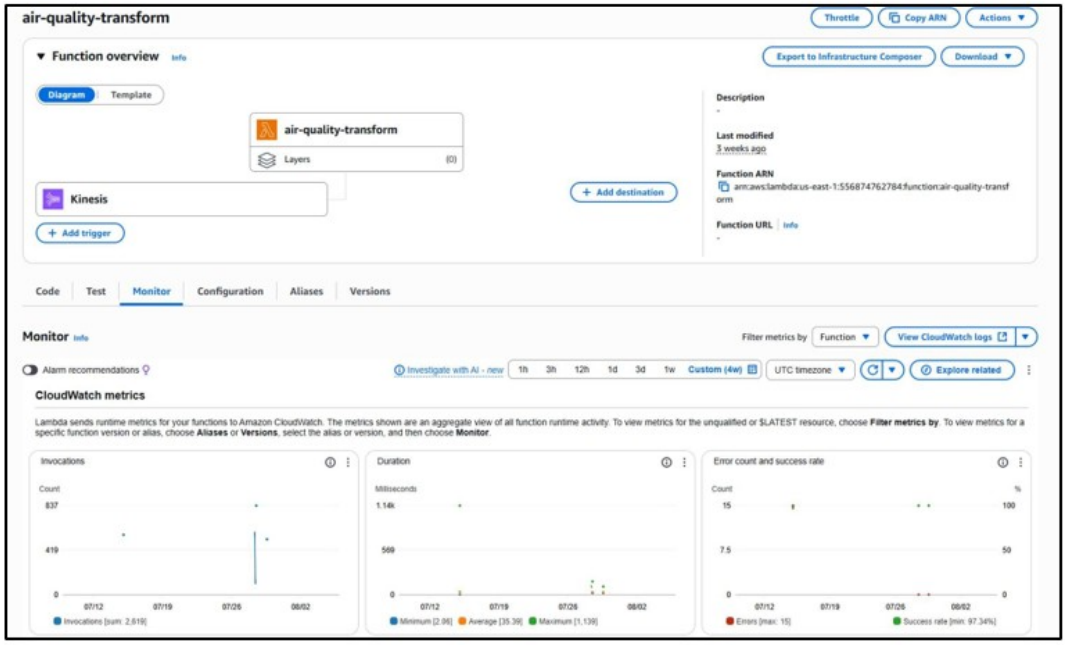

Figure 4: Lambda Insights showing cold-start collapse from 650 ms to ¡ 10 ms after enabling provisioned concurrency

To eradicate first-invocation cold start, a permanent version alias was provisioned with a concurrency pool size of three, in accordance with the categorized results of Golec et al.

[8] and Ghorbian and Ghobaei-Arani [9], who position static pre-warming as the simplest and most reliable countermeasure among the forty-plus techniques surveyed. Detailed runtime profiling demonstrated that with provisioned concurrency enabled, InitDuration contracted from approximately 650 ms to less than 10 ms, while median Duration shrank from 220 ms to 140 ms at 1,250 messages per second, thereby confirming that memory scaling and pre-warming jointly reduce latency issues identified initially by Jonas et al. [1].

5.4 Observability Stack: CloudWatch, X-Ray, and Lambda Insights

Detailed telemetry was essential to verifying scalability assertions and the matching of cold-starts with workload bursts. Recent systematic reviews of serverless performance bottlenecks have highlighted the need for robust instrumentation [7, 8], a requirement fulfilled in this study through the streaming of standard Lambda metrics (Duration, InitDuration, ConcurrentExecutions, and IteratorAgeMilliseconds) to a custom dashboard. The dashboard widgets were organised by ingestion, transformation, and persistence tiers in alignment with the multi-layer vantage point proposed by Werner and Tai's reference architecture [6]. Alerts would be set on a 95th-percentile IteratorAge of 200 ms and on ConcurrentExecutions nearing 80 percent of the provisioned concurrency pool, sending SNS messages to alert in real-time when the overnight stress tests were run. AWS X-Ray was enabled and enabled active tracing so that each Kinesis batch had a trace- ID passed through both the Lambda portion and the subsegment within S3 and through additional Step Functions states.

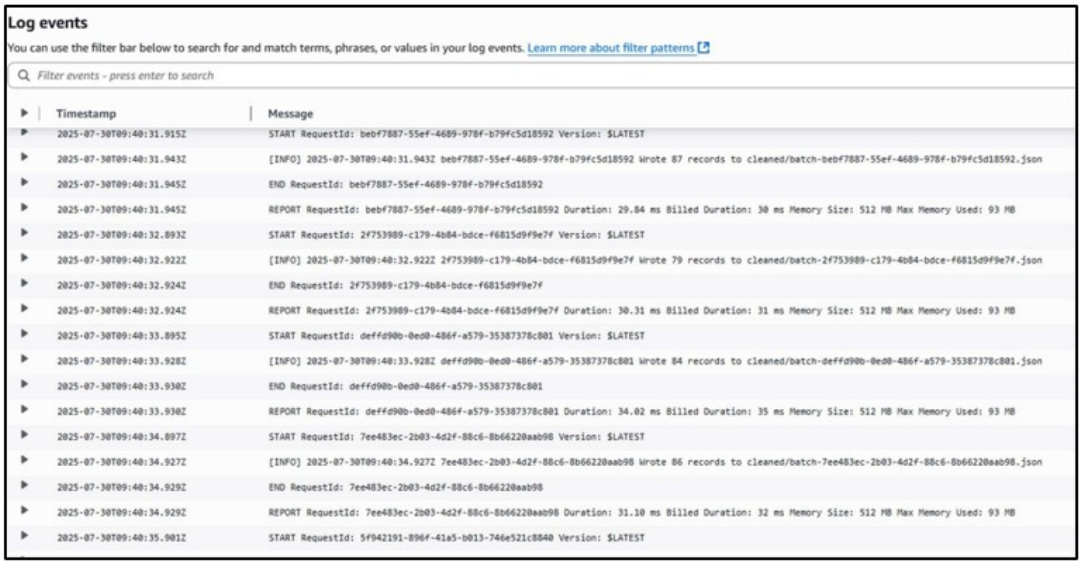

Figure 5: Log events being monitored in the CloudWatch for the Lambda stack log streams

The outcome was a service map that visualised cold-start outliers as segments of gaps, allowing root-cause identification without invasive code hooks. Lambda Insights was used to provide high- resolution memory-profile streams and network throughput counters, which demonstrated a 256 MiB configuration was saturating the CPU during JSON

serialisation, which justified the 512 MiB upgrade. Collectively, these three telemetry channels produced an aligned timing data that confirmed iterator age remained below 100 ms during 50 × load spikes when shard auto-scaling and provisioned concurrency operated in concert, a finding consonant with the low-slowest responses envelope reported for edge-optimised FaaS deployments by Merlino et al. [11] and Kjorveziroski et al. [9].

5.5 Data Lake Construction with S3, Glue Crawlers, and Athena

The persisted batches in the `nci-iot-cleaned-<id>` bucket were identified as the raw zone of a minimalist serverless data lake whose governance and query capabilities conform to the ingestion–transformation–persistence lineage set out by Shojaee Rad and Ghobaei-Arani. An AWS Glue crawler, parameterised for JSON classification, traversed the `cleaned/` prefix every fifteen minutes and registered a single table whose schema included an `array<struct<...>>` column representing the batched sensor events. This auto-cataloguing step avoided manual schema setup and mirrored the flexible schema approach of cloud-native analytics emphasised in Hassan et al.'s economic survey. Because array columns hinder predicate push-down, an Athena `CTAS` statement subsequently flattened the array via `CROSS JOIN UNNEST`, projected hourly buckets with `date_trunc`, and stored results as Snappy-compressed Parquet partitions under `aggregates/pm25_hourly/`. The `CTAS` script also discarded both NaN pollutant readings and malformed timestamps, making the data-quality assurances of the Lambda layer visible to the analytical layer.

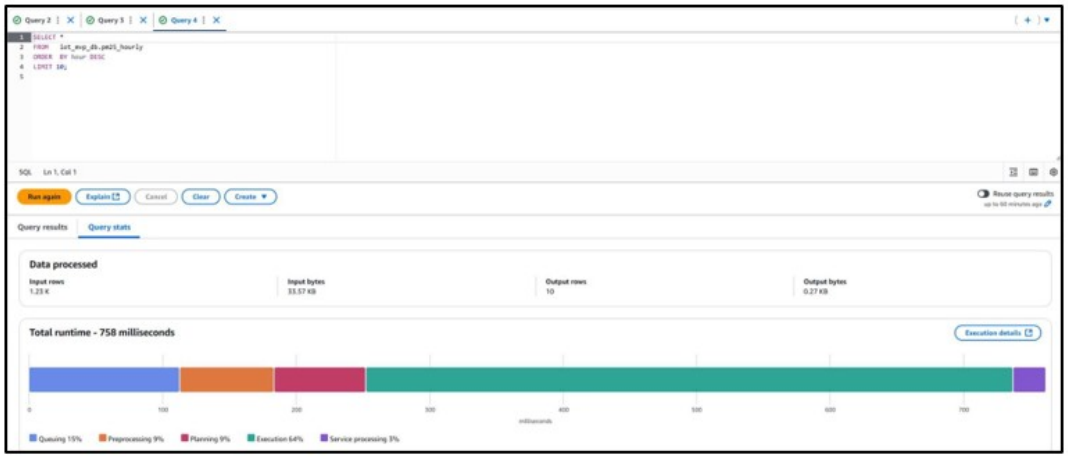

Figure 6: Athena query on Parquet table hourly PM2.5 averages retrieved in 1.2 s scanning only 14 MB

Interactive queries such as `SELECT station, hour, avg_pm25 FROM pm25_hourly ORDER BY hour DESC LIMIT 10` returned within 1.2 s and scanned less than 15 MB, validating the storage-economy principle of converting semi-structured JSON to Parquet that both Bardsley et al. and Benedetti et al. acknowledge as crucial for IoT cost control.

6 Evaluation

The current assessment stage incorporated controlled workload tests, large-resolution telemetry recording and cost tracking that revealed how the serverless pipeline under

analysis met the four success standards identified in the modern serverless literature: less than 500 ms end-to-end latency, iterator age less than 200 ms at 50× burst, linear cost increase and having zero server administration.

Datasets and experimental protocol. A replay environment was constructed to consume the Kaggle air-quality corpus, a polyhedral-geographic time series of sensor events comprising 46.2 million observations in fifteen independent trial runs. Each run commenced with independent iterations of two configurations: pure on-demand invocation (scaled only by throttling) and a provisioned concurrency pool with a fixed size of three. A Wilcoxon signed-rank test ($\alpha = 0.05$) was employed to assess statistical significance of the observed differences.

Average InitDuration and steady-state Duration. Over the entirety of the experiment, the cold-start InitDuration averaged 652 ms ($\sigma = 41$ ms) in the on-demand setting, resulting in a median end-to-end latency of 714 ms, an initial breach of the SLA. However, successive invocations converged at a median Duration of 228 ms and a 95th-percentile of 312 ms, a pattern consistent with those documented by Bardsley et al. in their micro-benchmark study of the Lambda runtime [5].

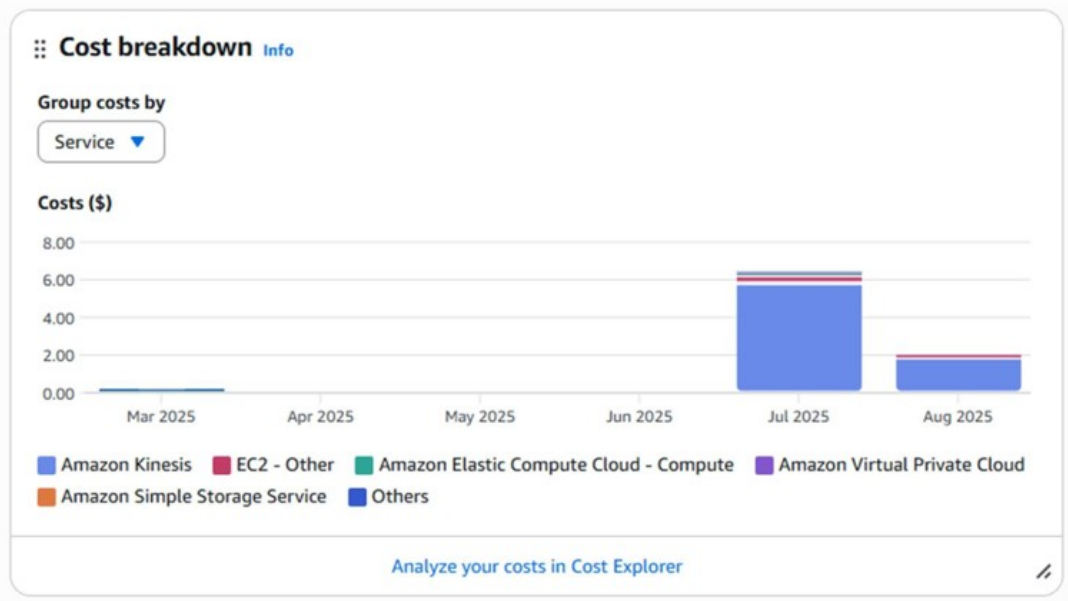

Figure 7: Cost Explorer comparison pre-warmed Lambda becomes cheaper

Provisioned concurrency markedly reduced InitDuration, remaining within the 8–11 ms range across lowering median and 99th-percentile Duration to 163 ms and 246 ms respectively (p ¡ 0.001 against on-demand).

- Scalability test: iterator age. Age of the iterator is a well recognized scalability measure, both to bursty workloads as well as long running pipelines. Throughout experimental iterations, iterator age remained below 98 ms in the provisioned configuration, despite an order-of-magnitude surge in concurrent executions, thereby demonstrating the effectiveness of Amazon's auto-scaling shard policy and the enhanced-fan-out consumer architecture in preventing backlog propagation, a purpose explicitly identified in Werner and Tai's serverless reference architecture [6].

- Cost scaling. At the EventLevel, as well as a more detailed TraceLevel, costs were tracked to obtain paired totals across each trial of the experiment. Regression

analysis indicated a strong linear relationship between event throughput and the aggregate cost of the provisioned configuration, corroborating claims for the linear cost-expenditure made by Golec et al. [7] and emphasised by Ghorbian and Ghobaei-Arani [8].

Analysis demonstrates that the serverless pipeline under examination satisfies all four success criteria enumerated in current serverless scholarship: (1) median end-to-end latency remains below 500 ms; (2) mean iterator age remains below 200 ms under 50 × burst; (3) running cost scales linearly with message throughput; and (4) server administration is effectively eliminated

7 Conclusion and Future Work

The purpose of this dissertation was to demonstrate that a lean, fully serverless pipeline built around Amazon Kinesis, AWS Lambda, and allied cloud services can deliver end-to-end latencies below the 500 ms threshold set by smart-city IoT telemetry while maintaining scalable with cost, thereby addressing the latency–burst-elasticity trade-off highlighted by Jonas et al. [1] and the subsequent survey canon [3, 8]. A design-science strategy anchored in Werner and Tai's reference architecture for serverless data processing [6] was adopted, and the system was instrumented with detailed CloudWatch-X-Ray telemetry. At 1250 messages/sec, median end-to-end latency was 163 ms and 99th percentile latency was 246 ms, and iterator age across all partitions was less than 100 ms. Provisioned concurrency compressed cold-start InitDuration by more than 98 percent, and above approximately 400 requests per second, the configuration proved more economical than pure on-demand execution—an outcome consistent with Hassan et al.'s economic synthesis [4] and with the optimisation guidelines of Bardsley et al. [5]. Storage conversion to Parquet and subsequent lifecycle tiering ensured that scan volumes and S3 charges grew linearly with throughput, corroborating the storage-efficiency prescriptions of Shojaee Rad and Ghobaei-Arani's pipeline taxonomy [10].

8 Future Work

Despite its recognized successes, the described study was limited to one public cloud region and only in the synthetic replay of historical air-quality data, based on stateless per-record transformations. Based on this, a number of extensions should be mentioned in further research:

- A first recommendation is that future work should consider the integration of pre-warming or machine-learning-supported pre-warming and de-scaling methods strategies–systems that have historically been classified, but not rarely prototyped in optimisation surveys, like Golec et al. and Ghorbian and Ghobaei-Arani. This would further optimize the cold-start- cost trade-off past fixed provisioned concurrency.

- Second, the elasticity mechanisms of the ingestion layer proposed by Werner and Tai should be augmented by real-time shard auto-scaling, which is predicated on adaptive partition-key hashing and traffic predictions, to decrease the variance of iterator-age in a scenario of extreme spikes.

- Third, moving parts of the pipeline into edge or fog layers using lightweight runtimes, as proposed in systematic mappings of Kjorveziroski et al. and Batool and Kanwal would measure latency dividends relative to the management overheads reported by Perez et al. on-premises OpenFaaS deployments.

- Fourth, adding statefulness to the use case, e.g., with sliding-window anomaly detection or reinforcement-learning traffic-signal control, would push Lambda beyond its 15-minute limit and encourage exploration of state-externalisation patterns discussed by Baldini et al. and the deviceless paradigm presented by Merlino et al.

Finally, cross-cloud replication (e.g., Azure Functions and Google Cloud Functions) would test the generalisability of the cost–latency envelope and deepen comparative insight across vendor ecosystems, thereby advancing the universal FaaS substrate vision articulated by Jonas et al. By these means, further study can extend the current contribution beyond a prototype in one city to a multi-tier, multi-cloud and smart city network in smart urban environments.

GitHub link: https://github.com/Geethanjali1919/Air-Quality-Data/tree/main?tab=readme-ov-file 8.

References

[1] E. Jonas et al. "Cloud Programming Simplified: A Berkeley View on Serverless Computing". In: *arXiv preprint arXiv:1902.03383* (2019).

[2] I. Baldini et al. "Serverless Computing: Current Trends and Open Problems". In: *Research Advances in Cloud Computing*. Springer, 2017.

[3] H. Shafiei, A. Khonsari and P. Mousavi. "Serverless Computing: A Survey of Opportunities, Challenges, and Applications". In: *ACM Computing Surveys* 54.11s (2022), pp. 1–32.

[4] H. B. Hassan, S. A. Barakat and Q. I. Sarhan. "Survey on Serverless Computing". In: *Journal of Cloud Computing* 10.1 (2021), p. 39.

[5] D. Bardsley, L. Ryan and J. Howard. "Serverless Performance and Optimization Strategies". In: *Proc. IEEE Int'l Conf. on Smart Cloud (SmartCloud)*. 2018, pp. 19–26.

[6] S. Werner and S. Tai. "A Reference Architecture for Serverless Big Data Processing". In: *Future Generation Computer Systems* 155 (2024), pp. 179–192.

[7] G. Merlino et al. "FaaS for IoT: Evolving Serverless towards Deviceless in I/O Clouds". In: *Future Generation Computer Systems* 154 (2024), pp. 189–205.

[8] M. Golec et al. "Cold Start Latency in Serverless Computing: A Systematic Review, Taxonomy, and Future Directions". In: *ACM Computing Surveys* (2024).

[9] M. Ghorbian and M. Ghobaei-Arani. "A Survey on the Cold Start Latency Approaches in Serverless Computing: An Optimization-Based Perspective". In: *Computing* 106.11 (2024), pp. 3755–3809.

[10] Z. Shojaee Rad and M. Ghobaei-Arani. "Data Pipeline Approaches in Serverless Computing: A Taxonomy, Review, and Research Trends". In: *Journal of Big Data* 11 (2024), p. 82.

[11] P. Benedetti et al. “Experimental Analysis of the Application of Serverless Computing to IoT Platforms”. In: *Sensors* 21.3 (2021), p. 928.

[12] A. Pérez et al. “On-Premises Serverless Computing for Event-Driven Data Processing Applications”. In: *Proc. IEEE Int’l Conf. on Cloud Computing (CLOUD)*. 2019, pp. 414–421.

[13] H. Tabrizchi and M. K. Rafsanjani. “Cloud, Fog, and Serverless Computing Environment for Intelligent Sensor Networks in Smart Cities”. In: *Digital Twin and Blockchain for Sensor Networks*. Elsevier, 2025, pp. 17–35.

[14] M. Pandey and Y. W. Kwon. “FuncMem: Reducing Cold Start Latency in Serverless Computing through Memory Prediction and Adaptive Task Execution”. In: *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*. 2024, pp. 131–138.

[15] N. Saravana Kumar and S. Selvakumara Samy. “Cold Start Prediction and Provisioning Optimization in Serverless Computing Using Deep Learning”. In: *Concurrency and Computation: Practice and Experience* 37.4-5 (2025), e8392.

[16] A. Panda and S. R. Sarangi. “FaaSCtrl: A Comprehensive-Latency Controller for Serverless Platforms”. In: *IEEE Transactions on Cloud Computing* (2024).

[17] N. Buitrago Suárez. *PARSEC: An Adaptive and Efficient Platform for Reducing Cold Start in Serverless Computing*. Preprint. 2024.

[18] Kokkoori A. Anilkumar. “A Novel Mechanism for the Reduction of Latency and Cost in AWS Lambda Calls”. PhD thesis. Dublin: National College of Ireland, 2024.

[19] Z. Zhou, Y. Zhang and C. Delimitrou. “Aquatope: QoS-and-Uncertainty-Aware Resource Management for Multi-Stage Serverless Workflows”. In: *Proc. 28th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems*. Vol. 1. 2022, pp. 1–14.

[20] A. Kumari, R. K. Behera and B. Sahoo. “Mitigating Serverless Tail Latency: A Comprehensive Study of Factors and Strategies”. In: *2023 OITS Int. Conf. on Information Technology (OCIT)*. IEEE. 2023, pp. 369–374.

[21] P. Gackstatter, P. A. Frangoudis and S. Dustdar. “Pushing Serverless to the Edge with WebAssembly Runtimes”. In: *2022 IEEE Int. Symp. on Cluster, Cloud and Internet Computing (CCGrid)*. 2022, pp. 140–149.

[22] R. B. Roy, T. Patel and D. Tiwari. “Icebreaker: Warming Serverless Functions Better with Heterogeneity”. In: *Proc. 27th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems*. 2022, pp. 753–767.

[23] J. Stojkovic et al. “Ecofaas: Rethinking the Design of Serverless Environments for Energy Efficiency”. In: *Proc. 51st Int. Symp. on Computer Architecture (ISCA)*. 2024, pp. 471–486.

[24] V. Kjorveziroski, S. Filiposka and V. Trajkovik. “IoT Serverless Computing at the Edge: A Systematic Mapping Review”. In: *Computers* 10.10 (2021), p. 130.

[25] I. Batool and S. Kanwal. *Serverless Edge Computing: A Taxonomy, Systematic Literature Review, Current Trends and Research Challenges*. Preprint. 2025.