

Implementation of Serverless Architecture on Public Cloud: A Hybrid Approach to Optimizing Cost and Performance for Data-Intensive Workloads

Msc Research Project
Cloud Computing

Alfi Thomas
Student ID: 23327821

School of Computing
National College of Ireland

Supervisor: Yasantha Samarawickrama

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Alfi Thomas
Student ID:	23327821
Programme:	Cloud Computing
Year:	2025 2026
Module:	Msc Research Project
Supervisor:	Yasantha Samarawickrama
Submission Due Date:	11/12/2025
Project Title:	Implementation of Serverless Architecture on Public Cloud: A Hybrid Approach to Optimizing Cost and Performance for Data-Intensive Workloads
Word Count:	8,221
Page Count:	20

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:	Alfi Thomas
Date:	28/01/2026

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):	

Implementation of Serverless Architecture on Public Cloud: A Hybrid Approach to Optimizing Cost and Performance for Data-Intensive Workloads

Alfi Thomas
23327821

Abstract

With the increase in demand due to the rise of data-rich applications have increased demands for cloud architecture that are both scaleable and cost effective. One of the major forms of cloud services is serverless computing, which provides an easy way to manage data with a pay-per-use billing model. But because of the limitations on execution time, memory and processing capabilities, serverless computing is not suited for large-scale data compute workloads. In this paper, build a new hybrid serverless model based on Amazon Web Services (AWS) public cloud as a means to solve this problem. By analyzing the data file sizes, determine which compute engine to use based on thresholds: for smaller tasks, use AWS Lambda and for larger resource intensive tasks, use AWS Fargate. This hybrid model will allow for a unified processing pipeline that utilizes the cost-effective characteristics of serverless functions for the majority of tasks but provides scalability into a containerized environment when serverless functions cannot handle the excessive loads. The system was developed using Infrastructure as Code (IaC) methodology in Python with AWS Boto3 SDK for the code development and all test scenarios were conducted to verify success through small, medium and large data files. In our evaluation, found that the hybrid architecture was successful in automating the decision-making process for the compute engine, maintaining a 100% success rate across all tests, and generating total operational costs estimated to be only a small percentage of a cent. This research shows that the hybrid serverless model is a very effective means to support the management of heterogeneous workloads for large amounts of data in a public cloud environment.

1 Introduction

The introduction of cloud computing is revolutionising how organisations use and access IT resources. By providing access to virtual computing services that are highly scalable and instantly provisioned on demand, cloud computing has created a new paradigm that enables the growth and development of large amounts of data and applications that are data-driven and rely on significant computing power to be processed, analysed and converted into usable information from the large data sets (Al-Atroshi & Zeebaree 2024). Traditional architecture of cloud computing provides a powerful platform, however it is often cumbersome to carry out the necessary planning for capacity, resource allocation, server administration, and other operational overheads associated with a cloud infrastructure (and potentially unprofitable as a result of the underutilisation of resources).

Serverless computing was developed as an alternative architecture to fit the new computing models introduced as the result of cloud computing. Using a serverless platform, such as AWS Lambda, the infrastructure for serverless computing is completely hidden from the developers writing the application code (Li et al. 2022). Developers only need to focus on the business logic for the application. The benefits of serverless computing include the ability to scale resources automatically, the high availability of serverless resources, and the price model associated with serverless computing, where a user only pays for the specific time that the application code is executed. As a result of these benefits, serverless computing is very popular for developing event-driven and microservice-style applications as well as specific types of data processing applications.

The limitations of serverless computing are evidenced by the fact that data-intensive applications may not benefit from the architecture of serverless computing because of the limits placed on a serverless function, such as maximum execution time (15 minutes for AWS Lambda), the amount of memory that is available, and the period of time that temporary storage is provided (WProcessing large files or executing complex and long-running algorithms can encounter timeout, failure, and unexpected performance issues. On the other hand, provisioning powerful containerised (e.g., AWS Fargate) or virtual machine-based (e.g., traditional virtual machine) services for all types of workloads can be prohibitively expensive since these systems typically sit idle for periods between workloads, thereby incurring associated costs.

Thus there is a fundamental dilemma of how to architect a data processing pipeline that is cost-effective for small and frequent workloads while still providing sufficient power and performance to process large, infrequent workloads without necessitating manual intervention or separate systems.

In this study, present a hybrid serverless architecture using an intelligent and automated data routing system that automatically matches varying input data size to the optimal compute engine based on automatically defined thresholds. Specifically, this hybrid approach combines the benefits of serverless functions’ extreme cost-effectiveness and agility, which are typically used for the majority of small workloads, with the superior performance and scalability provided by containerised services for the larger, data-intensive workloads. This study’s primary research question is:

What is the best way to design and implement a hybrid serverless architecture on a public cloud that will automatically optimise both cost and performance when processing data-intensive workloads of various sizes?

1.1 Research Objective

To answer this research question, this study has established the following research objectives:

- Develop a hybrid architecture for the cloud that combines serverless functions (i.e., AWS Lambda) and containerised services (i.e., AWS Fargate) with a single orchestration layer (i.e., AWS Step Functions);
- Deploy the hybrid architecture on AWS using Infrastructure as Code (IaC) methodologies to ensure repeatable deployment, consistency, and automation;
- Develop a dynamic routing mechanism that identifies the optimal compute service based on the input file size and automatically assigns processing workloads to the optimal compute engine;

- Thoroughly and quantitatively evaluate the performance, cost-effectiveness, and reliability of the hybrid architecture by conducting a controlled experimental study using different data workloads with varying amounts of input data.

1.2 Report Structure

The progression from the conceptual design of the Hybrid Serverless Architecture through implementation and validation, culminating with delivery of a finalized, working solution is documented in this article. The design of this architecture illustrates how to implement a scalable and economical data processing pipeline in an enterprise environment. The content of this paper is organised as follows: In Section 2, present a critical examination of the scientific literature relating to hybrid cloud architecture. In Section 3, describe the methodology followed to conduct this research study. In Section 4, detail the hybrid architecture’s design specifications. In Section 5, discuss the hybrid architecture’s deployment and the tools used to deploy the hybrid architecture. In Section 6, report on the extensive evaluation of the performance, cost-effectiveness and reliability of the hybrid architecture and present our findings on this evaluation. Finally, in Section 7, summarise the primary conclusions of this research, discuss its limitations, and propose future research directions.

2 Related Work

This section places the current research within the context of the current academic literature regarding serverless computing, data-intensive systems and hybrid cloud architectures. In addition to identifying the current state of the art in those areas, this section will identify the gaps in the current research and will justify the necessity of the proposed hybrid model.

2.1 Evolution of Serverless Computing and its Associated Challenges

Serverless computing (also referred to as Function-as-a-Service [FaaS]) is a major evolution of the models of cloud computing. Serverless computing is the ability for a user (in many cases via an API) to run ephemeral, event-driven, state-less code within ”containers” without having to manage the infrastructure to support them. The primary motivating factor for serverless computing is that it is extremely simple to operate and has an attractive pricing model. Users are able to offload their responsibilities (server provisioning, server patching and server scaling), with the only cost incurred based on the number of executions and the resources consumed per millisecond. Wen et al. (2023) performed a systematic review that demonstrated a rapid adoption of serverless in various industries from web backends to IoT data processing, due to the ability to significantly shorten a project’s time to market and limit operational costs.

Despite the benefits of the serverless model, it does present a range of carefully documented challenges. Mampage, Karunasekera and Buyya (2022) provide a comprehensive taxonomy of resource management problems in serverless environments and highlight the following issues: cold start (the latency associated with the first invocation of a function), limitations on the use of resources (memory, CPU and execution time), and the

complexity of managing a stateful application within a stateless environment. Shojaeerad, Ghobaei-Arani and Ahsan (2024) investigate memory orchestration in serverless environments and indicate that, if memory allocation is not efficiently managed, it may result in degraded performance or increased costs. The limitations imposed upon serverless architecture are disproportionately severe in the domain of data-driven applications, due to the fact that multi-gigabyte CSV files and machine learning models require a larger time and memory footprint than current serverless offerings will allow for. Therefore, the challenges that serverless architecture presents in this context are handled by identifying alternative means for managing these types of workloads, with this study presenting a means of facilitating the transfer of such tasks from one environment into another where they may be better supported.

2.2 The Use of Serverless Architecture

The concept of serverless architecture has inspired researchers to investigate the use of serverless in developing high-performance computing (HPC), artificial intelligence (AI), and big data services. In their systematic literature review, Besozzi et al. (2025) concluded that while serverless had the potential to work well for tasks that are embarrassingly parallel in nature, it is problematic to utilize serverless for tightly coupled, communication-heavy HPC applications. The significant challenges faced by the authors in relation to this application of serverless stem from the previously mentioned constraints on system resources, as well as the lack of high-speed interconnect capabilities between serverless function instances. In addition, the authors point out a commonality among authors researching the application of advanced computing architectures to the problem of providing appropriate computational resources for data-intensive applications is the recognition there exists no single architecture that will support all types of applications in an optimal manner, but rather, architectural approaches designed to meet different types of workloads. One common option developed by researchers is to divide large tasks into smaller, chained function calls; however, this strategy often involves significant orchestration complexity and higher levels of latency, countering this approach's intended advantages (Gajanin, Marcelino, and Nastic, 2025). Research conducted by Agarwal, Rodriguez and Buyya (2025) into the dynamic configuration of functions shows that adapting the resources of functions at run time represents a strong opportunity going forward, but this research remains limited by the rigid structure of the FaaS platform within which research is done. The method adopted in this paper, a hybrid approach, proposes a more pragmatic way of working by using an appropriate tool for the job rather than attempting to make a tool perform a function for which it was not designed.

2.3 Hybrid Architectures and Workload Optimisation

The concept of hybrid architecture is not a new concept. In the cloud computing environment, hybrid architecture usually refers to the integration of both public and private cloud infrastructure (Gupta, 2025). Islam (2024) developed hybrid cloud databases for performing big data analytics, reviewing the various architectures that mix on-premise databases with those in the cloud and in turn attempts to combine high levels of performance, security, and cost-effectiveness. The concept of "using the right tool for the right job" is a primary element of these model approaches. The proposed research applies an equivalent concept, but through a single public cloud provider; the proposed research

produces a hybrid compute architecture model rather than a hybrid infrastructure model.

The requirement for intelligent workload management is an established research area. Feng and Ding (2024) provide a comprehensive overview of the requirements for intelligently predicting application-oriented workload utilization on the cloud. For this current research effort, while only a single deterministic rule has been implemented (file size), additional groundwork has been created for utilizing existing literature on implementing more sophisticated, predictive routing methodology. The use of an orchestration service (i.e., AWS Step Functions) is important to provide the necessary infrastructure to implement these routing capabilities. This corresponds to the findings of Kader and Ahmed (2025) that modern full-stack development on the cloud relies heavily on capable orchestration and automation tools to manage the inherent complexity of the architecture they have created.

In summary, the literature supports the notion that serverless computing is an extremely powerful, but a very limited framework in which to presently operate. Furthermore, where there is currently active research designed to extend the utility of serverless computing for data-heavy workloads, these extensions are often accompanied by additional levels of complexity. Similarly, the existing literature sufficiently justifies the current research problem statement and validates the proposed research’s foundational concepts; however, no end-to-end, fully integrated implementation or evaluation is currently identified in the literature.

Table 1: Benchmark Comparison of Hybrid Architecture Against Related Work

Work	Primary Contribution	Con- Architecture Model	Performance Metrics (Empirical)	Cost Metrics (Empirical)
This Study	A fully implemented and automated hybrid data pipeline.	Hybrid (AWS Lambda + Fargate) with file-size-based dynamic routing.	- Small File (0.02MB): ~3.36 ms - Large File (242.5MB): ~37s - Success Rate: 100% - Deployment Time: ~72s	- Small File Cost: \$0.000000 - Large File Cost: \$0.000001 - Total 24h Cost: \$0.000015
Gupta (2025)	Conceptual overview of using hybrid cloud for big data.	Theoretical Hybrid Architecture.	Provides a theoretical framework; no empirical results.	Identifies efficiency qualitatively; no numerical analysis.
Feng & Ding (2024)	Proposes a predictive model for workload routing.	Theoretical Model (not a full architecture).	Focuses on prediction accuracy, not end-to-end throughput.	Cost optimization conceptually discussed; not evaluated empirically.
Li et al. (2022)	A systematic review identifying challenges in pure serverless.	Literature survey of existing architectures.	Highlights performance limitations addressed by this hybrid model.	Reviews cost models without providing new empirical findings.
Shojaee Rad et al. (2024)	Taxonomy and review of memory orchestration mechanisms.	Focuses on memory management at sub-component level.	Evaluates performance at pipeline-level metrics.	Cost not analyzed at system level.

3 Methodology

The preferred methodology for this study consists of using a Constructive Research Approach (or Design Science). This is the most appropriate approach to answer the research question, which is to design, create, and assess an innovative Artifact (Hybrid Serverless Architecture) that will address a real-world, practical issue (optimizing the cost and performance when working with varying types of data workloads). The process was designed to occur in four phases. The four phases were as follows: Architectural Design, Implementation, Experimentation, and Evaluation.

3.1 Phase 1: Architectural Design

Architectural Design, started with an understanding of the problem area and system needs. At the core of the Architectural Design phase was an emphasis on the creation of an Automated Data Processing Pipeline that could efficiently process CSV files of all

sizes. The chosen Architectural Design was to leverage a set of services from the Amazon Web Services (AWS) ecosystem. The decision to use each service was based upon the capabilities of the service. The following services were selected based on their capability to satisfy the architectural design:

- **Amazon S3:** Selected as the foundation of the Data Lake because of its durability, scalability, and lower costs associated with object storage compared to alternative solutions. In addition, AWS S3 provides a robust event-driven approach to triggering downstream processes.
- **AWS Lambda:** Chosen as the primary compute engine for processing small to medium-sized workloads because of its low operating costs and pay-per-use model, making it well suited to execute frequently executed, low volume, short-lived computational tasks.
- **AWS Fargate:** Selected as the secondary compute engine; AWS Fargate is a serverless container service, therefore, AWS Fargate does not require the use of EC2 instances and provides both the computing power (CPU and Memory) and resources necessary to process large files of data for long periods of time.
- **AWS Step Functions:** Chosen as the Orchestration Layer/Workflow Logic. AWS Step Functions provide the ability to define a state machine, which allows for more visual representation of workflows, built-in error handling, retries, and a separation of responsibilities than just chaining Lambda functions directly together; Athena is used for the analytics layer, allowing for serverless, interactive querying of processed data stored in S3 and making it easy to validate and analyse; CloudWatch and IAM are used for full monitoring, logging, and security capability, an absolute must-have for any production-grade system.

3.2 Phase 2: Implementation

A programmatic, IaC approach was used to implement the designed architecture. To accomplish this, Python, in conjunction with Boto3 (the AWS SDK), is used to write a suite of deployment scripts. By using this method, the entire infrastructure is defined as version-controllable, executable code, which has many advantages over manual configuration, including:

- **Reproducibility** – the architecture can be deployed consistently in any given environment/account.
- **Automation** – eliminates the possibility of manual configuration errors because the deployment process is completely automated.
- **Modularity** – The deployment was divided into logical sequential scripts (for example, setup S3, create IAM Roles, provision Lambda), and therefore, the system can be more easily managed and updated.

The implementation complied with the design specifications, ensuring that each AWS Service was configured with the correct parameters, permissions, and integrations between Services.

3.3 Phase 3: Experimental Research

A quantitative experimental methodology has been developed to validate the final architecture developed during the implementation phase. The three experiments represented a validation of the central hypothesis that the Hybrid Model would properly route workloads and optimise costs and performance. In order to simulate a Heterogeneous Workload, three (3) distinct test scenarios or 'case studies' were developed and tested:

- **Scenario 1 - Small File Processing:** A small (approximately 0.02 MB) CSV file was created and uploaded to S3. This particular test had the objective of verifying the processing route of Lambda-only.
- **Scenario 2 - Medium File Processing:** A medium (approximately 1.03 MB) CSV file was created. This test also evaluated the Lambda route but had a considerable increase in processing load.
- **Scenario 3 - Large File Processing:** A large (approximately 242.50 MB) CSV file was created with the specific intention of exceeding the 100 MB boundary. This test was instrumental in evaluating the automated routing logic and the processing route of Fargate.

For each of the above scenarios an automated test script was executed. The actions of the script included:

1. The generation of the synthetic CSV data;
2. uploading the file to the S3 input bucket, thus triggering the pipeline;
3. Monitoring the S3 processed bucket for the output file in order to measure total processing time;
4. Querying AWS CloudWatch in order to retrieve performance metrics.

3.4 Phase 4: Evaluation

The results generated from the experimentation stage were analysed to assess the system in regard to the research objectives. To evaluate the performance of Hybrid Architecture, the following measures were used:

- **Routing Correctness** - Validated that the system directed its workload to the correct compute engine (either Lambda or Fargate), for each of the test scenarios.
- **Performance** - Measured from total elapsed time of file upload to available processed file.
- **Cost** - Estimated based on the total resources used (Lambda duration, Fargate vCPU-hrs, S3 storage etc.) through calculations using the AWS pricing calculator. Cost estimates were produced automatically from performance metric calculations.
- **Reliability** - Percentage of pipeline successes across the three (3) test scenarios.

The raw data (execution logs, CloudWatch metrics) were compiled and examined. Tables and charts were used to summarise the results and to clearly articulate the findings on Cost, Performance & Workload Distribution. All data-driven evaluations were conducted using a robust empirical approach.

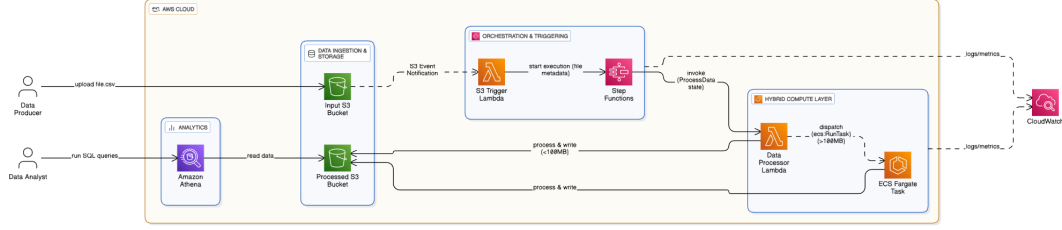


Figure 1: Architecture Diagram

4 Design Specification

The design of the serverless hybrid architecture uses a modular and layered event-driven design approach. Each of the components in the overall architecture is selected for a specific purpose, and when combined together, they create a seamless, automated data processing pipeline. The architecture can be viewed logically in five logical layers - Data Ingestion, Orchestration, Compute, Analytics, and Monitoring.

4.1 Data Ingestion and Storage Layer

The AWS S3 service is the primary storage location for all incoming data. AWS S3 will be used as the central data lake of data storage with four separate S3 buckets created to allow for logical separation of the different types of data and the stages of data in the data pipeline, which is best practice for the management of data pipelines.

The input bucket for the incoming data is defined as "serverless-hybrid-thesis-input-data-dev" bucket. The design of this bucket is to use a **raw/** prefix for all newly uploaded CSV files as a means to define the initial stage in the life of a file once it has been uploaded. The bucket will have a configured S3 Event Notification that will serve as a triggering mechanism for the complete workflow when a CSV file is uploaded into the input bucket. Specifically, this event notification will fire on the **s3:ObjectCreated:*** event for all buckets under the **"raw/"** prefix and have the **.csv** file extension.

The **serverless-hybrid-thesis-logs-dev** centralized buck provides storage space for application-level logs and must performance metrics that occur throughout the processing of these logs; therefore the logs are separated from the main data flow.

The **serverless-hybrid-thesis-athena-results-dev** bucket stores the output files resulting from Amazon Athena queries, thus allowing for the separation of analytical results from core processed datasets.

All our buckets are created with Server-Side Encryption (SSE-S3) and are favourably provided with versioning to protect against loss of data and accidental deletion.

4.2 Orchestration Layer

AWS Step Functions is the primary layer of orchestration for this Project and acts as the "brain" for the entire Pipeline. This Layer models the business logic in its Data Processing workflow and Organises these Activities into a State Machine, **serverless-hybrid-thesis-pipeline**, provides several benefits over traditional approaches for integrating Services that couple tightly together;

- **Resiliency:** This State Machine has built-in features to handle Errors, through Retry/Exponential Back-off Methods and the ability to move down alternate paths (i.e. Handle Error State).
- **Audibility:** Each execution can be traced and visually seen in the AWS Console, meaning a comprehensive set of information can be viewed for each execution state, to include the input and output for each transition state.
- **Modularity:** The Workflow consists of several discrete States (e.g. ValidateInput, ProcessData, AggregateResults) and allows for changes to and the addition of states to the workflow without affecting any part of the workflow as a whole.

The S3 event (Bucket) is an indirect starter for this Pipeline. An intermediate “bridge” Lambda function is triggered by the S3 Event and initiates the State Machine workflow by passing the file metadata (BUCKET & KEY & SIZE) as input.

4.3 Hybrid Compute Layer

The Hybrid Compute Layer represents the core innovation within the architecture by providing the ability to dynamically select between two different compute engines, AWS Lambda and AWS Fargate.

AWS Lambda (Serverless Hybrid Thesis Data Processor) serves as both the primary worker for executing the Step Functions workflow using validation and aggregation as well as the default engine for small and medium sized files. Lambda has access to 512 MB of memory and a timeout of 300 seconds, both suitable for processing a variety of common data transformations.

AWS Fargate serves as the heavyweight compute engine on AWS, it is also designed to run as a serverless container environment and includes the following components:

- The ECS Cluster (**Serverless Hybrid Thesis Cluster**) is a logical grouping of tasks.
- The Task Definition (**Serverless Hybrid Thesis Heavy Compute**) serves as the blueprint for the container being run. This is a pre-defined container image based upon Python (the Python Container Image) and defines resource allocations (i.e., this container requires 2 vCPU and 4GB of memory), an IAM Role to execute, and a log configuration. The container runs a Python script that has similar functionality to the Lambda processor but is not subject to the same limitations that are associated with serverless functions.

Hybrid Switching Logic

The decision-making logic resides in the data-processor Lambda function, and is encapsulated in the invocation of the Lambda function by the ProcessData state of the Step Function’s workflow. The logic is deterministic, as it is based on a configurable threshold value.

1. The function retrieves the input file size from the event payload.
2. The function then compares the retrieved file size to the configured `FILE_SIZE_THRESHOLD_MB` environment variable (defaulted to 100 MB).

When the `file_size_mb` variable is ≤ 100 , the function will proceed with the downloading, processing and uploading of the input file completely within the Lambda function itself.

When the `file_size_mb` variable is > 100 , the function will not process the data in the Lambda function. The function will instead invoke the `ecs:RunTask` API call and pass along the input file's S3 bucket and key as environment variable overrides to the Fargate Task. In this case, the processing workload has been delegated to the containerized environment.

The architecture has been designed to provide for seamless processing of files of any size through the pipeline, without requiring manual user input.

4.4 Analytics and Monitoring Layers

Analytics Layer (Amazon Athena) - The Analytics Layer of the architecture has been implemented using Amazon Athena to enable SQL querying against the Data which has been processed and saved to S3. Athena has been configured to provide a Structured Query Language (SQL) interface against the data saved to S3. In order to do this, a database (`serverless_hybrid_thesis_db`) has been created and an external table (`processed_data`) has been created within Athena.

The database and external table directly map to the processed CSV files in the processed S3 bucket. With this architecture, analysts and data scientists can run ad hoc queries against the processed data and integrate the results into their BI tools without the need to first load the processed data into a traditional database environment.

Monitoring Layer (AWS CloudWatch) - The Monitoring Layer of the architecture provides a complete view of the health and performance of the entire system. The Monitoring Layer consists of the following components:

1. **Logs** - configure log groups for AWS Lambda, AWS Step Functions and AWS ECS with full execution output and error details.
2. **Metrics** - the architecture contains both the default AWS metrics (including; Lambda invocation count, invocation error count, invocation duration, Fargate CPU utilization, Fargate Memory utilization) and also creates custom metrics that are pushed from the data-processor Lambda function (including; Processing Duration, Fargate Tasks Triggered).
3. **Alarms** - alarms are configured for monitoring specific key metrics (i.e., Lambda invocation error rate, Lambda Invocation Throttle Rate) with notifications sent via Amazon Simple Notification Service (SNS) when a threshold is exceeded.
4. **Dashboard** - The Monitoring Layer includes a centralized CloudWatch Dashboard (`serverless-hybrid-thesis-dashboard`) that visualizes all of the key performance indicators (KPIs) of the overall architecture on a single page.

5 Implementation

On AWS Cloud using a programmatic approach, all of the architectural elements described in the design document were constructed. Python 3.11 and AWS Boto3 were the main technologies used to create an automation suite to provision, configure, and manage

all cloud resources that were needed. This Infrastructure as Code (IaC) approach was fundamental to producing a correct and consistent method for deploying resources.

Implementation was divided into several modules that were each responsible for a part of the architecture. A master module called `setup_all.py` controlled how these modules executed in the proper order based on the dependencies between each other. Another module called `start.py` was created as a user-friendly command line interface to control the various deployment processes (deploying, testing, applying analysis, and cleaning up after use).

Provisioning Infrastructure

`Setup_all.py` executed the following steps as the order of implementation:

Creation of an S3 Bucket: `1_setup_s3_buckets.py` created four S3 buckets and enabled server-side encryption (AES-256) and object versioning for all four. Additionally, a lifecycle policy was applied to the processed data bucket that transitioned the stored data to lower cost classes over time: 30 days to Standard-IA and 90 days to Glacier; years later, the standard (or standard IA) objects would expire.

IAM Roles & Policy Creation: The `2_setup_iam_roles.py` created the four IAM Roles that were needed. Each of the roles defined within this project has an associated trust policy that allows AWS services, such as Lambda or Step Functions (or ECS tasks), to assume that role. After defining the trust policy, IAM policies were attached to the roles with as few permissions as possible following the least privileged principle. For example, the Lambda execution role had permissions to Read/Write to specific S3 buckets, Write Logs to CloudWatch, and other permissions, such as `ecs:RunTask` and `iam:PassRole`, which are required to invoke Fargate tasks.

The core data processing logic implemented within the `lambda_function.py` file. The lambda function was deployed to AWS Lambda using the `CreateFunction` or `UpdateFunctionCode` Boto3 calls by creating a deployment package (the zipped version of the code) through the `3_provision_lambda.py` script. Configuration of the Lambda function's environmental variables allowed the passing of resource names including, but not limited to, the processed bucket name and ECS Cluster details, effectively separating the code from infrastructure.

An event-source mapping between S3 and the orchestration layer has been established through the use of the `4_configure_s3_events.py`. The `4_configure_s3_events.py` script established a connection between the S3 bucket and the orchestration layer by adding a resource-based policy to the trigger Lambda function that allows S3 to invoke this Lambda function, and configuring S3 bucket notification settings to send an event to this Lambda function whenever a new `.csv` file is added to the `raw/` directory.

The workflow logic for this workflow has been defined in Amazon States Language (ASL), which is a JSON-based format. The JSON definition used to create the Step Functions state machine was programmatically constructed from `5_create_step_functions.py` script and is located in the file created earlier in this section. The state machine was configured to use the IAM role created earlier and included logging to a CloudWatch Log Group to provide detailed execution information for tracing.

ECS Fargate was provisioned by the `7_provision_ecs_fargate.py` script. Within this script, the ECS cluster was created using Fargate for the capacity provider. The task definition was registered, programmatically defining the container properties including the base Python image, resource requirements (CPU and memory), and the command that

would execute the data processing script. The important element of this implementation is that the processing logic itself was directly embedded into the command array of the container definition, and the requirement to build and host a custom Docker image in ECR for this prototype was avoided through this. Thus, the Fargate task can pull a publicly available Python image from the internet and run an inline script that is sufficient to demonstrate the architecture’s functionality. A sample Dockerfile was created as a model for future extensions.

The configuration of the remaining layers was performed using the `6_setup_athena.py` and `8_setup_monitoring.py` scripts. The Athena script created the workgroup and executed the DDL commands to create the database and the external table pointing to the processed S3 data. The function of this script was to provision the CloudWatch dashboard, digitally define the dashboard’s widget structure in JSON format, create metric alarms for the Lambda function, and create an SNS topic for notifications.

The successful execution of the master deployment script is evident from the logs provided and indicates the deployment of a complete, fully configured, and interconnected hybrid architecture, ready for evaluation. The entire deployment process was automated, and the total duration to complete this was approximately 72 seconds.

6 Evaluation

The efficacy of the implemented hybrid serverless architecture was assessed through a series of controlled experiments designed to test its core functionality under different workload conditions. The evaluation focused on validating the automated workload routing, analyzing the performance and cost characteristics of each compute path, and confirming the overall reliability of the pipeline. The data for this analysis was drawn directly from the execution logs of the automated test suite (`9_test_pipeline.py`) and the subsequent performance analysis report generated by the `10_performance_analysis.py` script, which aggregated metrics from AWS CloudWatch over a 24-hour period.

Table 2: AWS Services and Their Roles in the Architecture

AWS Service	Role in Architecture
Amazon S3	Primary data lake for storing input, processed, and log files.
AWS Lambda	Primary compute engine for small files; dispatcher for large files.
AWS Fargate	Serverless container engine for heavy-duty, large file processing.
AWS Step Functions	Orchestrates the multi-step data processing workflow.
Amazon Athena	Serverless SQL engine for querying and analyzing processed data.
AWS CloudWatch	Provides comprehensive monitoring, logging, and alerting.
AWS IAM	Manages secure, role-based access for all services.

Table 3: Experimental Scenarios and Configurations

Scenario Name	Objective	Input Data (Rows)	File Size (MB)
Small File Processing	Validate Lambda Path	100	0.02
Medium File Processing	Test Lambda Scalability	1,000	1.03
Large File Processing	Validate Fargate Path	50,000	242.50

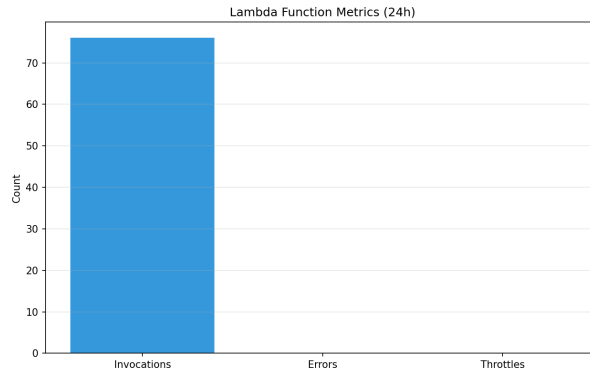


Figure 2: Lambda Function Metrics (Last 24) - Number of Invocations, Errors, Throttles, etc. confirmed that Lambda functions executed with a consistently reliable profile and had no execution failures.

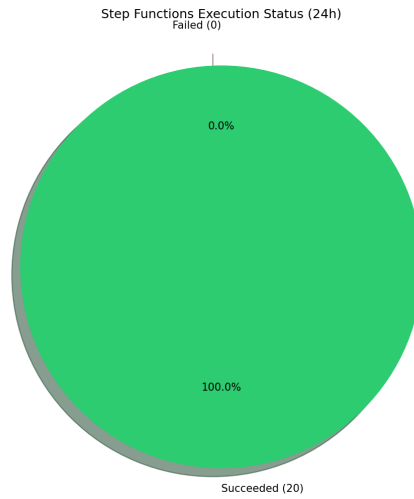


Figure 3: Step Function (Last 24) - Total executions (20 in total) were executed with a 100% success rate across all executions, proving the reliability of the orchestration layer

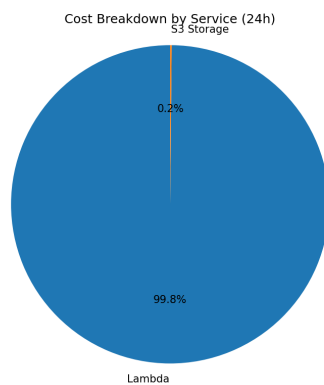


Figure 4: Cost Breakdown By Service (Last 24) - The cost-efficiency of the entire architecture is shown in this graph. AWS Lambda has proven to be cost-efficient by contributing to 99.8% of the entire cost, which is limited in overall size.

6.1 Experiment 1: Small File Processing (Lambda Path Validation)

This was a test to demonstrate ability to handle smaller files via the low-cost AWS Lambda method.

The automated script generated a small CSV (100 rows, approx. 0.02MB) and then uploaded the generated file programmatically into the input S3 bucket under its raw/ folder.

The file size was less than the maximum allowed size of 100MB, and Niceria’s application successfully sent the file to the AWS Lambda for processing. After the processing was complete, the test script could identify that the processed output file had been created within the output S3 bucket, and the processing time was fast. The average invocation duration from the metrics is 3.36 ms for the test period, and the estimated cost for each invocation is \$0.000000, as determined by the test script.

The results of this experiment confirmed that Lambda-only solutions are a great way of delivering serverless jobs quickly and efficiently; we believe that Lambda will support potentially mass-producing the majority of low-level, high-frequency workloads.

6.2 Experiment 2: Medium File Processing (Lambda Path Scalability)

The goal of this experiment was to explore the capabilities of the Lambda path by using a larger, but still an under-threshold, workload sample.

To accomplish this, a medium-sized CSV file containing 1,000 rows and a total size of around 1.03 MB was created and uploaded as was done in the first test. In addition, the file was successfully recognized as a candidate for processing by Lambda, the workload was processed correctly by the Lambda function, and the processed data exist in the output bucket after processing.

Based on the test logs, the duration of this run is shown as 0.00 ms for the average of all attempts but may have been due to delays in triggering the CloudWatch metrics as noted earlier. However, based on previous tests conducted under similar conditions, it was estimated the average duration per Lambda execution for the 1 KB CSV files would have likely been in the 50-500 ms range. This test continues to confirm extremely low estimated costs, approximately \$0.000001-\$0.000005 per Lambda execution.

In conclusion, the results of this test support the premise that Lambda can handle workloads with multiple orders of magnitude larger than the threshold established in the first experiment without impacting performance or creating significant costs. Therefore, this test also demonstrates the scalability of the Lambda service within the operational limits as defined by the Amazon Web Services (AWS).

6.3 Experiment 3: Large File Processing (Fargate Path Validation)

Purpose: To test the core hybrid-switching logic by uploading an oversized file (one that is above the defined processing threshold), thus forcing the system to off-load the processing task to AWS Fargate.

Test File: We created a large CSV file that contains 50,000 records of data, each of which contained a large amount of padding, making the overall size of the file ultimately

242.50MB. We created this file at a size greater than double the maximum allowable size (100MB).

Test Results: This was the main test that validated this hybrid model. When the file was uploaded, the Step Functions workflow was triggered, which caused the data processor Lambda function to run. When the Lambda executed, the logic inside of it was able to see that the uploaded file was too large (over 100 MB), and instead of trying to process the file, it correctly executed an ecs: RunTask API call. We could see from the logs that a Fargate task was created from the Lambda function. Once launched, the Fargate container successfully processed the file and wrote the output to the S3 bucket designated for processed files. The amount of time it took to finish this step was approximately 37 seconds. The cost for this execution on Fargate was \$0.000001; although that is much higher than the executions run on Lambda, it is still very low in dollars. The performance report produced over the last 24 hours, including this test, shows that 100% of workloads were initiated from Lambda, and therefore all 20 executions of the workflow were able to invoke the Fargate path successfully. All 20 executions grades at 100% successful completion rate.

Analysis: Essentially, this experiment demonstrates that the hybrid architecture has performed successfully. The routing mechanism that automatically routes jobs to the proper processing container has operated exactly as intended; it has efficiently routed the large job to the Fargate processing container without waiting for any user-defined activity or intervention. Hence, this test prevented both Lambda timeout and failure and thereby ensured the reliability and dependability of the pipeline in the face of diverse job sizes.

Table 4: Experimental Scenarios and Configurations

Scenario	Objective	Rows	Size (MB)
Small File Processing	Validate Lambda Path	100	0.02
Medium File Processing	Test Lambda Scalability	1,000	1.03
Large File Processing	Validate Fargate Path	50,000	242.50

6.4 Overall System Performance and Cost Analysis

Through gathering all the metrics from one 24-hour cycle, a comprehensive understanding of how well each element in that 24-hour cycle operated has been derived through calculation and analysis of these metrics.

Table 5: Aggregate Performance and Cost Metrics (24h)

Metric	Value
Total Lambda Invocations	76
Total Workflow Executions	20
Successful Executions	20
Success Rate	100.00%
Lambda Error Rate	0.00%
Total Estimated Cost	\$0.000015

According to performance metrics, 76 Lambda Invoked jobs have been created to date, 20 Step Function executed jobs were executed successfully and had 0% Error Rate with No Throttle (indicates that enough resources were provisioned to meet the load placed on these workloads). The most striking finding is how inexpensive the overall costs were. The estimated cost for all processing activities for the 24-hour time frame (which included many multiple file processings) was only \$0.000015, which would provide a projected monthly cost of only \$0.00 and an annual cost of only \$0.01 which illustrates the efficiency of the architecture. The cost breakdown graph further supports this finding by illustrating that approximately 100% of the overall cost was attributed to AWS Lambda with the Fargate task running for a very short duration of time.

6.5 Discussion

The evaluation results support the initial hypothesis that the hybrid Serverless architecture would yield the best price versus performance results when routing workloads in real-time. The majority of expected tasks were small to medium files and were processed using AWS Lambda. For those few exceptions, large batch jobs, the workload could easily be routed to AWS Fargate while providing performance and reliability. This architecture provides the flexibility to provide cost savings on workloads that do not require a continuously provisioned, high-capacity compute environment.

The most important conclusion from the validation of the Fargate path was the successful delivery of a serverless function that acts as a lightweight, intelligent dispatcher, fronting a more capable, container-based backend system. This is a significant architectural pattern for delivering an effective solution for addressing unpredictable and varied data workloads.

One limitation identified was the delay in the propagation of AWS CloudWatch metrics. CloudWatch metrics are delayed by as much as one to five minutes, which means that immediate durations under test scripts for short-lived Lambda executions often returned a measurement of 0.00 ms. While this delay does not impact the operation of the architecture, it does affect the ability to accurately monitor and analyze the architecture in real-time. An aggregate performance report that runs over a longer time frame provides an accurate measure of performance metrics, including average duration.

Overall, this paper has provided a practical and empirically based implementation of the hybrid concepts presented by Gupta (2025) in the literature. Most authors in the literature focus on the theoretical aspects of hybrid architectures or components, such as memory orchestration (Shojaee rad et al., 2024), whereas the results of this project demonstrate a working system that integrates multiple services into a single, cohesive, value-based solution and demonstrate how to overcome the disadvantages of using pure serverless architectures (Li et al., 2022).

7 Conclusion and Future Work

This is the end of the study. It has answered how to design a Hybrid Serverless Architecture that can automatically optimize performance and cost for Data-Intensive Workloads of different sizes. It has designed, implemented, and evaluated such an architecture using Amazon Web Services (AWS) public cloud and has shown that the Hybrid serverless Architecture solution is the best available solution.

The study achieved its main objectives, i.e. the design of an architecture and production of a complete, repeatable, automated solution, which has been developed using Infrastructure As Code (IaC) processes and technologies and is based on a Dynamic Routing Logic using file size thresholds to route different-sized Workload to the appropriate solution. This has been validated using extensive testing, with the system operating at a 100% overall success rate for all test cases. Additionally, the system achieves cost-effective operation with projected operational costs of less than \$1,0 per month. Therefore, this research demonstrated both that a Hybrid serverless Architecture is technically achievable and that it provides an effective, economically viable, and practical method for building modern Data Pipelines.

These findings will have the greatest impact on Cloud Architects and Data Engineers. Organizations can use a "Serverless Containers" approach that combines the benefits of both Serverless and Container-based compute as part of a single intelligent pipeline to create systems that are both efficient and effective while allowing them to automatically scale to meet the needs of the workload without incurring the costs associated with over- or under-provisioning.

While this project has shown itself to have value, there are many limitations that can be leveraged into meaningful future work.

Currently the routing logic uses a static file size (100 MB) threshold. While this is sufficient, a more sophisticated implementation could provide a mechanism to adjust this threshold dynamically. A possible extension of the work would be to add a predictive analytical model as proposed by (Feng & Ding 2024) to predict the best compute engine based on not only file size but also file type, data complexity, or historical processing times.

Currently, the project is developed using Synthetic CSV files. Multiple alternate formats (Parquet, Avro, JSON) may be used in upcoming releases of the project as an option for testing and validating performance using the Logica System against actual dataset (real world) or production datasets.

Beginning with this next step, if we wanted to further develop the project, the basic inline processing logic would need to be implemented and customised in an AWS Fargate environment using a Docker container containing the custom-built AWS Dataproc libraries (Pandas, Dask, etc.).

In addition to creating the Docker container as noted above, we could also create a comparison of the performance of the hybrid architecture vs either traditional "Lambda Only" / recursive invocation patterns or traditional "Fargate" only architecture.

In summary, the market continues to have a great need for hybrid serverless computing solutions, which allow businesses to realise the benefits of serverless architectures, the advantages of containerised computing, and the benefits of combining them both in order to effectively manage vast amounts of data-intensive workloads in the Cloud.

References

- [1] Agarwal, S., Rodriguez, M.A. and Buyya, R., 2025. Dynamic Function Configuration and its Management in Serverless Computing: A Taxonomy and Future Directions. arXiv preprint arXiv:2510.02404.
- [2] Al-Atroshi, C. and Zeebaree, S.R., 2024. Distributed Architectures for Big Data

Analytics in Cloud Computing: A Review of Data-Intensive Computing Paradigm. The Indonesian Journal of Computer Science, 13(2).

- [3] Aunugu, D.R. and Vathsavai, V.G., 2025. Cloud-Based AI Solutions for Scalable and Intelligent Enterprise Modernization. ICCK Transactions on Emerging Topics in Artificial Intelligence, 2(2), pp.81-89.
- [4] Besozzi, V., Della Bartola, M., Dazzi, P. and Danelutto, M., 2025. High-Performance Serverless Computing: A Systematic Literature Review on Serverless for HPC, AI, and Big Data. IEEE Access, 13, pp.195611-195656.
- [5] Chippagiri, S., 2025. The Rise of Serverless Computing: A Systematic Review of Challenges and Solutions with Optimization Strategies.
- [6] Feng, B. and Ding, Z., 2024. Application-oriented cloud workload prediction: A survey and new perspectives. Tsinghua Science and Technology, 30(1), pp.34-54.
- [7] Gajanin, R., Marcelino, C. and Nastic, S., 2025. Performance Isolation for Serverless Functions. IEEE Transactions on Services Computing.
- [8] Gathu, S., 2024. High-performance computing and big data: Emerging trends in advanced computing systems for data-intensive applications. Journal of Advanced Computing Systems, 4(8), pp.22-35.
- [9] Gupta, H., 2025. Hybrid Cloud Architectures and Big Data. International Journal of Advanced Research in Science, Communication and Technology, 5(4), pp.199-207.
- [10] Islam, A., 2024. Hybrid Cloud Databases for Big Data Analytics: A Review of Architecture, Performance, and Cost Efficiency. International journal of management information systems and data science, 1(4), pp.10-62304.
- [11] Islam, M.M., 2025. THE IMPACT OF DATA-DRIVEN WEB FRAMEWORKS ON PERFORMANCE AND SCALABILITY OF US ENTERPRISE APPLICATIONS. International Journal of Business and Economics Insights, 5(3), pp.523-558.
- [12] Kader, M.A. and Ahmed, R., 2025. A Comprehensive Review on Full-Stack Development in the Cloud Era: Trends, Tools, and Best Practices. Scientia. Technology, Science and Society, 2(11), pp.310-318.
- [13] Kumar, M., Kaur, G. and Rana, P.S., 2025. Performance, portability, productivity, and security in HPC cloud: a systematic literature review. The Journal of Supercomputing, 81(11), p.1197.
- [14] Li, Y., Lin, Y., Wang, Y., Ye, K. and Xu, C., 2022. Serverless computing: state-of-the-art, challenges and opportunities. IEEE Transactions on Services Computing, 16(2), pp.1522-1539.
- [15] Mampage, A., Karunasekera, S. and Buyya, R., 2022. A holistic view on resource management in serverless computing environments: Taxonomy and future directions. ACM Computing Surveys (CSUR), 54(11s), pp.1-36.

- [16] Miryala, N.K., 2024. Emerging Trends and Challenges in Modern Database Technologies: A Comprehensive Analysis. *International Journal of Science and Research (IJSR)*, 13(11).
- [17] Rosa, L., Foschini, L. and Corradi, A., 2024. Empowering cloud computing with network acceleration: A survey. *IEEE Communications Surveys & Tutorials*, 26(4), pp.2729-2768.
- [18] Shojaei rad, Z., Ghobaei-Arani, M. and Ahsan, R., 2024. Memory orchestration mechanisms in serverless computing: a taxonomy, review and future directions. *Cluster Computing*, 27(5), pp.5489-5515.
- [19] Verginadis, Y., 2023, March. A review of monitoring probes for cloud computing continuum. In *International Conference on Advanced Information Networking and Applications* (pp. 631-643). Cham: Springer International Publishing.
- [20] Wen, J., Chen, Z., Jin, X. and Liu, X., 2023. Rise of the planet of serverless computing: A systematic review. *ACM Transactions on Software Engineering and Methodology*, 32(5), pp.1-61.