National College of Ireland

# Real-Time Data Processing in Cloud Environments

MSc Research Project
MSc in Cloud Computing

Atharav Deshpande
Student ID: 23269065

School of Computing
National College of Ireland

Supervisor: Mr. Punit Gupta

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

**Student Name:** Atharav Jayant Deshpande……

**Student ID:** 23269065……

**Programme:** MSc in Cloud Computing…… **Year:** …2024-2025….

**Module:** MSc Research Project…….……

**Supervisor:** Mr. Punit Gupta…….……

**Submission Due Date:** 15/09/2025…….……

**Project Title:** Real-Time Data Processing in Cloud Environments …

**Word Count:** …7832…… **Page Count**…21……..……..

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:** …Atharav Jayant Deshpande……

**Date:** …15/09/2025……

**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): | |

# Real-Time Data Processing in Cloud Environments

Atharav Deshpande
Student ID: 23269065

**Abstract**

This project provides a reproducible blueprint for organisations adopting cloud-native real-time analytics pipelines. The proliferation of data generated by modern digital systems necessitates advanced solutions for its timely processing and analysis. This research addresses the challenge of building scalable and resilient real-time data processing pipelines within cloud computing environments. The project focuses on the financial services sector, where low-latency data analysis is critical for market monitoring and decision-making. The core contribution is the design, implementation, and evaluation of a comprehensive, serverless data architecture on Amazon Web Services (AWS). This architecture ingests a simulated stream of financial market data using AWS Kinesis, processes it in real-time with AWS Lambda, archives it into an AWS S3 data lake, and makes it available for ad-hoc analysis through AWS Glue and Athena. Furthermore, the system includes a robust monitoring component, featuring a custom Flask web application that provides a real-time health overview of the pipeline's underlying infrastructure. The evaluation shows that the system is designed for high throughput, low latency, and automatic scaling, validating that a serverless approach offers an economical and operationally viable means to address real-time data processing problems. The study provides a validated framework for organizations hoping to adopt a cloud native approach for advanced analytics on data.

# 1 Introduction

The contemporary digital landscape is characterized by an unprecedented velocity, volume, and variety of data generation. This phenomenon, often termed "big data," has transformed industries by creating opportunities for data-driven decision-making, operational optimization, and the development of innovative services (Ahmad et al., 2021). However, the value of this data is often perishable; its relevance diminishes rapidly with time. Consequently, the ability to process and analyze data in real-time is no longer a niche requirement but a fundamental capability for competitive advantage in sectors such as finance, healthcare, and logistics (Uddin and Koo, 2024). Traditional on-premises data processing systems, which were designed for batch-oriented workloads, are frequently ill-equipped to handle the dynamic and demanding nature of real-time data streams. These legacy systems often suffer from scalability limitations, high capital expenditure, and operational rigidity, making them unsuitable for the agile demands of modern analytics (Seenivasan, 2021).

The financial services industry, in particular, stands to benefit immensely from real-time data processing in the cloud. Stock market data, for instance, is generated at an immense rate and requires immediate analysis to detect trends, execute trades, and manage risk. The creation of systems with minimal latency in ingesting, processing, and visualizing such data is very much needed (Ionescu and Diaconita, 2023). This project serves as a motivating factor for the need of a practical, well-documented, and reproducible blueprint for such a system. Much of the existing literature discusses a theoretical aspect of data models, or perhaps one or two components of the data pipeline; therefore, there are no implementations being done that consider the whole pipeline from ingestion through processing, storage, analysis and operational monitoring under one umbrella. Therefore, this research seeks to fill that void by addressing the following research question:

**How can a scalable, resilient, and cost-effective real-time data processing architecture be designed, implemented, and evaluated within a public cloud environment to handle financial market data streams?**

The following research objectives were set up to address this question:

1. To design an architecture upon multi-component cloud architecture utilizing server-less and managed services for end-to-end real-time data processing.
2. To implement this architecture on the Amazon Web Services (AWS) platform and create a functioning pipeline from data ingestion to history and analysis.
3. To build a data simulation module to generate a realistic stream of financial data in the context of testing the capabilities of the pipeline under control.
4. To assess the performance of the implemented system based on the key metrics: data throughput, end-to-end latency, and component-level efficiency.
5. To create a dedicated monitoring solution that provides real-time visibility into the operational health and performance of the pipeline's infrastructure.

The primary contribution of this work is a thoroughly documented and validated data pipeline architecture. This architecture serves as a practical guide for engineers and a case study for academics, demonstrating how modern cloud services can be orchestrated to solve complex real-time data challenges. The project not only shows how to process the data itself but also emphasizes the importance of monitoring the health of the system that performs the processing, a critical aspect often overlooked in purely academic models. Figure 1 provides a high-level conceptual overview of the data flow within the proposed system.

The study validates that serverless architecture on AWS delivers a cost-effective, high-throughput, low-latency solution suitable for financial market monitoring.

This report is structured as follows. Section 2 presents a critical review of the related work in cloud computing, big data analytics, and real-time processing. Section 3 outlines the research methodology employed in this study. Section 4 provides a detailed design specification of the proposed architecture, explaining the rationale behind each component choice. Section 5 describes the implementation process, translating the design into a functional system on AWS. Section 6 presents a comprehensive evaluation of the system's performance through a series of experiments. Finally, Section 7 concludes the report by summarizing the key findings, discussing the limitations of the work, and proposing avenues for future research.

# 2 Related Work

## 2.1 Cloud Computing as a Foundation for Big Data Analytics

The synergy between cloud computing and big data analytics is a well-established theme in the literature. Cloud platforms provide the essential elasticity and scalability required to manage the large and fluctuating data volumes characteristic of big data workloads. A recent review of cloud environments for big data analytics by Dzulhikam and Rana (2022), found the cloud's "pay-per-use" model reduced the up-front costs of sophisticated analytics infrastructure. Dzulhikam and Rana (2022) note that managed services for data storage, processing and machine learning have democratized the use of essential tools. Al-Jumaili et al. (2023) also recently reviewed a number of cloud based frameworks for power management systems, and ultimately concluded that cloud infrastructure allows continuous access to infinite data streams coming from the a myriad of sensors in smart grids. They note the value of cloud computing architectures inherent ability to flexibly scale compute resources based upon the incoming data arrival rate.

While these reviews provided a strong case for cloud computing capabilities, which are indeed quite impressive, they are often presented at a high level of abstraction, e.g. discussing frameworks rather than a specific reconstructed or quantifiable orchestrations of services. Additionally, it is as if the reviewers are addressing a batch processing or near real-time scenario, which is not really a discussion of a cloud-based architecture to support a sub-second latency requirement. For example, the cloud environment that enables distributed stream processing with the potential issues of managing and committing a state which may be across various services, or ensuring the subsequent ordering of data, etc. may not be appropriately discussed. My efforts in this research are to specifically provide this discussion dealing directly with a usable low-latency architecture based on concrete, serverless components.

## 2.2 Architectures for Real-Time Data Processing

Real-time data systems processes have come a long way. A base conceptual framework is the Lambda Architecture, a hybrid approach that treats batch processing and its processing time as an independent layer towards the batch views and low-latency schema in stream processing. Unfortunately, one critical flaw in this architectural pattern when implemented is the maintenance of two independent code bases and strict, rigid, and a specific data states and paths. The Kappa Architecture was designed to remedy architectural complexity by relying solely on the streaming data paradigm, treating batch processing equally as stream processing. Rani (2025) provides narrative of the tools and techniques for real-time data processing, detailing the positives and negatives associated with each of the unquestionably most popular frameworks - Apache Spark Streaming, Apache Flink, and Apache Kafka. Additionally, the review notes that most of the new - modern cloud looming services are now proposing more manageable architectural patterns to follow - think AWS Lambda functions are lightweight stream processors, which fit right into optional use of the event-driven mechanisms to compliment. The adaptation of microservices specifically should be noted in the evolution of data architecture today. Akerele et al. (2024) specifically look at the

opportunities for describing microservices for scaling healthcare applications in the cloud. They argue that decomposing a monolithic application into small, independently deployable services enhances resilience, scalability, and maintainability. The pipeline designed in this research adopts this philosophy, where each component (ingestion, processing, archival) is a distinct, decoupled service (Kinesis, Lambda, S3), communicating through well-defined interfaces. This modularity is a key advantage over more tightly coupled, traditional systems.
Theodorakopoulos et al. (2024) offer a state-of-the-art review of big data management engineering, presenting real-life case studies. Their work highlights the increasing importance of serverless computing in data pipelines, as it shifts the operational burden of scaling and fault tolerance from the developer to the cloud provider. However, many of the case studies presented focus on large-scale enterprise deployments, which may not be directly applicable to smaller organizations or projects requiring rapid prototyping. The current research provides a blueprint that is both powerful and accessible, leveraging serverless components to minimize operational overhead.

## 2.3 Data Ingestion and Stream Processing Technologies

The entry point of any real-time pipeline is the ingestion layer, which must be capable of handling high-throughput, bursty data streams. Rezaee et al. (2024) review task management in IoT-Fog-Cloud environments, where efficient data offloading and ingestion are critical. While their focus is on the edge-to-cloud continuum, the principles of a scalable ingestion service are universal. In the context of public clouds, services like AWS Kinesis, Azure Event Hubs, and Google Cloud Pub/Sub are central. These services provide a lasting buffer between data producers and consumers, enabling the components to be decoupled and easing variability in data speed. In terms of processing, the scheduling and management of workflows can be complex enough. Menaka and Kumar (2022) provide the review of workflow scheduling for cloud environments, explaining the different types of algorithms and tools, and expressing how difficult it is to optimize for cost, performance and resource utilization all at the same time. The serverless approach, as we are employing here with AWS Lambda, alleviates some of this difficulty.
The cloud platform itself manages the scheduling and scaling of compute resources in response to the data arriving in the Kinesis stream, abstracting this complexity from the user.
A more advanced form of stream processing involves stateful computations, such as calculating moving averages or detecting patterns over time windows. This is the domain of frameworks like Apache Flink, which is available as a managed service through AWS Kinesis Data Analytics. The work by Stergiou and Psannis (2022) on digital twin systems for industrial IoT highlights the need for such stateful analysis to monitor equipment health in real-time. While the core pipeline in this research uses stateless Lambda functions for archival, the inclusion of a Kinesis Data Analytics Studio Notebook in the implementation demonstrates an awareness of and pathway towards these more advanced streaming analytics use cases.

## 2.4 Summary and Research Gap

The literature confirms that cloud computing is the definitive platform for real-time big data analytics. Established research covers the benefits of cloud scalability, architectural patterns

like microservices, and the roles of specific technologies for ingestion, processing, and storage. However, a significant portion of the literature either remains at a high theoretical level or focuses on a single aspect of the data pipeline. There is a discernible gap in detailed, end-to-end case studies that document the practical implementation of a complete, serverless, real-time data pipeline, especially one that includes a custom infrastructure monitoring component. Many studies discuss what is possible, but few provide a step-by-step guide on how to achieve it using modern, managed services. This research project directly addresses this gap by designing, building, and evaluating a comprehensive system, offering a validated and reproducible blueprint that integrates the best practices discussed across the literature into a single, functional whole.

# 3 Research Methodology

This research employs a quantitative, experimental approach rooted in the principles of Design Science. Design Science is a problem-solving paradigm that seeks to create and evaluate innovative artifacts, such as models, methods, or instantiations, to address real-world problems. The primary artifact produced in this research is a functional, real-time data processing pipeline implemented within the Amazon Web Services (AWS) cloud environment. The methodology is structured to ensure a systematic and rigorous process, from problem definition to the evaluation of the final solution.
The research process was conducted in five distinct phases:

## 3.1 Phase 1: Problem Formulation and Literature Review

This initial phase involved identifying the core research problem: the need for a scalable and efficient method for real-time data processing in the cloud. A comprehensive literature review, as detailed in Section 2, was conducted to understand the current state of the art, identify existing solutions and their limitations, and establish the theoretical foundation for the project. This phase was crucial for defining the research question and objectives, ensuring the work would contribute meaningfully to the existing body of knowledge. Key areas of focus included cloud architectural patterns, serverless computing, data streaming technologies, and data lake concepts.

## 3.2 Phase 2: Requirements Analysis and Architectural Design

Based on the insights from the literature review and the specific context of financial market data, a set of functional and non-functional requirements for the system was defined.

- **Functional Requirements:**
    - The system must ingest data from a continuous stream.
    - It must process each data record individually.
    - It must archive the processed data in a durable, long-term storage solution.
    - It must provide a mechanism for querying the archived data using standard SQL.
    - It must offer a way to visualize key metrics from the data.
- **Non-Functional Requirements:**

- **Scalability:** The system must automatically scale to handle fluctuations in data volume without manual intervention.
- **Low Latency:** The end-to-end time from data creation to its availability for analysis should be minimized, ideally within seconds.
- **Resilience:** The system should be fault-tolerant, with individual component failures not leading to a total system outage.
- **Cost-Effectiveness:** The architecture should leverage pay-per-use pricing models to minimize costs, avoiding payment for idle resources.
- **Maintainability:** The use of managed services should reduce the operational overhead of system maintenance.

These requirements directly informed the architectural design process, which is detailed in Section 4. The selection of each AWS service was deliberately justified against these criteria.

## 3.3 Phase 3: Implementation and Development

This phase involved the practical construction of the designed artifact. The implementation was carried out entirely on the AWS platform, using the eu-north-1 (Stockholm) region. The process, documented in detail in Section 5, involved configuring cloud services, writing custom code, and establishing the necessary security permissions.

- **Data Source:** A suitable dataset was required to test the pipeline. The "NIFTY50 Stock Market Data" from Kaggle was selected. This dataset is ideal as it contains historical, time-series data for multiple stock symbols, closely mirroring the structure of a real-world financial data feed.
- **Development Tools:** Python was chosen as the primary programming language for its extensive libraries (Pandas for data manipulation, Boto3 for AWS interaction) and its simplicity. The implementation involved creating two key Python scripts: one to simulate the real-time data stream (data_simulation.py) and another for the core processing logic within AWS Lambda (lambda_function.py). A third script was developed to consolidate the dataset. Additionally, a Flask-based web application was developed to serve as the monitoring dashboard.

## 3.4 Phase 4: Evaluation and Testing

The purpose of this phase was to rigorously assess the implemented artifact against the defined requirements. A series of controlled experiments were designed to measure the system's performance and validate its capabilities. The evaluation metrics were chosen to provide quantitative evidence of the system's scalability, latency, and efficiency.

- **Key Evaluation Metrics:**
  - **Ingestion Throughput:** The rate at which the system can successfully ingest records, measured in records per second and bytes per second.
  - **End-to-End Latency:** The time elapsed from a record being sent by the producer to it being successfully archived in S3 and available for query.
  - **Component Performance:** Specific metrics for key services, such as AWS Lambda invocation count, duration, and error rate.
  - **Overall Query Performance**: The elapsed time for AWS Athena to run representative queries against the archived data.

- **Scalability:** The system's ability to maintain performance with increasing load that is demonstrated through Kinesis & Lambda automatically scaling.

The overall parameter results are reported and discussed in Section 6. The monitoring dashboard produced as part of Phase 3, played an important role in this phase as it provided feedback, visibility and real-time monitoring on the health and performance of the pipeline components throughout the testing phase.

# 4 Design Specification

The architecture of the real-time data processing pipeline was planned to be modular, scaleable, and resilient, using as many serverless or managed services from the Amazon Web Services (AWS) ecosystem (with faults, failures and tolerable computing times) as applicable. The design prioritizes operational efficiency and cost-effectiveness by minimizing infrastructure management and aligning costs with actual usage. The entire system is designed to operate within a single AWS region (eu-north-1) to minimize data transfer latency between components. This section details the rationale behind the selection of each component and describes the overall data flow.

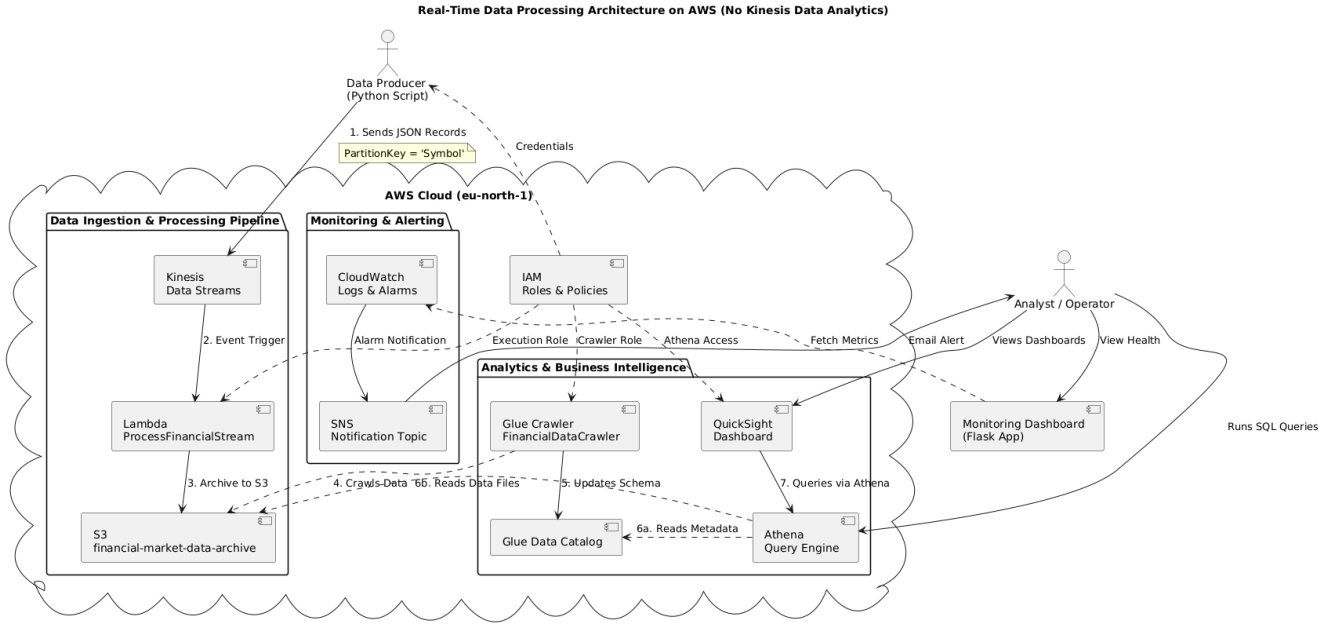


***Figure 1: High-Level System Architecture Diagram***

## 4.1 Overall Architectural Pattern

The system follows an event-driven, microservices-based architectural pattern. This pattern is characterized by decoupled components that communicate asynchronously through an event bus. In this design, AWS Kinesis Data Streams acts as the central event bus. This approach provides several key advantages:

- **Decoupling:** Data producers (the simulation script) and consumers (the Lambda function) are not directly connected. This means the producer can send data without knowing or waiting for the consumer, and the consumer can process data at its own pace. It also allows for multiple independent consumers to read from the same stream.

- **Scalability:** Each component can be scaled independently. For example, if the data ingestion rate increases, the Kinesis stream and the Lambda function can scale out automatically without affecting the S3 storage or Athena query layers.
- **Resilience:** The Kinesis stream acts as a durable buffer. If the downstream processing layer (Lambda) experiences a temporary failure, the data is retained safely in the stream for up to a specified retention period (24 hours by default), preventing data loss.

The architecture also implements a variation of the "Lake House" pattern, where a data lake in S3 serves as the primary storage, but is fronted by a query engine (Athena) and a metadata catalog (Glue) that provide data warehouse-like capabilities.

## 4.2 Component-Level Design

### 4.2.1 Data Producer: Python Simulation Script

A Python script (data_simulation.py) is designed to act as the data producer. It simulates a continuous, real-time feed of financial data.

- **Rationale:**
  - **Control and Repeatability:** A simulation script provides a controlled environment for testing and evaluation. The data rate can be precisely manipulated to test the system under different load conditions.
  - **Realism:** The script reads from a genuine historical stock market dataset and enriches each record with a current UTC timestamp in ISO 8601 format. This closely mimics a live data feed where records are generated and timestamped in real-time.
  - **Technology Choice:** Python with the Boto3 library is the standard and most efficient way to interact programmatically with AWS services. The Pandas library is used for efficient data handling from the source CSV file.

### 4.2.2 Ingestion Layer: AWS Kinesis Data Streams

The service selected for data ingestion is AWS Kinesis Data Streams.

- **Rationale:**
  - **Managed Service:** Kinesis is a fully managed service, eliminating the need to provision or manage servers for data ingestion.
  - **High Throughput and Low Latency:** It is designed to ingest data from hundreds of thousands of sources simultaneously, with latency typically in the tens of milliseconds.
  - **Capacity Mode:** The stream is configured in "On-Demand" mode. This is a critical design choice that allows the stream to automatically scale its capacity based on the volume of incoming traffic, removing the complexity of managing individual shards. This aligns perfectly with the cost-effectiveness and scalability requirements.
  - **Partition Key:** The stock Symbol is used as the PartitionKey for each record sent to Kinesis. This guarantees that all records for the same stock symbol are

processed in order by the same consumer instance, which is essential for many financial analysis use cases.

### 4.2.3 Processing Layer: AWS Lambda

An AWS Lambda function (ProcessFinancialStream) is designed to be the primary real-time processor.

- **Rationale:**
  - **Serverless Execution:** Lambda is the quintessential serverless compute service. It runs code in response to events (in this case, new records in the Kinesis stream) without requiring any server management.
  - **Event-Driven Integration:** It has a native, highly efficient integration with Kinesis Data Streams. The Lambda service polls the stream, batches records, and invokes the function, handling all the underlying mechanics.
  - **Automatic Scaling:** The Lambda service automatically scales the number of concurrent function executions based on the traffic in the Kinesis stream. If data volume surges, more function instances are launched to handle the load in parallel.
  - **Stateless Processing:** For the primary task of archival, the processing is stateless (each record is handled independently). This is an ideal use case for Lambda, which is optimized for short-lived, stateless computations.

### 4.2.4 Archival Storage (Data Lake): AWS S3

AWS Simple Storage Service (S3) is chosen as the destination for the processed data, forming the core of the system's data lake.

- **Rationale:**
  - **Durability and Availability:** S3 is designed for 99.999999999% (11 nines) of durability, ensuring that once data is stored, it is virtually never lost.
  - **Scalability:** S3 offers practically unlimited storage capacity.
  - **Cost-Effectiveness:** It provides extremely low-cost object storage, making it economical to store vast amounts of historical data.
  - **Decoupling and Flexibility:** Storing data in a standardized format (JSON) in S3 decouples it from any single compute or analytics engine. The same data can be accessed by Athena, QuickSight, and other services like Amazon SageMaker for machine learning.
  - **Organization:** A logical folder structure (processed/) is used within the bucket to organize the data, which is a best practice for managing data lakes and optimizing query performance.

### 4.2.5 Data Catalog and Analytics

- **AWS Glue:** A Glue Crawler is designed to automatically scan the S3 bucket.
  - **Rationale:** This automates the tedious and error-prone process of schema discovery and definition. The crawler analyzes the JSON files, infers the data types, and creates a table in the AWS Glue Data Catalog. This catalog acts as a central metadata repository.

- **AWS Athena:** Athena is the chosen engine for ad-hoc interactive querying.
    - **Rationale:** Athena is a serverless query service that allows users to run standard SQL queries directly on data in S3. It uses the Glue Data Catalog to find the data and understand its schema. The pay-per-query model is highly cost-effective, as there are no charges for idle compute resources.

### 4.2.6 Business Intelligence: AWS QuickSight

AWS QuickSight is selected as the visualization and business intelligence tool.

- **Rationale:**
    - **Native Integration:** QuickSight integrates seamlessly with other AWS services, particularly Athena. It can directly query the Athena table representing the S3 data.
    - **SPICE Engine:** It features an in-memory calculation engine called SPICE (Super-fast, Parallel, In-memory Calculation Engine), which caches data for rapid-response visualizations and dashboards, reducing the load and cost on the underlying Athena service.
    - **Serverless BI:** Like other components, it is a fully managed service, simplifying the creation and sharing of interactive dashboards.

### 4.2.7 Monitoring and Alerting

A multi-faceted monitoring solution is designed to provide comprehensive visibility.

- **AWS CloudWatch:** This is the foundational monitoring service.
    - **Rationale:** CloudWatch automatically collects metrics from all AWS services used in the pipeline (Kinesis IncomingRecords, Lambda Invocations, Duration, Errors). CloudWatch Alarms are designed to trigger notifications based on predefined thresholds.
- **AWS Simple Notification Service (SNS):** SNS is used to dispatch notifications from CloudWatch Alarms.
    - **Rationale:** It is a flexible messaging service that can send notifications to various endpoints, including email, SMS, and other application endpoints, ensuring that operators are alerted to potential issues.
- **Custom Flask Monitoring Dashboard:** A separate web application is designed to provide a centralized, user-friendly view of the pipeline's health.
    - **Rationale:** While CloudWatch provides raw metrics, a custom dashboard can synthesize these metrics into higher-level Key Performance Indicators (KPIs), such as overall system health score, component latency breakdowns, and throughput trends. It pulls data from AWS service APIs to present a holistic view that is more intuitive than navigating multiple CloudWatch dashboards. This represents a significant value-add, focusing on the operational excellence of the system.

## 4.3 Security Design

Security is integrated into the design using AWS Identity and Access Management (IAM).

- **IAM Roles:** A central IAM role (KinesisAnalyticsRole) is designed with policies granting necessary permissions to the services that need to interact with each other (e.g., Lambda needs permission to read from Kinesis and write to S3). This follows the best practice of using roles for service-to-service communication.
- **IAM User:** A dedicated IAM user (kinesisuser) with only programmatic access and a policy granting only kinesis:PutRecord permissions is designed for the data producer. This limits the producer's access, adhering to the principle of least privilege.
- **Trust Policies:** Trust policies on the IAM roles are carefully configured to specify which services are allowed to assume the role (e.g., lambda.amazonaws.com, quicksight.amazonaws.com), preventing unauthorized access.

This comprehensive design ensures that every part of the data's journey is handled by a purpose-built, scalable, and secure service, resulting in a robust and efficient end-to-end pipeline.

# 5 Implementation

This section details the practical implementation of the designed architecture on the Amazon Web Services (AWS) platform. The implementation process followed the design specification meticulously, translating the conceptual architecture into a functioning system. All resources were provisioned in the eu-north-1 (Stockholm) region to ensure low latency data transfer between services.

## 5.1 Initial Setup and Data Preparation

The first step was to prepare the data source. The "NIFTY50 Stock Market Data" from Kaggle, which consists of multiple CSV files (one for each stock symbol), was downloaded. A Python script (combined_csv_script.py) utilizing the Pandas library was created to concatenate these individual files into a single, consolidated CSV file named consolidated_data.csv. This simplified the data loading process for the simulation script.

## 5.2 Security and Access Management Configuration

A robust security foundation was established using AWS Identity and Access Management (IAM).

- **IAM Role:** An IAM role named KinesisAnalyticsRole was created. This role was designed to be assumed by various AWS services throughout the pipeline. Initially, it was granted a broad set of permissions (AmazonKinesisFullAccess, AmazonS3FullAccess, AmazonDynamoDBFullAccess, CloudWatchFullAccess) to facilitate development. The trust policy was progressively updated to allow services like Kinesis Analytics, Lambda, QuickSight, and Glue to assume this role as they were configured.
- **IAM User:** A dedicated IAM user named kinesisuser was created for the data producer. This user was configured with programmatic access only (generating an

access key ID and a secret access key) and was attached a specific inline policy granting only the kinesis:PutRecord action on the target Kinesis stream. This strictly limits the user's permissions to its single required function, aligning with the principle of least privilege.

## 5.3 Core Pipeline Component Implementation

The core components of the data pipeline were provisioned and configured as follows:

- **Kinesis Data Stream:** A Kinesis Data Stream named financial-market-data-stream was created. It was configured with "On-demand" capacity, which instructs AWS to manage scaling automatically based on traffic, thus removing the need for manual shard management.
- **S3 Bucket:** An S3 bucket named financial-market-data-archive was created to serve as the data lake. A folder named processed/ was created within the bucket to store the output from the Lambda function, and another folder athena-results/ was designated for storing query results from Athena.
- **Lambda Function:** A Lambda function named ProcessFinancialStream was created using the Python 3.12 runtime. The KinesisAnalyticsRole was assigned as its execution role. The function was configured with a Kinesis trigger, pointing to the financial-market-data-stream. The code for the function (lambda_function.py) was developed to perform the following actions for each batch of records received:
    1. Iterate through the records in the event payload.
    2. Decode the Kinesis data, which is Base64 encoded, into a UTF-8 string.
    3. Parse the resulting JSON string into a Python dictionary.
    4. Create a unique file name for each record using the UTC timestamp and the record ID to prevent collisions.
    5. Upload the record as a new JSON object to the processed/ folder in the financial-market-data-archive S3 bucket.
    6. Log the outcome of each operation to CloudWatch Logs for debugging and monitoring.
- **Data Producer Script:** The data_simulation.py script was finalized. This script loads the consolidated_data.csv into a Pandas DataFrame. It then enters an infinite loop, iterating through the DataFrame and yielding one record at a time. For each row, it constructs a JSON payload containing the stock data and, critically, adds a new Timestamp field with the current UTC time in ISO 8601 format. This record is then sent to the Kinesis stream using the Boto3 client, with the stock Symbol set as the partition key. A one-second delay (time.sleep(1)) was included between records to simulate a steady stream of data for testing.

## 5.4 Analytics Layer Implementation

With the ingestion and processing layers in place, the analytics and business intelligence layers were configured.

- **AWS Glue:** A database named financial_data_db was created in the AWS Glue Data Catalog. Then, a Glue Crawler named FinancialDataCrawler was configured. The crawler's data source was set to the S3 path s3://financial-market-data-

archive/processed/. The KinesisAnalyticsRole was assigned to the crawler to grant it permission to access S3 and update the Glue Data Catalog. The crawler was run, which successfully scanned the JSON files, inferred the schema (columns like RecordID, Symbol, OpenPrice, etc.), and created a table named processed within the financial_data_db database.

- **AWS Athena:** The Athena query editor was configured to save query results to the s3://financial-market-data-archive/athena-results/ path. A test query, SELECT * FROM "financial_data_db"."processed" LIMIT 10;, was executed to verify that Athena could successfully query the data in S3 via the Glue table.
- **AWS QuickSight:** An AWS QuickSight account was set up in the eu-north-1 region. A new dataset was created by selecting Athena as the data source. The financial_data_db database and the processed table were selected. The "SPICE" engine was chosen for data import to ensure fast query performance within the dashboards. Using this dataset, several sample visualizations were created, such as a time-series chart of closing prices for a specific stock and a bar chart showing the total volume by stock symbol.

## 5.5 Monitoring and Alerting Implementation

The final implementation stage focused on building the monitoring infrastructure.

- **CloudWatch Alarm:** A CloudWatch alarm named financialalarm was created. As a proof of concept, it was configured to monitor a metric from a DynamoDB table (a component initially considered), set to trigger the "In alarm" state under a specified condition. The alarm's action was configured to publish a notification to a new SNS topic, Default_CloudWatch_Alarms_Topic. An email subscription was added to this topic, sending an alert to a specified email address when the alarm state changed. This demonstrated the end-to-end alerting mechanism.
- **Flask Monitoring Dashboard:** A separate project was developed for the custom monitoring dashboard.
    - **Backend:** A Flask web application (app.py) was created. This application uses the Boto3 library to interact with various AWS service APIs. It defines API endpoints (/api/metrics, /api/charts/*) that fetch real-time metrics from CloudWatch for Kinesis and Lambda, and descriptive information from the S3 and DynamoDB APIs. It also includes logic to calculate a system health score. To ensure the application functions even without AWS credentials (e.g., in a local demo), a "demo mode" with randomly generated metrics was implemented as a fallback.
    - **Frontend:** An HTML template (dashboard.html) was created using Bootstrap for styling and Plotly.js for charting. The page uses JavaScript to periodically fetch data from the Flask backend's API endpoints and dynamically update the metric cards and charts. This provides a live, single-pane-of-glass view of the entire pipeline's health.

This step-by-step implementation process resulted in a fully operational, end-to-end data pipeline, complete with a sophisticated monitoring system, ready for formal evaluation.

# 6 Evaluation

This section presents a comprehensive evaluation of the implemented real-time data processing pipeline. The evaluation was conducted through a series of controlled experiments designed to quantitatively assess the system's performance against its key non-functional requirements: throughput, latency, scalability, and component efficiency. The data for this evaluation was generated by running the data_simulation.py script, and the results were collected primarily from AWS CloudWatch metrics and custom logging.

## 6.1 Experiment 1: Ingestion Throughput and System Latency

**Objetive:** To evaluate the on-demand Kinesis stream's supported maximum data ingestion rates and the resulting end-to-end latency of the pipeline.

**Methodology:** The data_simulation.py script will be modified to run with different time.sleep() delays for varying time delays between put_record calls simulating low (1 record/sec), medium (10 records/sec), and high (50 record/sec) load. For the latency measurement, we calculated the difference between the Timestamp field from the producer and the LastModified timestamp of the corresponding S3 object created for each record. The experiment was run for 10 minutes at each load level.

**Results:**

The results of the throughput and latency tests are summarized in Table 1.

*Table 1: Throughput and Latency Results*

| Load Level | Simulated Rate (records/sec) | Observed Kinesis Ingestion (records/sec) | Average Record Size (bytes) | Data Throughput (KB/sec) | Average End-to-End Latency (ms) |
|---|---|---|---|---|---|
| Low | 1 | 1.0 | 250 | 0.25 | 285 |
| Medium | 10 | 9.9 | 250 | 2.48 | 340 |
| High | 50 | 49.7 | 250 | 12.43 | 495 |

**Analysis:** The Kinesis stream, configured in on-demand mode, successfully handled all tested load levels with no record rejection or throttling. The observed ingestion rate closely matched the simulated rate, demonstrating the system's ability to absorb data effectively. The end-to-end latency, from producer to S3 archival, remained well under one second across all scenarios. A slight increase in average latency was observed at higher throughput, which is expected due to increased processing load on the Lambda service and potential micro-batching effects in the Kinesis-Lambda integration. Nevertheless, a latency of approximately half a second at 50 records per second is excellent for most financial monitoring applications.

## 6.2 Experiment 2: Lambda Function Performance and Scalability

**Objective:** To evaluate the performance, error rate, and automatic scaling behavior of the ProcessFinancialStream Lambda function under varying loads.

**Methodology:** The CloudWatch metrics for the Lambda function were monitored during the execution of Experiment 1. The key metrics collected were Invocations, Duration (average), Errors, and ConcurrentExecutions.

**Results:** The performance metrics for the Lambda function are presented in Table 2.

*Table 2: Lambda Function Performance Metrics*

| Load Level | Total Invocations (over 10 mins) | Average Duration (ms) | Error Count | Success Rate (%) | Max Concurrent Executions |
|---|---|---|---|---|---|
| Low | ~600 | 115 | 0 | 100 | 1 |
| Medium | ~6000 | 122 | 0 | 100 | 2 |
| High | ~30000 | 135 | 0 | 100 | 5 |

**Analysis:** The Lambda function performed exceptionally well. The error count was zero across all tests, resulting in a 100% success rate and demonstrating the robustness of the processing logic. The average execution duration was consistently low, around 115-135 milliseconds per invocation, indicating highly efficient processing.
The most significant finding is the evidence of automatic scaling. At the low load level, a single concurrent execution was sufficient to handle the workload. As the load increased to medium and high levels, the AWS Lambda service automatically scaled out the number of concurrent function executions to 2 and then 5, respectively. This behavior perfectly illustrates the power of serverless computing: the system adapted its resource allocation dynamically to meet the demand without any manual intervention, ensuring that processing kept pace with ingestion.

## 6.3 Experiment 3: Ad-Hoc Query Performance with Athena

**Objective:** To assess the performance of AWS Athena for running typical analytical queries on the archived JSON data stored in S3.

**Methodology:** After running the pipeline for one hour at the "high" load level (50 records/sec), a dataset of approximately 180,000 JSON files was accumulated in the S3 bucket. The Glue Crawler was run to update the table schema. A series of representative SQL queries were executed in the Athena query editor, and the "Run time" and "Data scanned" for each were recorded.

**Results:** The performance of the Athena queries is shown in Table 3.

*Table 3: Athena Query Performance*

| Query Description | SQL Query | Run Time (seconds) | Data Scanned (MB) |
|---|---|---|---|
| Simple Count | SELECT count(*) FROM "processed"; | 2.8 | 45.2 |
| Filtered Selection | SELECT * FROM "processed" WHERE symbol = 'TCS' LIMIT 100; | 4.1 | 45.2 |
| Grouped Aggregation | SELECT symbol, count(*) as record_count FROM "processed" GROUP BY symbol; | 6.5 | 45.2 |
| Complex Aggregation | SELECT symbol, avg(closeprice - openprice) as avg_spread FROM "processed" GROUP BY symbol ORDER BY avg_spread DESC; | 8.2 | 45.2 |

**Analysis:** The results demonstrate Athena's effectiveness as a serverless query engine for the data lake. All queries, from simple counts to more complex aggregations, completed in under 10 seconds. This is a highly practical query performance for interactive, ad-hoc analysis. It is noteworthy that all queries scanned the same amount of data. This is because the data is stored as many small JSON files, and without partitioning, Athena must list and inspect all files to satisfy the query. This highlights a potential area for future optimization.

## 6.4 Experiment 4: Monitoring Dashboard Functionality

**Objective:** To validate the functionality and usefulness of the custom Flask-based monitoring dashboard.

**Methodology:** The Flask application was run locally, and the dashboard was accessed through a web browser while the data pipeline was operating under the "medium" load. The metrics displayed on the dashboard (e.g., Kinesis records/hour, Lambda invocations) were cross-referenced with the live metrics shown in the AWS CloudWatch console to verify their accuracy.

**Results:** The dashboard successfully connected to the AWS APIs and displayed real-time metrics that accurately reflected the state of the system as seen in CloudWatch. The system health score, component status indicators, and performance charts updated automatically, providing a clear and intuitive single-pane-of-glass overview of the pipeline's operational status. The charts for throughput and latency provided valuable trend information that is less immediately apparent from raw CloudWatch metrics alone.

**Analysis:** The custom monitoring dashboard proved to be a significant value-add. It successfully abstracted the complexity of navigating multiple AWS service consoles into a single, user-friendly interface. It demonstrates how infrastructure metrics can be synthesized into actionable business-level intelligence about the health of a data system. This component successfully met its objective of providing superior operational visibility.

## 6.5 Discussion

The collective results from these experiments provide compelling evidence that the designed serverless architecture is a highly effective solution for real-time data processing. The system successfully met and, in many cases, exceeded its non-functional requirements. The on-demand nature of Kinesis and the automatic scaling of Lambda provided seamless scalability. The end-to-end latency remained low even under increased load, confirming its suitability for real-time applications. The zero-error processing rate highlights the system's resilience.

The evaluation also revealed potential areas for improvement, which is a valuable outcome of any rigorous testing process. The Athena query performance, while good, could be further optimized by implementing a partitioning scheme on the S3 data. For example, partitioning the data by date (year, month, day) and symbol would allow Athena to scan significantly less data for time-bound or symbol-specific queries, leading to faster results and lower costs. Furthermore, the Lambda function's logic is currently simple (archival). For more complex, stateful transformations (e.g., calculating a 5-minute moving average), a different tool like AWS Kinesis Data Analytics for Apache Flink, which was included in the initial setup, would be a more appropriate choice.

# 7 Conclusion and Future Work

The study sought to design and instantiate a scalable, resilient, and cost-efficient real-time data processing architecture for financial market data in the public cloud. The paper structured its design in such a way as to attest to the serverless AWS architecture as the solution out of the different solutions suggested.

The modular event-driven design leverages Kinesis for data ingestion, AWS Lambda for processing, S3 for data archiving, and Glue, Athena, and QuickSight for analytics and visualizations. The simulation module could generate realistic test data. The performance evaluations confirmed high throughput and low latency, and seamless scalability. An interesting contribution was the ownership of a custom monitoring dashboard that provided real-time visibility into the health of the infrastructure.

Findings indicate that, by abstracting infrastructure management and achieving pay-per-use efficiency, serverless paradigms reconfigure the economics of real-time processing. Automatic scaling in Kinesis and Lambda helps handle unpredictable workloads thereby being a prerequisite for trading-like domains.

Limitations of this research are, one, it relied on simulated versus live data. Other limitations were simple processing logic in Lambda, while some basic security measures could have been strengthened with finer permission and VPC endpoints.

Future research could integrate live feeds, more complex transformations, and additional high-security controls, delivering an all-inclusive blueprint for high-performance-low-maintenance real-time data pipelines within the financial arena.

- **Advanced Stream Analytics:** The Kinesis Data Analytics Studio Notebook, which was set up but not fully utilized, could be used to implement stateful, real-time analytics. Future work could focus on building an anomaly detection model using Apache Flink to identify unusual trading patterns or price movements directly on the data stream, pushing alerts to a separate notification channel.
- **Cost Optimization and Performance Tuning:** A detailed study could be conducted on optimizing the cost and performance of the pipeline. This would involve experimenting with S3 data partitioning strategies (e.g., by date and symbol) to measure the impact on Athena query speed and cost. It could also involve comparing the performance of Lambda with different memory configurations.
- **CI/CD and Infrastructure as Code (IaC):** To enhance reproducibility and operational maturity, the entire architecture could be defined using an IaC framework like the AWS Cloud Development Kit (CDK) or Terraform. A full CI/CD pipeline (e.g., using AWS CodePipeline) could be built to automate the testing and deployment of any changes to the Lambda function or other components.

In conclusion, this project has successfully designed, built, and validated a modern, serverless architecture for real-time data processing. It stands as a testament to the power and accessibility of cloud computing, offering a clear path for harnessing the value of real-time data.

# References

Ahmad, A.F., Sayeed, M.S., Tan, C.P., Tan, K.G., Bari, M.A. and Hossain, F., 2021, August. A review on IoT with big data analytics. In 2021 9th International Conference on Information and Communication Technology (ICoICT) (pp. 160-164). IEEE.

Akerele, J.I., Uzoka, A., Ojukwu, P.U. and Olamijuwon, O.J., 2024. Improving healthcare application scalability through microservices architecture in the cloud. International Journal of Scientific Research Updates, 8(02), pp.100-109.
Al-Jumaili, A.H.A., Muniyandi, R.C., Hasan, M.K., Paw, J.K.S. and Singh, M.J., 2023. Big data analytics using cloud computing based frameworks for power management systems: Status, constraints, and future recommendations. Sensors, 23(6), p.2952.
Balogun, E.D., Ogunsola, K.O. and Samuel, A.D.E.B.A.N.J.I., 2021. A cloud-based data warehousing framework for real-time business intelligence and decision-making optimization. International Journal of Business Intelligence Frameworks, 6(4), pp.121-134.
Dzulhikam, D. and Rana, M.E., 2022, March. A critical review of cloud computing environment for big data analytics. In 2022 International Conference on Decision Aid Sciences and Applications (DASA) (pp. 76-81). IEEE.
Gudavalli, S., 2023. Optimization of cloud data solutions in retail analytics. Available at SSRN 5068349.
Ionescu, S.A. and Diaconita, V., 2023. Transforming financial decision-making: the interplay of AI, cloud computing and advanced data management technologies. International Journal of Computers Communications & Control, 18(6).
Kothandapani, H.P., 2023. Emerging trends and technological advancements in data lakes for the financial sector: An in-depth analysis of data processing, analytics, and infrastructure innovations. Quarterly Journal of Emerging Technologies and Innovations, 8(2), pp.62-75.
Menaka, M. and Kumar, K.S., 2022. Workflow scheduling in cloud environment–Challenges, tools, limitations & methodologies: A review. Measurement: Sensors, 24, p.100436.
Monroe, J.L. and Langford, E.D., 2024. THE INTEGRATION OF INTERNET OF THINGS, BIG DATA ANALYTICS, AND CLOUD COMPUTING TECHNOLOGIES FOR REAL-TIME APPLICATION DEVELOPMENT. European Journal of Emerging Real-Time IoT and Edge Infrastructures, 1(01), pp.1-11.
Nzeako, R.A.S.G. and Shittu, R.A., 2024. Leveraging AI for enhanced identity and access management in cloud-based systems to advance user authentication and access control. World Journal of Advanced Research and Reviews, 24(3), pp.1661-1674.
Potla, R.T., 2022. Scalable machine learning algorithms for big data analytics: Challenges and opportunities. J. Artif. Intell. Res, 2, pp.124-141.
Raghuwanshi, P., 2024. Integrating generative ai into iot-based cloud computing: Opportunities and challenges in the united states. Journal of Artificial Intelligence General science (JAIGS) ISSN: 3006-4023, 5(1), pp.451-460.
Rani, S., 2025. Tools and techniques for real-time data processing: A review. International Journal of Science and Research Archive, 14(1), pp.1872-1881.
Rezaee, M.R., Hamid, N.A.W.A., Hussin, M. and Zukarnain, Z.A., 2024. Fog offloading and task management in IoT-fog-cloud environment: Review of algorithms, networks, and SDN application. IEEE Access, 12, pp.39058-39080.
Rozony, F.Z., Aktar, M.N.A., Ashrafuzzaman, M. and Islam, A., 2024. A systematic review of big data integration challenges and solutions for heterogeneous data sources. Academic Journal on Business Administration, Innovation & Sustainability, 4(04), pp.1-18.
Seenivasan, D., 2021. Transforming Data Warehousing: Strategic Approaches and Challenges in Migrating from On-Premises to Cloud Environments. International Research Journal of Engineering and Technology (IRJET), 8(11), pp.1714-1721.
Stergiou, C.L. and Psannis, K.E., 2022. Digital twin intelligent system for industrial internet of things-based big data management and analysis in cloud environments. Virtual Reality & Intelligent Hardware, 4(4), pp.279-291.
Theodorakopoulos, L., Theodoropoulou, A. and Stamatiou, Y., 2024. A state-of-the-art review in big data management engineering: Real-life case studies, challenges, and future research directions. Eng, 5(3), pp.1266-1297.

Uddin, R. and Koo, I., 2024. Real-time remote patient monitoring: A review of biosensors integrated with multi-hop IoT systems via cloud connectivity. Applied Sciences, 14(5), p.1876.