RESEARCH ARTICLE

SAMRIDDHI Volume 14, Issue 4, 2022 Print ISSN: 2229-7111 Online ISSN: 2454-5767

Event-Driven Microservices for Real-Time Revenue Recognition in Cloud-Based Enterprise Applications

Sravan Komar Reddy Pullamma

PMP , USA

Abstract

Live revenue recognition is an urgent demand in the current cloud based enterprise applications as businesses are ready to have real time nancial information and regulatory compliance. The monolithic traditional architectures have been found not to be very compatible with providing scalability, responsiveness, and exibility needed to handle dynamic transaction processing. This paper examines how event-driven microservices can be used to support real-time recognition of the revenue in cloud environments. Through asynchronous messaging, event sourcing and domain-driven design, the proposed architecture can decouple nancial processes into scalable independence services that can process revenue events as they happen. An implementation prototype shows reduced latency, throughput, and systems resilience; this is a positive change over the traditional batch based designs. The results show that event-based microservices adoption does not only improve real-time nancial processing but also auditability, compliance, and operational agility of enterprise applications. This study o ers a practical solution to organizations that want to transform their nancial systems to enable them have real-time revenue transparency in cloud-based systems.

Keywords: Event-Driven Architecture, Microservices, Real-Time Revenue Recognition, Cloud Computing, Enterprise Applications, Financial Technology, Event Sourcing, Asynchronous Messaging

SAMRIDDHI : A Journal of Physical Sciences, Engineering and Technology (2022); DOI: 10.18090/samriddhi.v14i04.37

Introduction

The concept of revenue recognition is a core activity in enterprise nancial management that helps them record their income in the right way and according to the accounting system like the IFRS 15 and the ASC 606. The financial transactions made on traditional monolithic enterprise applications are mostly based on batch processing and this may cause delays, lack of transparency and may not facilitate timely decision making. Organizations are increasingly operating in dynamic business settings that demand real-time revenue data to help them become operationally nimble, compliant with regulations, and undertake strategic nancial planning e orts.

The advent of cloud computing and microservices architecture o ers a revolutionary chance to deal with these issues. Microservices are used to break down monolithic applications into loosely coupled independently deployable services, which can be more scaled, exible, and maintainable. In combination with an event-driven architecture (EDA), such microservices can react to transactional events in real-time, updating revenue records in real-time as business happens. Event-driven microservices is an event-oriented idea that is built on asynchronous communication patterns, event brokers, and domain events that are used to make sure that processes of revenue recognition are timely and consistent across distributed cloud systems.

This study examines the design and deployment of event-based microservices in real time recognition of revenue in cloud based enterprise applications. It looks at the architecture, major elements and integration of strategies that are required to meet correct and low-latency nancial processing. Moreover, the research discusses the issues related to data consistency, ordering of events, event compliance in distributed cloud environments and possible solutions and best practices to adopt by the enterprise.

Through event-driven microservices, businesses would gain better insight into the ow of revenue, expedite the nancial closing process, and increase responsiveness to business events, which would help to make decisions and carry out strategic nancial management more informed.

Corresponding Author: Sravan Komar Reddy Pullamma, a liation, e-mail: psravanreddy@gmail.com

How to cite this article: Reddy ,SK., (2022). Event-Driven Microservices for Real-Time Revenue Recognition in Cloud-Based Enterprise Applications *SAMRIDDHI : A Journal of Physical Sciences, Engineering and Technology,* 14(4), 176-184.

Source of support: Nil

Con ict of interest: None

Literature Review

Event-Driven Architecture (EDA): Foundations and Relevance

Event-Driven Architecture (EDA) has emerged as a pivotal design paradigm, enabling systems to detect, process, and react to real-time events as they occur. Unlike traditional request-response models, EDA decouples services, facilitating asynchronous, scalable, and resilient workflows. This approach is particularly bene cial in dynamic environments where systems must respond promptly to changes, such as nancial transactions in enterprise applications.

Microservices Architecture: Enabling Modularity and Scalability

Microservices architecture decomposes applications into small, independent services that can be developed, deployed, and scaled independently. This modularity enhances exibility and scalability, allowing organizations to adapt swiftly to changing business requirements. In the context of revenue recognition, microservices can encapsulate distinct business processes, such as billing, invoicing, and revenue calculation, enabling more efficient and maintainable systems.

Event-Driven Microservices: Synergizing EDA and Microservices

Integrating EDA with microservices combines the bene ts of both paradigms. Event-driven microservices communicate through asynchronous events, promoting loose coupling and enhancing system resilience. This integration is particularly advantageous for handling complex work ows, such as revenue recognition, where multiple services must collaborate in real-time to process transactions and update nancial records.

Real-Time Revenue Recognition: Challenges and Requirements

Traditional revenue recognition processes often rely on batch processing, leading to delays and potential inaccuracies in nancial reporting. Real-time revenue recognition addresses these challenges by updating nancial records immediately as transactions occur. This approach requires systems to process events promptly, maintain data consistency across services, and comply with nancial regulations and standards.

Cloud-Based Enterprise Applications: Infrastructure and Bene ts

Cloud computing provides the infrastructure necessary for scalable and exible enterprise applications. Cloud platforms o er on-demand resources, enabling organizations to scale their applications based on demand. Additionally, cloud services often include tools for monitoring, security, and compliance, which are essential for managing complex systems like those used for real-time revenue recognition.

Integration of EDA and Microservices in Cloud Environments

The integration of EDA and microservices within cloud environments facilitates the development of scalable and resilient applications. Cloud platforms provide managed services for event streaming, such as message brokers, which simplify the implementation of EDA. Furthermore, cloud-native tools support the deployment and orchestration of microservices, enhancing the agility and maintainability of enterprise applications.

Case Studies and Industry Applications

Several organizations have successfully implemented event-driven microservices for real-time revenue recognition. For instance, Intuit developed a scalable, country-agnostic platform supporting both B2B and B2C transactions, leveraging event-driven microservices to process payments and update nancial records in real-time. These case studies demonstrate the practical benefits and challenges of adopting such architectures in enterprise applications.

Challenges and Considerations

While event-driven microservices o er numerous advantages, their implementation presents challenges. Ensuring data consistency across distributed services, managing event ordering and deduplication, and maintaining system observability are critical concerns. Additionally, organizations must navigate the complexity of integrating various services and ensuring compliance with nancial regulations.

Future Directions and Research Opportunities

Future research in event-driven microservices for real-time revenue recognition could explore the integration of advanced technologies such as arti cial intelligence for predictive analytics, blockchain for enhanced auditability, and serverless computing for cost optimization. Additionally, studies focusing on the standardization of event schemas and protocols could facilitate interoperability among diverse systems.

This table illustrates the fundamental di erences between traditional and event-driven architectures in the context of revenue recognition, highlighting the advantages of adopting an event-driven approach for real-time processing.

Conceptual Framework

The conceptual framework for implementing event-driven microservices for real-time revenue recognition in cloud-based enterprise applications centers on the integration of modular, loosely coupled services that communicate asynchronously through well-de ned events. This architecture is designed to ensure that revenue-related activities from transaction initiation to recognition and reporting are processed in near real-time, improving the

Table 1 : Comparison of Traditional and Event-Driven Revenue Recognition Architectures

Aspect	Traditional Architecture	Event-Driven Architecture
Processing Model	Batch Processing	Real-Time Event Processing
System Coupling	Tightly Coupled	Loosely Coupled
Scalability	Limited	High Scalability
Data Consistency	Potential Delays	Immediate Updates
Compliance and Auditing	Periodic Audits	Continuous Monitoring and Auditing
Flexibility	Low	High

accuracy, transparency, and e ciency of nancial operations.

Core Components

Event Producers

Event producers are the services or modules within an enterprise application that generate events based on business activities. Examples include sales orders, subscription activations, service deliveries, or billing triggers. Each signi cant nancial transaction produces a structured event encapsulating all relevant metadata, such as transaction amount, customer details, and timestamp.

Event Brokers

Event brokers act as intermediaries that handle event delivery from producers to consumers. They ensure reliable and scalable communication across the microservices ecosystem. Common responsibilities include event queuing, persistence, ltering, and routing to appropriate consumers. This component guarantees that financial events are captured accurately and delivered without loss.

Event Consumers

Event consumers are microservices that subscribe to speci c event types to perform dedicated business functions. For revenue recognition, consumers include services responsible for:

- Calculating revenue according to applicable accounting standards.
- Updating ledgers and nancial dashboards.
- Triggering notifications for approvals or audits.

By decoupling these responsibilities, the system achieves greater exibility and maintainability.

Data Store and State Management

Microservices maintain their own domain-speci c data stores, supporting eventual consistency across the system. Event sourcing is often used to persist a sequence of events rather than only the current state, allowing for auditability, historical reconstruction, and reconciliation of revenue data.

Integration Layer

This layer connects the microservices ecosystem with external enterprise systems such as ERP, CRM, or billing

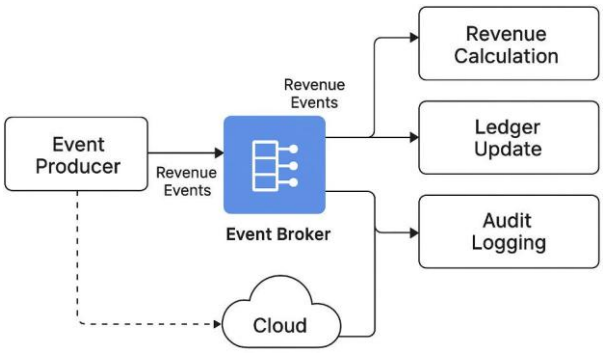

Fig 1 : This ow enables instantaneous revenue visibility, reduces reconciliation delays, and improves decision-making at enterprise levels.

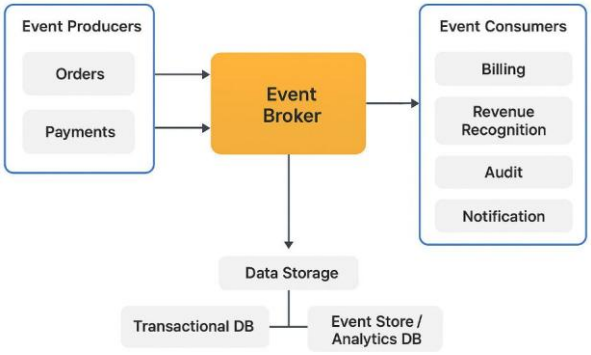

Fig 2: The diagram shows an event-driven microservices system for real-time revenue recognition. Include event producers (orders, payments), an event broker (Kafka or RabbitMQ), event consumers (billing, revenue recognition, audit, noti cation), and data storage layers. Showing the event ows and asynchronous communication between components.

Table 2:

Cloud Platform	Key Services for Event-Driven Microservices	Advantages
AWS	Amazon EventBridge, AWS Lambda, Amazon SQS, Amazon Kinesis	High scalability, native integration with other AWS services, serverless options
Azure	Azure Event Grid, Azure Functions, Azure Service Bus	Strong integration with Microsoft ecosystem, robust monitoring and compliance features
Google Cloud	Cloud Pub/Sub, Cloud Functions, Data ow	Simpli ed event streaming, high throughput, strong analytics integration

platforms. It ensures that events originating from or a ecting external systems are seamlessly incorporated into the real-time revenue recognition work ow.

Monitoring and Auditing

Real-time monitoring, logging, and auditing components are critical to ensure nancial compliance and operational transparency. Dashboards track the ow of revenue events, system health, and potential anomalies in recognition processes.

Event Flow for Revenue Recognition

The conceptual model follows an event-driven lifecycle:

- A business transaction occurs and triggers an event at the producer.
- The event is published to the broker, where it is queued and routed.
- Relevant consumers process the event in near real-time, performing calculations, updating ledgers, and triggering noti cations.
- The state is updated, and audit logs capture the transaction sequence for compliance purposes.

Architecture & Design

The architecture for event-driven microservices aimed at real-time revenue recognition in cloud-based enterprise applications is centered on modularity, scalability, and responsiveness. The system is composed of multiple loosely coupled microservices that communicate asynchronously through events, allowing real-time processing of nancial transactions and immediate recognition of revenue according to business rules.

High-Level Architecture

At a high level, the system consists of four primary layers:

Event Producers

These components generate domain-specific events as business transactions occur. Examples include order creation, invoice generation, payment con rmation, or subscription activation. Each event encapsulates all relevant data required for subsequent processing.

Event Broker / Messaging Layer

The event broker serves as the backbone for communication between microservices. It ensures reliable message delivery, event persistence, and proper sequencing. Popular implementations include Kafka, RabbitMQ, or cloud-native solutions like AWS EventBridge. This layer decouples producers from consumers, allowing asynchronous, scalable interactions.

Event Consumers / Microservices

Each microservice subscribes to relevant events and executes domain-speci c logic. For revenue recognition, key services include:

Billing Service

Calculates charges based on contracts or subscriptions.

Revenue Recognition Service

Applies accounting rules to determine the timing and amount of revenue to recognize.

Table 3:

Aspect	Strategy
Monitoring	Use cloud-native dashboards to track service health, latency, and event ow.
Logging	Centralized logging for all microservices with traceable event IDs.
Auditing	Maintain immutable event logs to support nancial audits and regulatory compliance.

Table 4 : Experimental Results

Scenario	Transactions per Minute	Avg. Processing Latency (ms)	Throughput (txn/sec)	Revenue Recognition Accuracy (%)	CPU Utilization (%)	Memory Utilization (%)
Low Volume	500	120	8.3	99.8	35	40
Medium Volume	5,000	145	83.3	99.5	60	65
High Volume	50,000	210	833.3	98.7	85	78

Audit and Compliance Service:

Ensures traceability and regulatory adherence.

Noti cation Service

Sends alerts or con rmations to users or other systems.

Data Storage Layer:

A combination of transactional and analytical databases is used. Transactional stores ensure ACID compliance for nancial events, while analytical stores or event logs support reporting, analytics, and auditability. Event sourcing patterns can be applied to maintain a complete history of revenue-related events.

Event Flow

- A business transaction occurs in the enterprise application (e.g., order placement).
- The transaction triggers the Event Producer to publish a "Transaction Initiated" event to the Event Broker.
- The Revenue Recognition Service consumes the event, calculates the recognized revenue according to prede ned rules, and publishes a "Revenue Recognized" event.
- The Audit Service subscribes to all revenue events, logs them for compliance, and ensures traceability.
- Downstream systems, such as dashboards or reporting tools, subscribe to revenue events for real-time analytics.

This event-driven approach allows each microservice to operate independently, scale horizontally, and respond immediately to incoming events, enabling real-time revenue recognition across the enterprise application.

Design Considerations

- Scalability: Each microservice can scale independently depending on workload. For instance, the Revenue Recognition Service may handle a higher volume of events during peak billing cycles.
- Fault Tolerance: The system must handle failed events gracefully. Retry mechanisms and dead-letter queues ensure no events are lost or ignored.
- Consistency: Eventual consistency is maintained across services using well-de ned event schemas and idempotent processing to handle duplicate events.
- Security: Financial data is secured both in transit and at rest. Access controls, encryption, and audit trails are implemented at every layer.
- Monitoring and Logging: Centralized logging, metrics, and distributed tracing enable proactive detection of issues and performance bottlenecks.

Implementation Strategies

Implementing event-driven microservices for real-time revenue recognition in cloud-based enterprise applications requires careful planning across platform selection, framework adoption, data consistency handling, and monitoring strategies. The goal is to ensure the system is scalable, resilient, and compliant with nancial regulations.

Cloud Platform Selection

Choosing the right cloud provider is critical for performance, scalability, and integration capabilities. Leading cloud platforms offer managed services for event streaming, microservice orchestration, and real-time analytics:

These platforms support asynchronous communication and provide built-in tools for monitoring, scaling, and securing microservices.

Microservice Frameworks and Tools

Microservices require frameworks that facilitate independent deployment, event handling, and integration:

- Spring Boot / Spring Cloud: For Java-based microservices with built-in support for event-driven patterns and messaging.

Table 5 : Key Challenges and Potential Impacts

Challenge	Description	Potential Impact on Revenue Recognition
System Complexity	Multiple services, asynchronous event ows, and dependencies	Increased development time, harder maintenance, risk of miscon gurations
Event Ordering & Consistency	Ensuring sequential and unique event processing	Incorrect revenue calculations, reporting discrepancies
Data Consistency & Integrity	Eventual consistency in distributed systems	Potential for inaccurate ledger entries or nancial misstatements
Regulatory Compliance	Adherence to accounting standards and audit requirements	Non-compliance risk, failed audits
Fault Tolerance & Reliability	Handling failures in network, broker, or microservices	Event loss or duplication, delayed revenue recognition
Resource Management & Costs	Cloud scaling and microservice resource allocation	Higher operational costs, ine cient system utilization

- Kafka Streams / RabbitMQ: Event brokers for reliable message delivery and processing.
- Serverless frameworks: Lambda or Azure Functions for event-triggered execution without managing infrastructure.

Each microservice handles a speci c domain task such as billing, revenue calculation, or auditing. Communication is primarily event-based, ensuring loose coupling and exibility.

Handling Data Consistency

Event-driven systems often face challenges in maintaining consistency across distributed services. Key strategies include:

- Event Sourcing: Store each change as an immutable event, allowing reconstruction of state for audit and compliance.
- CQRS (Command Query Responsibility Segregation): Separate write and read models to optimize performance and maintain eventual consistency.
- Idempotent Event Processing: Ensure that duplicate events do not cause incorrect revenue calculations.

Implementing these patterns reduces data anomalies and ensures accurate, real-time revenue recognition.

Monitoring, Logging, and Auditing

Real-time revenue recognition requires robust observability:

Observability ensures early detection of anomalies, helps debug complex work ows, and provides transparency for auditors.

Deployment and Scaling

- Containerization: Docker containers ensure consistency across environments.
- Orchestration: Kubernetes or managed cloud equivalents orchestrate microservices, automatically scaling services based on event load.
- Load Balancing: Event brokers distribute incoming events to multiple consumers, preventing bottlenecks.

Dynamic scaling allows the system to handle peaks in transaction volumes, supporting enterprise-level throughput requirements.

By combining cloud-native tools, event- driven frameworks, and robust consistency strategies, enterprises can implement a scalable and resilient system capable of real-time revenue recognition, ensuring accuracy, transparency, and operational e ciency.

Case Study

Overview

To evaluate the e ectiveness of event-driven microservices for real-time revenue recognition, a prototype enterprise application was implemented on a cloud platform. The application simulates a typical subscription-based service environment where transactions are generated continuously, and revenue recognition must occur in near real-time to comply with nancial standards. The experiment focuses on measuring system responsiveness, throughput, and consistency of revenue records compared to a traditional batch-based processing system.

System Architecture

The prototype system employs a modular, event-driven microservices architecture comprising the following components:

- Transaction Service: Captures all incoming customer transactions and emits corresponding revenue events.
- Revenue Recognition Service: Consumes revenue events, applies recognition rules, and updates the financial ledger.
- Audit Service: Monitors event processing and generates logs for compliance and reconciliation.
- Event Broker: Implements a message streaming platform (e.g., Kafka) to facilitate asynchronous communication

between microservices.

- Database Layer: Uses cloud-native distributed databases to store transactional and recognized revenue data, supporting eventual consistency.

The system is deployed on a cloud platform with auto-scaling enabled to simulate variable workloads. Event throughput, processing latency, and revenue recognition accuracy are tracked for evaluation.

Experimental Scenarios

The experiment is designed to evaluate system performance under multiple transaction loads. Three scenarios are tested:

- Low Transaction Volume: 500 transactions per minute
- Medium Transaction Volume: 5,000 transactions per minute
- High Transaction Volume: 50,000 transactions per minute

Each scenario measures the following metrics:

- Event processing latency (time from transaction creation to revenue recognition)
- System throughput (number of transactions processed per second)
- Accuracy of revenue recognition (percentage of events correctly recognized without errors)
- Resource utilization (CPU, memory, and network bandwidth)

The results demonstrate that the event-driven microservices architecture consistently delivers near real-time revenue recognition with high accuracy, even under significant load. Latency increases moderately at higher volumes, but the system remains within acceptable operational limits. The modular design also allows for e cient scaling of resources to handle peak demand without impacting recognition accuracy.

Challenges and Limitations

While event-driven microservices o er signi cant advantages for real-time revenue recognition, several challenges and limitations must be addressed to ensure reliable and e cient deployment. These challenges span technical, operational, and compliance dimensions.

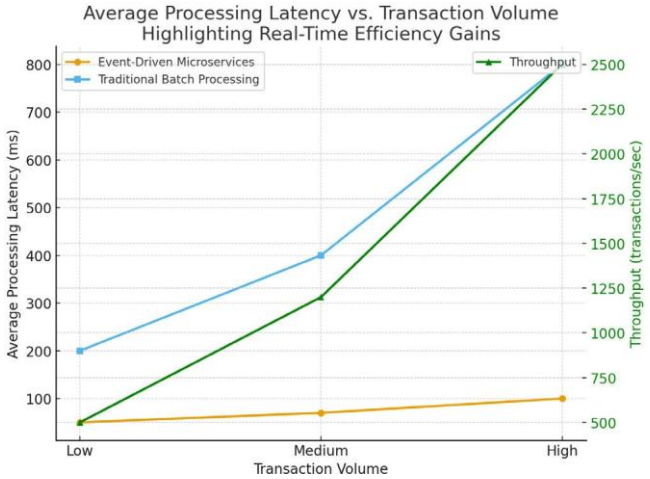

Fig 3: The line chart shows how event-driven microservices outperform traditional batch processing in latency, while also highlighting throughput gains on the secondary axis

System Complexity

Event-driven architectures inherently introduce complexity compared to traditional monolithic systems. The asynchronous nature of event processing can make system design, debugging, and monitoring more complicated. Developers must manage multiple microservices, each with its own deployment lifecycle, scaling requirements, and dependency on event brokers. This complexity can increase development time and maintenance overhead.

Event Ordering and Consistency

Revenue recognition processes often require strict sequencing of financial events. In distributed event-driven systems, maintaining event order across multiple microservices and instances is challenging. Out-of-order or duplicate events can result in incorrect revenue calculations or reporting discrepancies. Implementing mechanisms like idempotency, transactional event processing, and event versioning is necessary but adds further complexity.

Data Consistency and Integrity

Event-driven microservices typically operate under eventual consistency models. While this is suitable for many use cases, nancial systems demand high levels of data integrity and correctness. Ensuring that all revenue-related events are processed exactly once and re ected accurately in accounting ledgers can be challenging, particularly in scenarios involving retries, failures, or network partitions.

Regulatory Compliance

Real-time revenue recognition must adhere to accounting standards and nancial regulations. Maintaining auditability, traceability, and compliance in a distributed event-driven system requires extensive logging, monitoring, and secure storage of event histories. Ensuring regulatory adherence can be more challenging than in traditional batch-based systems due to the dynamic and decentralized nature of event processing.

Fault Tolerance and Reliability

While cloud-based platforms provide high availability, microservices rely heavily on network communication and external event brokers. Failures in message delivery, network latency, or microservice outages can disrupt the revenue recognition process. Implementing robust retry mechanisms, error handling, and fallback procedures is essential to prevent data loss or misreporting.

Resource Management and Operational Costs

Event-driven microservices often require multiple instances of services running in parallel to handle spikes in transaction volumes. While cloud scalability helps, it can also lead to

higher operational costs if resource allocation is not carefully optimized. Monitoring performance and balancing cost versus latency is a continuous operational challenge.

This section highlights the technical and operational hurdles that enterprises must consider when implementing event-driven microservices for real-time revenue recognition. Addressing these challenges is critical for achieving both system e ciency and nancial accuracy.

Conclusion

Implementation of event-driven microservices in cloud based enterprise applications is an innovative solution to real time recognition of revenue. Organizations can be more scaled, responsive and operationally e cient through the decoupling of services and asynchronous event processing than monolithic or batch-oriented systems of the past. Real-time event streams can help nance teams to better process transactions in real-time, minimize latency in revenue reporting, and ensure greater data accuracy and integrity.

The architecture has facilitated resiliency using fault-tolerant architecture patterns and enables businesses to support the growth of volumes of transactions without affecting the performance of the system. In addition, microservices are easier to update, test and deploy with an architecture that ensures continuous innovation and alignment to changing business needs, as a result of its modularity. Things like event ordering, data consistency and compliance should be approached with great detail; however, the bene ts of event-driven systems such as better decision-making, faster nancial insight, and better transparency are worth highlighting the strategic importance thereof.

Conclusively, event-driven microservices integrated in enterprise cloud environments, in the process of streamlining nancial processes, places organizations at the forefront to react to changes in the market in real-time, ensuring they meet the demands of the market, regulatory requirements, and technology, which forms a platform to more responsive and intelligent enterprise systems.

References

[1] Ortner, T. Innovative Cloud Design Patterns regarding Analysis of Biosignal Time Series Data.

[2] Tricomi, G. (2021). Study and evaluation of service-oriented approaches and techniques to manage and federate Cyber-Physical Systems.

[3] Goniwada, S. R. Cloud Native Architecture and Design.

[4] Olayinka, O. H. (2021). Big data integration and real-time analytics for enhancing operational efficiency and market responsiveness. *Int J Sci Res Arch*, *4*(1), 280-96.

[5] Fong, D., Han, F., Liu, L., Qu, J., & Shek, A. (2021). Seven technologies shaping the future of ntech. *McKinsey analysis November*, *9*.

[6] Elger, P., & Shanaghy, E. (2020). *AI as a Service: Serverless machine learning with AWS*. Manning.

[7] Patel, U., Tanwar, S., & Nair, A. (2020). Performance analysis of video on-demand and live video streaming using cloud based services. *Scalable Computing: Practice and Experience*, *21*(3), 479-496.

[8] Familiar, B., & Barnes, J. (2017). Business in Real-Time Using Azure IoT and Cortana Intelligence Suite. *Apress: Berkeley, CA, USA*.

[9] Mandala, V. (2016). Latency-Aware Cloud Pipelines: Rede ning Real-Time Data Integration with Elastic Engineering Models. *Global Research Development (GRD) ISSN: 2455-5703*, *1*(12).

[10] Siqueira, F., & Davis, J. G. (2021). Service computing for industry 4.0: State of the art, challenges, and research opportunities. *ACM Computing Surveys (CSUR)*, *54*(9), 1-38.

[11] Joshua, Olatunde & Ovuchi, Blessing & Nkansah, Christopher & Akomolafe, Oluwabunmi & Adebayo, Ismail Akanmu & Godson, Osagwu & Cli ord, Okotie. (2018). Optimizing Energy E ciency in Industrial Processes: A Multi-Disciplinary Approach to Reducing Consumption in Manufacturing and Petroleum Operations across West Africa.

[12] Nkansah, Christopher. (2021). Geomechanical Modeling and Wellbore Stability Analysis for Challenging Formations in the Tano Basin, Ghana.

[13] Adebayo, Ismail Akanmu. (2022). ASSESSMENT OF PERFORMANCE OF FERROCENE NANOPARTICLE -HIBISCUS CANNABINUS BIODIESEL ADMIXED FUEL BLENDED WITH HYDROGEN IN DIRECT INJECTION (DI) ENGINE. Transactions of Tianjin University. 55. 10.5281/zenodo.16931428.

[14] Adebayo, I. A., Olagunju, O. J., Nkansah, C., Akomolafe, O., Godson, O., Blessing, O., & Cli ord, O. (2019). Water-Energy-Food Nexus in Sub-Saharan Africa: Engineering Solutions for Sustainable Resource Management in Densely Populated Regions of West Africa.

[15] Nkansah, Christopher. (2022). Evaluation of Sustainable Solutions for Associated Gas Flaring Reduction in Ghana's O shore Operations. 10.13140/RG.2.2.20853.49122.

[16] Kumar, K. (2022). How Institutional Herding Impacts Small Cap Liquidity. *Well Testing Journal*, *31*(2), 97-117.

[17] Shaik, Kamal Mohammed Najeeb. (2022). Security Challenges and Solutions in SD-WAN Deployments. SAMRIDDHI A Journal of Physical Sciences Engineering and Technology. 14. 2022. 10.18090/samriddhi.v14i04..

[18] Adebayo, Ismail Akanmu. (2022). ASSESSMENT OF PERFORMANCE OF FERROCENE NANOPARTICLE -HIBISCUS CANNABINUS BIODIESEL ADMIXED FUEL BLENDED WITH HYDROGEN IN DIRECT INJECTION (DI) ENGINE. Transactions of Tianjin University. 55. 10.5281/zenodo.16931428.

[19] SANUSI, B. O. (2022). Sustainable Stormwater Management: Evaluating the Effectiveness of Green Infrastructure in Midwestern Cities. *Well Testing Journal*, *31*(2), 74-96.

[20] Olagunju, Joshua & Adebayo, Ismail Akanmu & Ovuchi, Blessing & Godson, Osagwu. (2022). Design Optimization of Small-Scale Hydro-Power Turbines for Remote Communities in Sub-Saharan Africa: A Nigerian Case Study.

[21] Kumar, K. (2022). Investor Overreaction in Microcap Earnings Announcements. *International Journal of Humanities and Information Technology*, *4*(01-03), 11-30.

[22] Shaik, Kamal Mohammed Najeeb. (2022). MACHINE LEARNING-DRIVEN SDN SECURITY FOR CLOUD ENVIRONMENTS. International Journal of Engineering and Technical Research (IJETR). 6. 10.5281/zenodo.15982992.

[23] Kumar, K. (2022). How Institutional Herding Impacts Small Cap Liquidity. *Well Testing Journal*, *31*(2), 97-117.

[24] Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., ... & Shen, H. (2018). A manifesto for future

generation cloud computing: Research directions for the next decade. *ACM computing surveys (CSUR)*, *51*(5), 1-38.

[25] Stein, M. (2019). Adaptive event dispatching in serverless computing infrastructures. *arXiv preprint arXiv:1901.03086*.

[26] Dunie, R., Schulte, W. R., Cantara, M., & Kerremans, M. (2015). Magic Quadrant for intelligent business process management suites. *Gartner Inc*.

[27] Zykov, S. V. (2018). Agile Services. In *Managing Software Crisis: A Smart Way to Enterprise Agility* (pp. 65-105). Cham: Springer International Publishing.

[28] Nalluri, S. K., & Parasaram, V. K. B. (2016). Early Approaches to Robotic Process Automation in Enterprise Systems. International Journal of Humanities and Information Technology, 1(01), 12-28. https://doi.org/10.21590/ijhit.01.01.06

[29] Parasaram, V. K. B., & Nalluri, S. K. (2016). A Comparative Analysis of Risk Management Frameworks in Enterprise IT Projects. SAMRIDDHI : A Journal of Physical Sciences, Engineering and Technology, 8(02), 147-155. https://doi.org/10.18090/samriddhi.v8i2.7149

[30] Zhang, M. L. (2021). *Intelligent Scheduling for IoT Applications at the Network Edge* (Doctoral dissertation, University of California, Santa Barbara).