Comparing Serverless and Microservices Architecture Patterns in Fintech Space

Ramasankar Molleti, Independent Researcher,
email: sankar276@gmail.com

ABSTRACT
The study aims to compare the serverless and microservices architectural patterns in the FinTech sector in 2021. It looks at the history of these architectures, their key concepts as well as complex techniques of applying these architectures to a Fintech context. Some of the main issues tackled in the study are specific to these architectures; they include security, compliance, scalability, and data consistency problems typical for the FinTech industry. Thus, the study presents a performance evaluation of serverless and microservices for financial services based on the analysis of performance indicators and the use of actual cases. The study concludes that while there are relative merits in both architectures, most FinTech firms are integrating the two to gain the benefits of both.

Keywords: ***FinTech Architecture, Serverless Computing, Microservices, Cloud-Native Finance, Scalability in FinTech***

I. Introduction

The FinTech industry has been transformed due to the development of the new technologies and new generation's expectations. This evolution is based on the decision to select an appropriate architectural strategy to construct reliable, extensible, and secure financial applications. This study focuses on comparing two major architectural patterns, serverless and microservices architecture. FaaS is a cloud computing model that is frequently described as serverless computing because it allows developers to create and deploy applications without having to worry about the supporting infrastructure. This model also suggests that operational overhead will be cut, scaling will be managed automatically and cost structures will involve pay-per-use, which will be interesting for FinTech startups and large institutions. While, the microservices architecture is the development of applications as small, autonomously deployable services, where each service is a separate process and communicates with other services simply. They include the following; Modularity is improved, and it is easier to scale and adopt several technologies for the different components.

In the process of attempting to grow while offering the best security, regulatory compliance, and efficiency, FinTech companies face a significant decision: whether to follow the serverless architecture or microservices architecture or use both. The aim is to give an overview of all these architectural patterns and compare them to FinTech applications, especially the key issues about their principles, implementation, and issues. After discussing practical examples and operational characteristics, it will be possible to assess the applicability of the described architectures to the financial industry requirements regarding the availability of high-volume transactions, data synchronization, and compliance.

II. Evolution of Architecture Patterns in FinTech

The use of technological advancement in the provision of financial services commonly known as FinTech has experienced a revolution in its architectural strategies in the last few years [1]. This evolution has resulted from the need to address the issues of flexibility, growth, and creativity in financial solutions.

Traditional Monolithic Architectures

FinTech applications were built using monolithic architectures at first. These systems were characterized by:

- Single, tightly-coupled codebase
- Shared database
- Limited scalability

Lengthy development and deployment cycles

Monolithic architectures provided easy development and deployment but were unable to provide what the financial industry needs today.

Shift towards Distributed Systems

When the firms began to expand and the products they offered started getting complicated, there was an incremental move toward distributed systems.

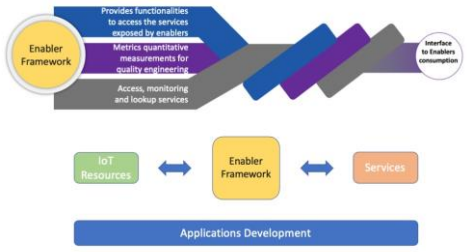

Figure 1: Service-Oriented Architecture

(Source: https://pub.mdpi-res.com/)

This transition was marked by:

- Service-Oriented Architecture (SOA) adoption
- Increased modularity
- Improved scalability and fault tolerance
- Enhanced reusability of components

Distributed systems enabled FinTech firms to divide the applications into smaller services which could easily be handled and enhanced both the system reliability and flexibility to be enhanced.

Emergence of Microservices and Serverless Paradigms

The last major change in the architecture of FinTechs has been the transition to microservices and serverless architecture [2]. These modern approaches offer:

- Fine-grained, loosely-coupled services
- Independent deployment and scaling
- Improved fault isolation
- Faster time-to-market for new features

Microservices are used in the context of building a system in which FinTech companies can create, implement, and evolve each segment separately, and serverless is the capability to execute code without managing the servers. The evolution is due to the FinTech industry's constant search for architectures that allow fast innovation and growth while keeping up with the security and compliance levels of the financial market [3]. In this context, the industry advances in implementing these patterns, sometimes using fragments from several approaches to accommodate concrete business requirements and technological opportunities.

III. Core Principles of Serverless and Microservices Architectures

FinTech has incorporated serverless and microservices to accommodate scalability, agility, and innovation. Despite having common objectives, these architectures greatly vary in their implementations and principles.

Serverless Computing Fundamentals

Serverless computing also known as Function-as-a-Service (FaaS) is an execution model in which the cloud provider takes care of the server's resource allocation and scheduling. Key principles include:

a) Event-Driven Execution: An event is anything that happens such as an HTTP request, changes to the database, or a scheduled event that will cause a function to execute.

b) Stateless Nature: Functions do not have a state between invocations; this enhances scalability and reduces the needed model in programming.

c) Auto-scaling: The platform means that resources can scale from zero volume to the maximum volume [4].

d) Pay-per-Use Pricing: Charges are by the rate of actual consumption of the time slices allocated to compute as opposed to 'licenses'.

e) Managed Infrastructure: For all the server-related issues, the responsibility lies on the cloud provider thus enabling developers to work on code only.

Formula for Serverless Cost Calculation:

Total Cost = (Number of Invocations × Execution Time × Cost per 100ms) + (Memory Allocated × Execution Time × Memory Price)

Microservices Architecture Principles

Microservices architecture can be defined as a method of constructing a large application as a collection of small services that are independent and collaborate using simple methods. Core principles include:

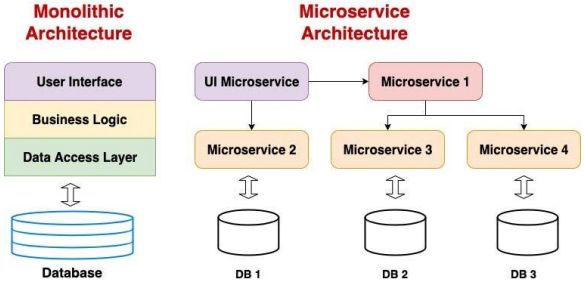

Figure 2: Monolithic v/s Microservice Architecture

(Source: https://www.google.com/)

a) Service Independence: It is also characterized by the fact that every microservice is built, released, and can be scaled on its own [5].

b) Decentralized Data Management: Every service has its database; it can be different instances of the same DBMS or it can be a completely different DBMS.

c) Design for Failure: Microservices are intended to be reactive and able to work with the failure of other services in a suitable manner.

d) Evolutionary Design: The architecture enables the enhancement of the application and in particular the service-oriented structure which can easily be updated and replaced.

e) Automation: CI/CD processes are also prescriptive for dealing with the complexity of multiple services.

Key Differences and Similarities

While both architectures aim to improve scalability and agility, they differ in several key aspects:

Aspect	Serverless	Microservices
Deployment Unit	Function	Service
State Management	Stateless	Can be stateful or stateless
Infrastructure Management	Fully managed by the provider	Managed by the development team
Scaling	Automatic and instant	Manual or automated, but requires configuration
Development Focus	Individual functions	Service-level APIs
Long-running Processes	Limited support	Fully supported

Pricing Model	Pay-per-execution	Pay-per-allocated-resource

Table 1: Comparison of Serverless and Microservices Architectures

Similarities between the two architectures include:

Modularity: Both approaches break down applications into smaller, manageable units [6].

Scalability: Both architectures support independent scaling of components.

Technology Diversity: Both allow different services/functions to use different technologies.

DevOps Culture: Both benefit from and often require DevOps practices for effective implementation.

Architectural Considerations for FinTech

In the FinTech context, several factors influence the choice between serverless and microservices:

a) Regulatory Compliance: Some of the compliance-type regulations that are likely to apply to a FinTech application include GDPR, PSD2, and SOX. Microservices might be useful to have more control when it comes to applying compliance requirements on the enterprise level.

b) Transaction Processing: While the large number of transactions with low response time might be more suitable for microservices whereas application of serverless can be more suitable in case of sporadic and bursty workloads like fraud detection.

c) Data Consistency: Data integrity is generally important in the operation of many financial applications [7]. Microservices with their database may be suitable for a highly transactional kind of scenario.

d) Cost Predictability: Even though serverless may be cheaper in terms of unpredictable workloads, microservices may have better cost efficiency for stable, heavy usage.

Formula for Microservices Cost Estimation:

Total Cost = Σ (Service Instance Count × Instance Cost) + Data Transfer Costs + Storage Costs

Both serverless and microservices architectures have numerous advantages in the context of FinTech applications. Its decision usually depends on certain particular functions, workloads, and circumstances that are characteristic of the organization. Systems of today's FinTech firms have many functions implemented using serverless; however, others use microservices for business logic and heavy computations [8]. This way, they are in a position to receive the better of the two architectures without the drawbacks associated with each of them.

IV. Advanced Implementation Techniques in FinTech

This area focuses on the more complex patterns and solutions that are designed specifically for the special cases of financial technology applications.

Serverless Patterns in FinTech Applications

Serverless computing offers several advanced implementation techniques particularly suited to FinTech:

a) Event-driven Processing for Real-time Transactions

FinTech applications often require real-time processing of financial transactions. Serverless functions can be initiated by events such as:

- New transaction initiation
- Account balance changes
- Fraud detection alerts

b) Micro-billing Systems

Serverless architectures excel at handling micro-billing scenarios common in modern FinTech applications. Functions can be designed to:

- Calculate usage-based fees
- Apply tiered pricing models
- Generate itemized bills

c) Scheduled Financial Operations

Leveraging serverless scheduled events for:

- End-of-day reconciliation
- Periodic interest calculations
- Automated report generation

d) Secure API Gateway Integration

Implementing secure API gateways with serverless functions for:

- Authentication and authorization
- Rate limiting
- Request/response transformation

Function Type	Use Case	Trigger
Transaction Processor	Payment processing	API Gateway event
Fraud Detector	Real-time transaction screening	Database change event
Report Generator	Daily financial summaries	Scheduled event
Account Reconciler	End-of-day balance checks	Scheduled event
Notification Sender	Transaction alerts	Queue event

Table 2: Serverless Function Types in FinTech

Microservices Patterns in FinTech Applications

Microservices architecture in FinTech leverages several advanced patterns [9].

a) Domain-Driven Design (DDD)

Applying DDD principles to define bounded contexts for microservices:

- Account Management Service
- Payment Processing Service
- Risk Assessment Service
- Compliance Monitoring Service

b) Event Sourcing and CQRS

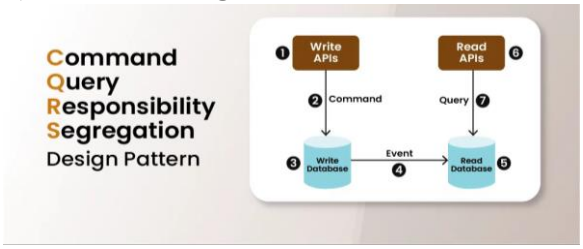

Figure 3: Command Query Responsibility Segregation

(Source: https://media.geeksforgeeks.org/)

Implementing Event Sourcing and Command Query Responsibility Segregation (CQRS) for:

Maintaining an immutable log of all financial transactions

Separating read and write operations for optimized performance

c) API Composition and Backend for Frontend (BFF)

Utilizing API composition to:

Aggregate data from multiple microservices

Implement BFF pattern for different client types (mobile, web, third-party)

d) Circuit Breaker Pattern

Implementing circuit breakers to:

Prevent cascading failures in interconnected financial services

Gracefully handle service unavailability

e) Saga Pattern for Distributed Transactions

Managing complex, multi-step financial transactions across multiple services [10].

Coordinating operations like fund transfers between accounts

Ensuring consistency in distributed systems

Hybrid Approaches and Their Applicability

Many FinTech companies are adopting hybrid architectures, combining elements of both serverless and microservices:

a) Serverless Functions as Microservice Extensions

Using serverless functions to extend microservices capabilities:

- Handling spiky workloads

- Implementing cross-cutting concerns (e.g., logging, monitoring)

b) Event-Driven Communication between Microservices and Serverless Functions

Leveraging message queues and event streaming platforms to facilitate communication:

- Apache Kafka for high-throughput event streaming
- Amazon SQS for decoupled, asynchronous processing

c) Serverless Data Processing Pipelines

Implementing data processing workflows using a combination of microservices and serverless functions [11].

ETL processes for financial data

Real-time analytics on transaction streams

Formula for Hybrid Architecture Cost Estimation:

Total Cost = (Serverless Costs) + (Microservices Costs) + (Integration Costs)

Where:

Serverless Costs = Σ (Function Invocations × Execution Time × Cost per 100ms)

Microservices Costs = Σ (Service Instance Count × Instance Cost)

Integration Costs = Data Transfer Costs + API Gateway Costs

Performance Optimization Techniques

Regardless of the chosen architecture, FinTech applications require careful performance optimization:

a) Caching Strategies

Implementing multi-level caching:

In-memory caches for frequently accessed financial data

Distributed caches for shared state across services

b) Asynchronous Processing

Utilizing asynchronous patterns for non-critical operations:

Background processing of analytical tasks

Deferred execution of reporting functions

c) Database Optimization

Applying advanced database techniques:

Sharding for horizontal scalability

Read replicas for improved query performance

d) Predictive Scaling

Implementing machine learning models for predictive auto-scaling [12].

Analyzing historical usage patterns

Proactively adjusting resources based on predicted demand

The implementation of advanced techniques of FinTech uses the best of the serverless and microservices architecture. Through the proper use of these patterns, FinTech companies can implement systems that are scalable as well as efficient to meet the demands of today's financial sector. The major issue is to identify the requirements of the particular component in the application and use the proper architectural pattern and implementation approach.

V. Overcoming Implementation Challenges

Applying serverless and microservices architectures in the FinTech area has its irregularities because of the high requirements for security, compliance, performance, and data management. This section discusses these issues and provides solutions to them [13].

Security and Compliance Considerations

Challenge: FinTech applications handle sensitive financial data and must adhere to strict regulatory requirements (e.g., GDPR, PSD2, SOX).

Strategies:

a) Encryption: Implement end-to-end encryption for data in transit and at rest.

Use TLS 1.3 for all network communications

Employ hardware security modules (HSMs) for key management

b) Fine-grained Access Control: Implement the least privilege principle using:

Role-Based Access Control (RBAC) for microservices

Resource-based policies for serverless functions

c) Audit Trails: Maintain comprehensive logs for all financial transactions.

Use distributed tracing tools like Jaeger or Zipkin

Implement event sourcing for immutable transaction history

d) Compliance Automation: Leverage Infrastructure as Code (IaC) to ensure compliance.

Use tools like Terraform or AWS CloudFormation

Implement automated compliance checks in CI/CD pipelines

Scalability and Performance Issues

Challenge: An application in FinTech is likely to experience high variability in load and the response time needs to be minimal when performing important operations [14].

Strategies:

a) Auto-scaling: Apply dynamic scaling for microservices and serverless functions.

Use Horizontal Pod Autoscaler of Kubernetes for microservices

Rely on the auto-scaling feature provided by the cloud providers for the serverless.

b) Caching: Caching is another technique that can be used to minimize the level of latency; this should be done at multiple levels.

It is recommended to use Redis or Memcached for distributed caching.

Make use of Application Cache for the data that is frequently retrieved.

c) Asynchronous Processing: Delegate the less important tasks to the background processes.

There are various message queues available, some of them are RabbitMQ or Apache Kafka.

Use event-driven architectures for more decoupling

d) Performance Monitoring: Ensure that the systems are constantly fine-tuned to provide optimal results.

Some of the frequently used APM tools include New Relic or Datadog.

It is necessary to set KPIs based on FinTech companies' field-specific peculiarities

Data Management and Consistency

Challenge: The challenge of keeping data synchronized across distributed systems and at the same time being highly available and high performing [15].

Strategies:

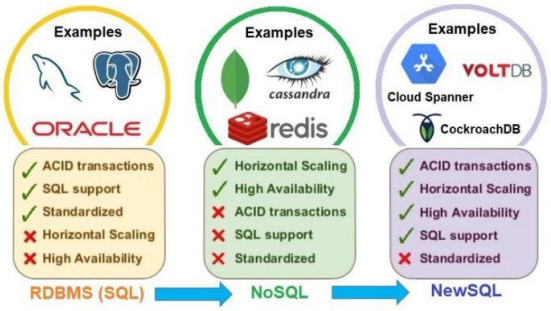

Figure 4: NewSQL

(Source: https://editor.analyticsvidhya.com/)

a) ACID Compliance: It employs appropriate databases for transactional consistency.

People can use NewSQL databases such as CockroachDB for distributed ACID transactions.

Introduce compensating transactions for the models of the eventual consistency.

b) Data Partitioning: Partition level data to enhance its capability and efficiency.

Lease always uses hash functions in such a way that it distributes the load evenly.

Use entity groups to keep data that are related in the same partition

c) Eventual Consistency: Welcome eventual consistency where applicable.

Use event sourcing and CQRS patterns

Introduce event sourcing and CQRS patterns

Introduce conflict resolution mechanisms (vector clocks, CRDTs).

d) Data Synchronization: Make sure that the services and regions maintain data consistency.

Set up CDC to enable real-time synchronization

Multi-region replication is a good practice for when the application should be available in several regions.

With the help of strong solutions for these difficulties and a cautious approach to applying serverless and microservices, FinTech companies can get the most efficiency from them [16]. The idea is to follow the right balance between the security levels, performance, and repeatability

without violating the legal and organizational requirements.

VI. Performance Analysis and Case Studies

This section includes a comparison of the serverless and microservices approaches in FinTech use cases along with practical examples and KPIs.

Comparative Analysis of Serverless vs Microservices in FinTech Scenarios

To effectively compare these architectures, we'll focus on key performance indicators (KPIs) relevant to FinTech applications:

KPI	Serverless	Microservices
Latency	Low for infrequent requests, potential cold starts	Consistent, generally low
Scalability	Automatic, rapid	Manual or automated, but requires configuration
Cost Efficiency	Pay-per-use, cost-effective for variable loads	Constant cost, efficient for steady, high loads
Development Speed	Rapid for simple functions	Moderate, depending on service complexity
Maintenance Overhead	Low, managed by the cloud provider	Higher, requires dedicated DevOps

Table 3: Performance Comparison of Serverless and Microservices in FinTech

Total Cost of Ownership (TCO):

TCO = IC + OC + MC

Where:

IC = Initial Costs (development, setup)

OC = Operational Costs (running costs, scaling costs)

MC = Maintenance Costs (updates, monitoring, troubleshooting)

Real-world Case Studies from FinTech Industry

Case Study 1: TransferWise (now Wise) - Microservices Architecture

TransferWise the international money transfer service, chose microservices as its architectural pattern to address their challenging and high throughput application [17].

Key Outcomes:

Cut new features' time-to-market in half

Better reliability of the systems, now with 99. 99% uptime

Designed to accommodate more than £4 billion a month in transactions.

Implementation Details:

Applied the concept of domain-driven design to establish the right level of granularity of services.

Introduced an event-driven system to enable real-time updates

Used containerization technique (Docker) and container orchestration (Kubernetes) for scalability.

Case Study 2: Capital One - Serverless Architecture

The large American bank, Capital One, uses its chatbot and fraud prevention services.

Key Outcomes:

Minimized infrastructure cost by 60 %

Reduced new feature time-to-market by 70 %

Enhanced capacity to accommodate a large number of transactions in millions every day

Implementation Details:

AWS Lambda used for Event-Driven Processing

Used API Gateway for API security, ease of scaling

Also used DynamoDB for the fast storage and access of low-latency data.

Metrics and Evaluation Criteria

To objectively evaluate the performance of these architectures in FinTech, consider the following metrics:

Transaction Processing Time (TPT):

TPT = Ts - Tr

Where Ts = Settlement time, Tr = Request time

Requests Per Second (RPS):

RPS = Total Requests/period (in seconds)

Error Rate (ER):

ER = (Failed Transactions / Total Transactions) * 100

Cost Per Transaction (CPT):

CPT = Total Operational Cost / Number of Transactions

Scalability Index (SI):

SI = (Performance at Peak Load / Performance at Average Load) * 100

Analysis:

The serverless architecture is found to be cheaper per transaction and more scalable than the serverful approach and hence is suitable for organizations that operate in an environment of unpredictable traffic and/or where costs are critical. Microservices show less latency and high throughput which is ideal for applications that run large volumes of work with high consistency [18].

Therefore, both of the architectures have been used in the FinTech applications successfully. The decision of whether to use serverless or microservices can be based on the use cases, the expected workload profiles, and organizational competencies. Currently, there is a trend where many FinTech companies incorporate both architectures to gain the best of each for their overall system result and cost.

VII. Future Trends and Research Directions

The FinTech sector has more architectural development opportunities in the future due to new technologies and the development of a new market. Key trends and research directions include:

1. AI-Driven Architectures: Application of the Machine learning models into the Serverless functions and Microservices for real-time decisions and Predictive Analytics.

2. Quantum-Safe Cryptography: Creating specific resistant algorithms for financial transactions with the further use of quantum computers.

3. Edge Computing in FinTech: Edge nodes are used where low latency is required for example in trading financial instruments, particularly in high-frequency trading.

4. Blockchain Integration: Diving deeper into the integration of conventional cloud solutions with distributed ledgers for improving the level of openness and security.

5. Green Computing: A study on the efficient design of serverless and microservices to align with energy objectives in the financial industry.

6. Regulatory Technology (RegTech): Designing and creating specific microservices for automated compliance checks and information reporting [19].

7. Cross-Cloud Interoperability: Studying the possibilities of proper integration of multi-cloud environments to avoid the lock-in situation and improve the reliability of the systems used in FinTech.

VIII. Conclusion

There is a benchmark to decide whether to adopt serverless or microservices for the FinTech applications that are under consideration, which includes; use case, company capability, and growth. Both have their benefits, and more businesses are starting to use the combination of both. Therefore, as the world adapts to changed dynamics in the FinTech sector, consequent research and development of these architectural patterns will define the FinTech technological breakthrough.

IX. Reference List

Journals

[1] Still, K., Lähteenmäki, I. and Seppänen, M., 2019. Innovation relationships in the emergence of Fintech ecosystems.

[2] Fan, C.F., Jindal, A. and Gerndt, M., 2020. Microservices vs Serverless: A Performance Comparison on a Cloud-native Web Application. In CLOSER (pp. 204-215).

[3] Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L. and Pallickara, S., 2018, April. Serverless computing: An investigation of factors influencing microservice performance. In 2018 IEEE international conference on cloud engineering (IC2E) (pp. 159-169). IEEE.

[4] Somma, G., Ayimba, C., Casari, P., Romano, S.P. and Mancuso, V., 2020, July. When less is more: Core-restricted container provisioning for serverless computing. In IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (pp. 1153-1159). IEEE.

[5] Bogner, J., Fritzsch, J., Wagner, S. and Zimmermann, A., 2019, March. Microservices in industry: insights into technologies, characteristics, and software quality. In 2019 IEEE international conference on software architecture companion (ICSA-C) (pp. 187-195). IEEE.

[6] Kratzke, N., 2018. A Brief History of Cloud Application Architectures: From Deployment Monoliths via Microservices to Serverless Architectures and Possible Roads Ahead.

[7] García-López, P., Sánchez-Artigas, M., Shillaker, S., Pietzuch, P., Breitgand, D., Vernik, G., Sutra, P., Tarrant, T. and Ferrer, A.J., 2019. Servermix: Tradeoffs and challenges of serverless data analytics. arXiv preprint arXiv:1907.11465.

[8] Van Eyk, E., Grohmann, J., Eismann, S., Bauer, A., Versluis, L., Toader, L., Schmitt, N., Herbst, N., Abad, C.L. and Iosup, A., 2019. The SPEC-RG reference architecture for FaaS: From microservices and containers to serverless platforms. IEEE Internet Computing, 23(6), pp.7-18.

[9] Rademacher, F., Sachweh, S. and Zündorf, A., 2018. Towards a UML profile for domain-driven design of microservice architectures. In Software Engineering and Formal Methods: SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA, Trento, Italy, September 4-5, 2017, Revised Selected Papers 15 (pp. 230-245). Springer International Publishing.

[10] Štefanko, M., Chaloupka, O., Rossi, B., van Sinderen, M. and Maciaszek, L., 2019, July. The saga pattern in a reactive microservices environment. In Proc. 14th Int. Conf. Softw. Technologies (ICSOFT 2019) (pp. 483-490). Prague, Czech Republic: SciTePress.

[11] Cordingly, R., Yu, H., Hoang, V., Perez, D., Foster, D., Sadeghi, Z., Hatchett, R. and Lloyd, W.J., 2020, August. Implications of programming language selection for serverless data processing pipelines. In 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech) (pp. 704-711). IEEE.

[12] Chithrananda, S., Grand, G. and Ramsundar, B., 2020. ChemBERTa: large-scale self-supervised pretraining for molecular property prediction. arXiv preprint arXiv:2010.09885.

[13] Muhammad, T., Munir, M.T., Munir, M.Z. and Zafar, M.W., 2018. Elevating Business Operations: The Transformative Power of Cloud Computing. International Journal of Computer Science and Technology, 2(1), pp.1-21.

[14] Kumar, M., 2019. Serverless architectures review, future trend and the solutions to open problems. American Journal of Software Engineering, 6(1), pp.1-10.

[15] García-López, P., Sánchez-Artigas, M., Shillaker, S., Pietzuch, P., Breitgand, D., Vernik,

G., Sutra, P., Tarrant, T. and Ferrer, A.J., 2019. Servermix: Tradeoffs and challenges of serverless data analytics. arXiv preprint arXiv:1907.11465.

[16] Smid, A., Wang, R. and Cerny, T., 2019, September. Case study on data communication in microservice architecture. In Proceedings of the Conference on Research in Adaptive and Convergent Systems (pp. 261-267).

[17] Papadis, N. and Tassiulas, L., 2020. Blockchain-based payment channel networks: Challenges and recent advances. IEEE Access, 8, pp.227596-227609.

[18] Gan, Y. and Delimitrou, C., 2018. The architectural implications of cloud microservices. IEEE Computer Architecture Letters, 17(2), pp.155-158.

[19] Johansson, E., Sutinen, K., Lassila, J., Lang, V., Martikainen, M. and Lehner, O.M., 2019. Regtech-a necessary tool to keep up with compliance and regulatory changes. ACRN Journal of Finance and Risk Perspectives, Special Issue Digital Accounting, 8, pp.71-85.