International Journal of Management, IT & Engineering
Vol. 15 Issue 05, May 2025,
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gate as well as in Cabell's Directories of Publishing Opportunities, U.S.A

Domain Autonomy in Microservices Architecture: Real-time Data Sharing

Shashi Nath Kumar

Abstract

This paper addresses the complex challenge of managing source of truth and reference data in a Microservices Architecture, drawing upon Domain-Driven Design, Event-Driven Architecture, Command and Query Responsibility Segregation, and Data Mesh principles. I analyze the limitations of traditional database replication and the complexities introduced by distributed systems and propose a hybrid approach, combining log-based Change Data Capture with other modern Design Patterns, to achieve a balance between domain autonomy and efficient data sharing. The proposed approach minimizes database load, promotes loose coupling, and enables independent evolution of services while embracing domain ownership of data.The proposed implementation evaluates the performance, scalability, and trade-offs of this hybrid methodology. The implications of these findings are discussed for architects and developers building distributed bespoke business systems, offering insights into achieving domain autonomy, data consistency, and efficient data access in the evolving landscape of microservices architecture.

Keywords:

Domain-Driven Design;
Command and Query Responsibility Segregation;
Event-Driven Architecture;
Microservices Architecture;
Data Mesh Architecture;
Change Data Capture;
Inter Service Communication

Copyright © 2025 International Journals of Multidisciplinary Research Academy. *All rights reserved.*

Author correspondence:
Shashi Nath Kumar
Bachelor of Engineering (2007), Jadavpur University, Kolkata
Independent Researcher, Tampa, Florida, USA.
Email: email2snku@gmail.com

1. Introduction

The evolution towards microservices architectures has unlocked significant benefits in software development, enabling agility, scalability, and fault tolerance [1]. However, this shift introduces complexities in managing data consistency and accessibility across independent services, particularly when striving for domain autonomy within a Domain-Driven Design (DDD) paradigm [2,3]. This paper explores the complex challenge of maintaining a source of truth or authoritative source and providing efficient data access patterns in a microservices environment, focusing on a hybrid approach that bridges the gap between real-time operational data and the principles of Data Mesh architecture [4].

Traditional approaches to data sharing in microservices often rely on real-time API calls over HTTP [5] which creates tight coupling between services, increasing the risk of cascading failures and hindering independent deployments or introduces additional complex ecosystems both in synchronous and asynchronous inter-service communication. Furthermore, solutions like database replication, while effective for disaster recovery, often fall short in addressing the nuanced needs of inter-domain data integration in a microservices world. Pursuing a perfect solution (e.g., complete domain isolation) can hinder the delivery of practical value (e.g., efficient data sharing) [6]. Approaches like SQL Server Always On Availability Groups [8], while robust, often necessitate replicating entire databases, leading to data redundancy and potential violation of domain boundaries. This exemplifies O'Reilly's "Residue Theory," where design choices create unintended consequences [7].

This research embraces a pragmatic approach, acknowledging the complexities inherent in distributed systems as described by Complexity Science. I propose a hybrid model that leverages log-based Change Data Capture (CDC) and Event-Driven Architecture (EDA) to achieve a balance between domain autonomy and inter-service data sharing. By utilizing efficient log-based CDC tools, I aim to minimize the performance impact on source databases while ensuring selective data propagation. Furthermore, I incorporate CQRS

(Command Query Responsibility Segregation) to enable command-initiated data replication through events as asynchronous inter service communication while addressing the dual write or extended transaction conditions that propagate data changes to other domains in a heterogeneous structure.

This paper delves into the challenges of real-time data synchronization in distributed systems, where traditional transactional mechanisms are often impractical. I analyze specific data access patterns and inter-domain communication needs across various industries, highlighting the practicality and effectiveness of this hybrid methodology. I also discuss the trade-offs involved in different data synchronization strategies and provide insights into implementation details and optimization techniques.

2. Literature Review

The concept of a single source of truth has long been a cornerstone of data management. However, the rise of microservices, with its emphasis on domain boundaries and independent services, challenges this traditional centralized approach. DDD advocates for each domain to manage its own data within a bounded context, promoting autonomy and flexibility [2,3]. This can lead to data silos and hinder inter-service communication if not managed effectively.

While synchronous communication via API calls may be suitable for certain scenarios, it introduces runtime dependencies and tight coupling between services [11]. This can lead to increased latency, reduced resilience, and scaling dependency. To address some of these challenges, Service Mesh Architecture has been introduced which introduces further moving parts in an already complex ecosystem.

EDA offers a solution for inter-service communication by enabling asynchronous communication and loose coupling between services [9]. Changes in one domain are captured as events and propagated to other interested domains, facilitating data synchronization without tight dependencies. However, implementing EDA in a complex microservices landscape requires careful consideration of eventual data consistency and event handling [10]. The search for appropriations in EDA leads to the Event Sourcing pattern amplifying the complexity of the ecosystem.

The Asynchronous data products in Data Mesh architecture emerges as a compelling paradigm for addressing these challenges, particularly in analytical contexts . While primarily focused on analytical data, the principles of Data Mesh can be extended to operational data in a microservices environment, promoting decentralized data governance and self-serve data infrastructure. It is notable that Data Mesh Architecture itself has its roots in Microservices Architecture [1] and the same set of evolved technologies can be used in transactional workloads as well.

CDC has become a crucial technology for data synchronization, enabling the capture and propagation of changes in a database [13]. Tools like Debezium offer robust CDC capabilities but can introduce performance overhead on the source database (Confluent, n.d.). This research explores alternative CDC mechanisms, such as log-based replication offered by platforms like Fivetran, to minimize database load while ensuring efficient data extraction.

Data Mesh architecture, with its roots in microservices, can be applied to transactional workloads, enabling real-time data sharing while maintaining loose coupling between services. This approach promotes data quality, discoverability, and development agility, but requires careful consideration of data consistency and conflict resolution to ensure data integrity.

This literature review highlights the existing research on managing sources of truth and/or authoritative sources in distributed systems, focusing on the interplay between DDD, EDA, CDC, and Data Mesh principles. I identify the gaps in current approaches, particularly the limitations of traditional database replication and the challenges of excessive data replication, and propose a hybrid methodology that addresses these limitations while leveraging the strengths of each paradigm.

3. Research Method

3.1 Theoretical Framework: I draw upon the principles of DDD, EDA, Data Mesh, and Complexity Science to establish a theoretical framework for managing source of truth and reference data in a microservices environment as an alternative to synchronous inter service communication. We analyze the concepts of domain boundaries, bounded contexts, event-driven communication, data consistency, and decentralized data governance to guide our methodology.

3.2 DDD, Domain Autonomy, CQRS and EDA: Domain-Driven Design (DDD) is an architectural approach that aligns software design with the core business domain. It establishes a shared language between technical and business stakeholders and divides complex systems into bounded contexts. Each bounded context represents a subdomain with its own models, terminology, and responsibilities. A key principle of DDD is domain autonomy, which emphasizes the independence and self-governance of each bounded context. This means that each context has control over its data, logic, and communication with other contexts. Domain autonomy reduces dependencies and coupling between contexts, leading to improved agility, scalability, and maintainability of the overall system. CQRS is a design pattern that complements DDD by separating command processing (which changes system state) from query processing (which retrieves data). This separation allows for greater flexibility in optimizing the system for different workloads, such as high write or read volumes. CQRS can also simplify domain logic and improve system performance and scalability. Event-Driven Architecture (EDA) is a paradigm that focuses on producing, detecting, consuming, and reacting to events. In the context of DDD, EDA can be used to enable communication and collaboration between bounded contexts. By publishing events that signify important changes within a context, EDA allows other contexts to respond and adapt accordingly. This promotes loose coupling and asynchronous communication, which can enhance the responsiveness and resilience of the overall system. DDD, Domain Autonomy, CQRS, and EDA are complementary concepts that can be used to build robust, scalable, and maintainable software systems. By focusing on the business domain, promoting autonomy, separating concerns, and embracing event-driven communication, these approaches can help organizations deliver software that meets the changing needs of their users and stakeholders. A Port and Adapter Architecture also known as Hexagonal Architecture is apt for this combination. While utilizing modern queuing and data streaming platforms like Kafka when a consumer scalability is directly tied to number of partitions, the consumer and the Consumer/Async adapter must be deployed separately from the REST/Sync Adapter.

3.3 Sync Lookup and Service Mesh: While the synchronous data lookup pattern offers benefits within a microservices architecture, awareness of its potential detriments is important. This methodology introduces runtime dependencies, potentially causing unanticipated malfunctions. Moreover, it can foster strong inter-service coupling, thereby complicating independent modification or updating of services. Such close dependencies may also constrain the system's adaptability and extensibility, as alterations in one component can produce ripple effects throughout the architectural landscape. A Service Mesh through a sidecar pattern is one of the ways to address these couplings by offloading the cross cutting concerns to a sidecar in a Kubernetes environment however it creates additional cost and complexity in ecosystem management.

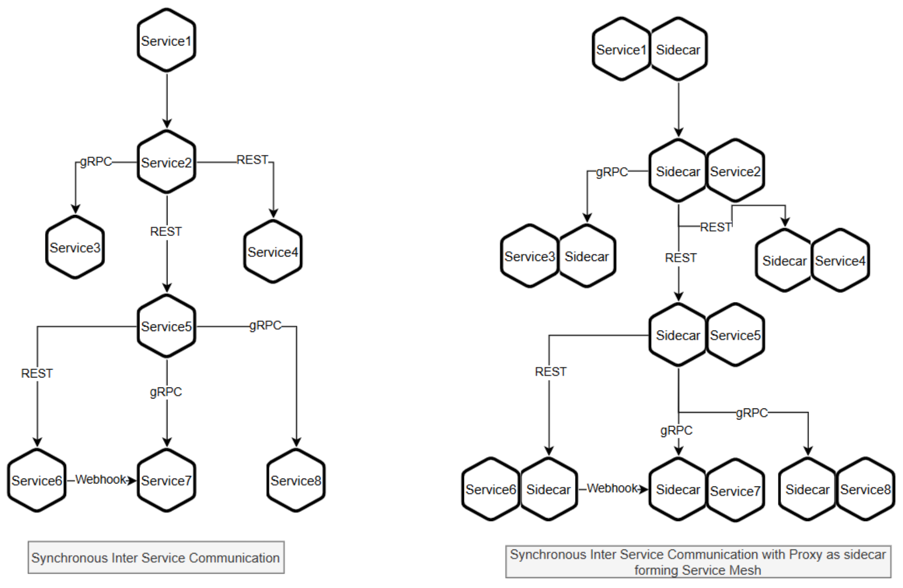

Figure 1. Synchronous Direct Inter Service Communication and Service Mesh

3.4 Async Data Propagation through Event Driven Architecture: EDA is a software design pattern that facilitates asynchronous communication between loosely coupled components or services. EDA uses events to represent important occurrences or changes within a system. Event producers publish these events, which are then consumed by event subscribers that may trigger additional actions or events. Due to its inherent decoupling and scalability, EDA is well-suited for asynchronous data propagation. Event producers can publish data changes as events without needing to know about the specific consumers or their processing requirements. This allows for flexibility and customization, as event subscribers can independently choose which events to subscribe to and how to process the data. By leveraging EDA, systems can achieve efficient and scalable asynchronous data propagation, where data changes are propagated as events in a decoupled and flexible manner. This approach enables real-time or near-real-time data updates, improves system responsiveness, and supports distributed and loosely coupled architectures. However in a Microservice environment with strong ties with Domain Logic, reliably producing Integration events and persisting the domain events (datastore) gets complicated and introduces challenges with runtime maintenance. Moreover, inter domain requirements and event carried state transfers makes the governance super complex job, effectively normalizing every event to a canonical message or rely on database lookups to enrich the domain events. Overall this pattern in itself makes this a data integration problem rather than an inter-domain communication challenge.

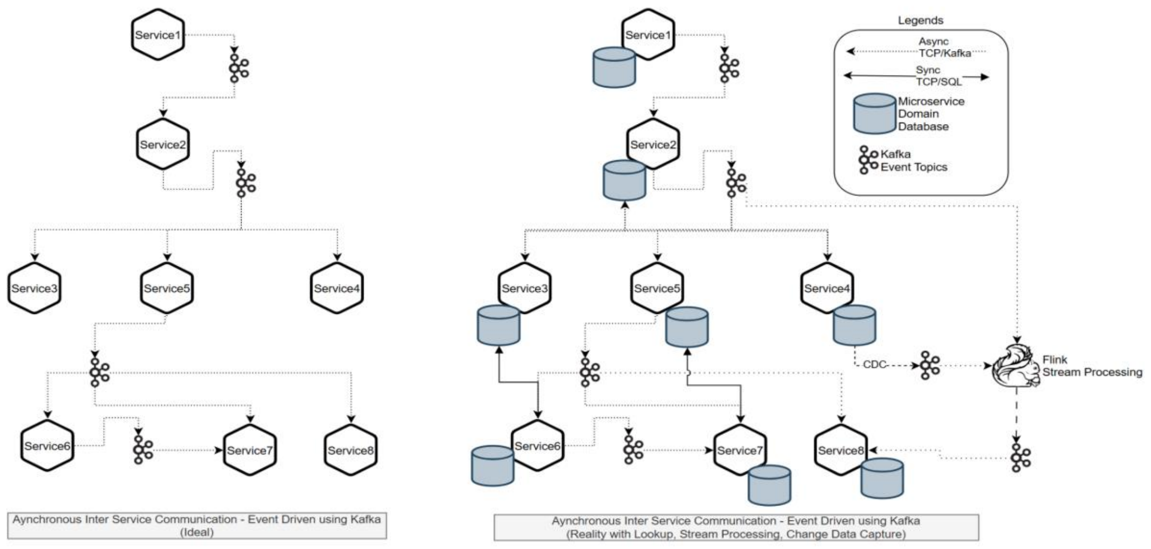
Service1
Service2
Service3
Service5
Service4
Service6
Service7
Service8
Aynchronous Inter Service Communication - Event Driven using Kafka (Ideal)
Legends
Async TCP/Kafka
Sync TCP/SQL
Microservice Domain Database
Kafka Event Topics
CDC
Flink Stream Processing
Aynchronous Inter Service Communication - Event Driven using Kafka (Reality with Lookup, Stream Processing, Change Data Capture)

Figure 2: The ideal EDA vs. the reality involving lookups/stream processing.

3.5 Database Replication through Availability Groups: SQL Server Availability Groups (AGs) let you replicate databases to keep read-only copies updated. They're great for high availability and disaster recovery, but they might not be the best fit for microservices. Things can get pretty tightly coupled, transactions can be tricky to keep consistent, scalability can hit some walls, and managing it all gets complicated. These issues can really mess with keeping domains separate and handling data in a microservices setup.

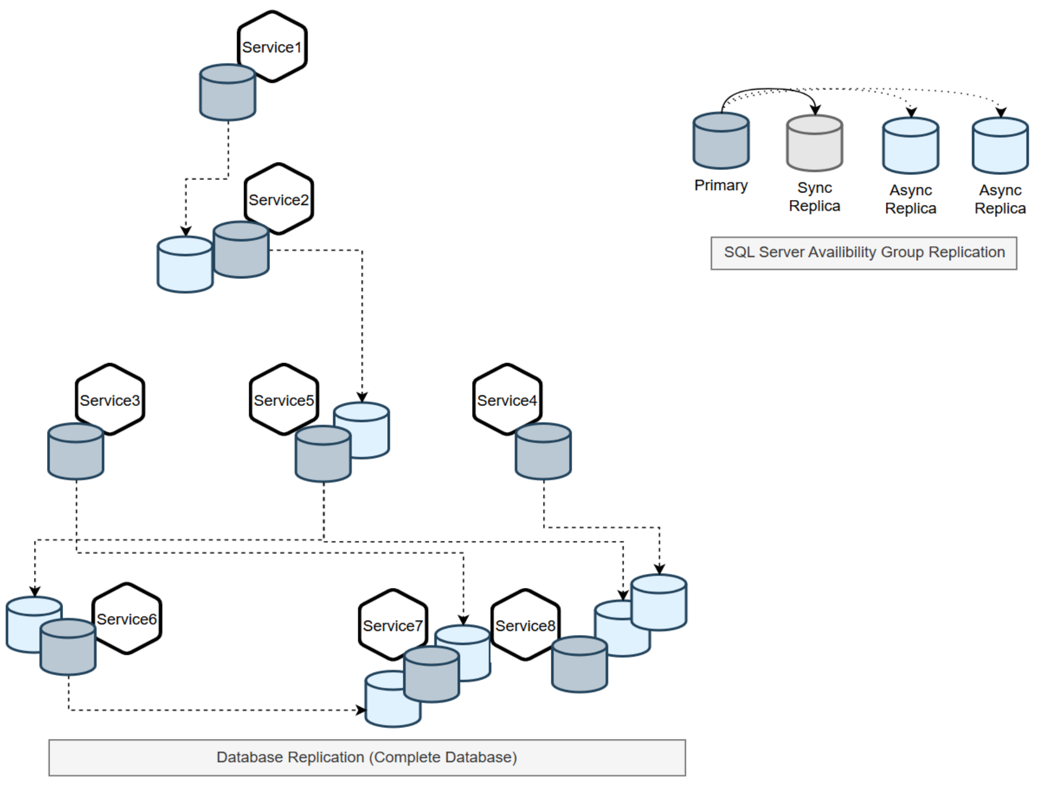

Figure 3: The Database Replication through Availability Groups.

3.6 Hybrid Approach: I propose a hybrid approach that combines log-based CDC with EDA, incorporating CQRS for command-initiated event origination and data replication. For reference data, we leverage efficient log-based CDC tools like Fivetran to replicate data from the source-of-record domain to consuming domains. For broader data synchronization, requiring a high change rate, we employ an event-driven approach using Kafka as the message broker, produced by commands in the source domain. This approach eradicates the need for complex event message enrichment with data lookup or stream joins as well. Establishing the Data Access pattern and the current technical landscape are key drivers of what information will be replicated vs what will be propagated in real time. Moreover, jump starting any project with a complex architecture as the golden target architecture always needs a roadmap to evolve the current state to the target state. Even if the target state architecture is completely event driven or Service Mesh, the hybrid approach will always help in coming up with the intermediate steps for the roadmap.

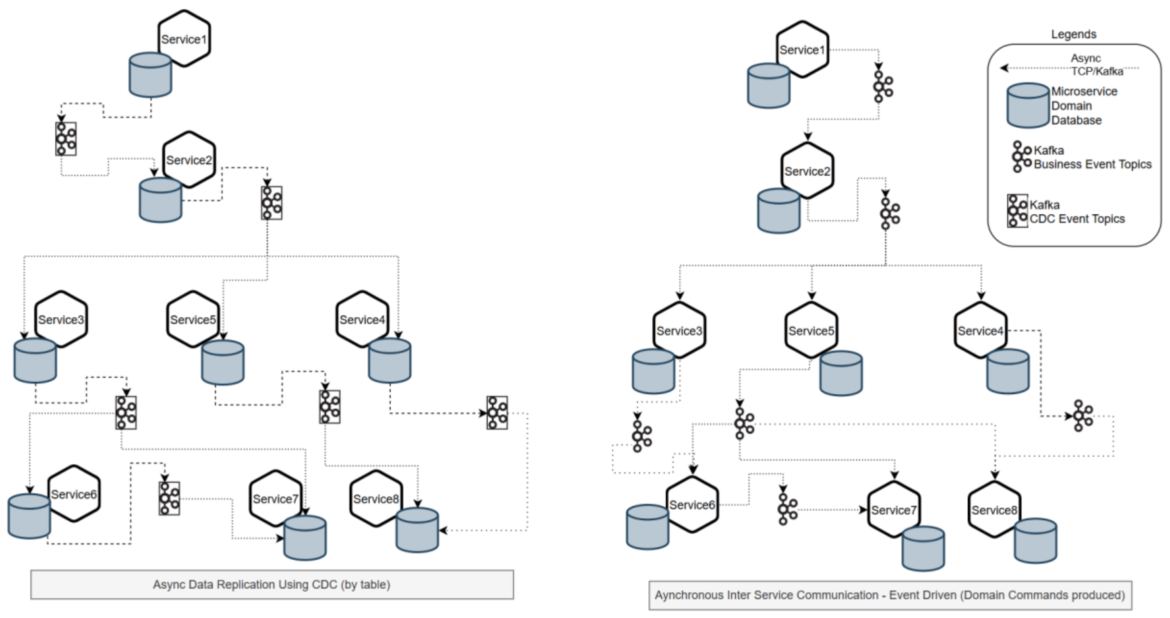
Service1
Service2
Service3
Service5
Service4
Service6
Service7
Service8
Async Data Replication Using CDC (by table)
Legends
Async TCP/Kafka
Microservice Domain Database
Kafka Business Event Topics
Kafka CDC Event Topics
Aynchronous Inter Service Communication - Event Driven (Domain Commands produced)

Figure 4: The proposed hybrid approach, combining CDC replication and event-driven communication initiated by domain commands.

3.7 Implementation: I implemented these architectures in multiple project environments with heterogeneous data structures across various industries. This allowed me to evaluate the performance, scalability, and consistency guarantees of all these approaches and encouraged me to consider the hybrid approach as the best suited approach.

3.7.1 Shipping/Logistics:

- **Scenario:** A shipment tracking service needs to display the current location of a package. This information is owned by the "Logistics" domain. The "Customer Service" domain also needs access to this location data to answer customer inquiries.
- **Data Access Pattern:** The "Customer Service" domain frequently performs read-only lookups of package location data. This is a high-volume, read-heavy access pattern.
- **Inter-Domain Communication:** For displaying the location on a tracking website, the "Customer Service" domain could use CDC to replicate the necessary data from the "Logistics" domain's database to its own read-optimized database. This allows for fast lookups without burdening the "Logistics" domain.
- **Event-Driven Need:** When a package's status changes (e.g., "in transit," "delivered"), the "Logistics" domain publishes an event to Kafka. Other domains, such as "Billing" (to update delivery status) or "Customer Facing Tracking" (to adjust shipment status), subscribe to this event and update their own systems accordingly. This is a true event-driven interaction for broader data synchronization.
- **Hybrid Justification:** Direct replication for the specific, high-volume lookup use case (package location) is more efficient than a full event-driven approach for every single lookup. The event-driven architecture is reserved for broader, less frequent data synchronization related to package status changes.

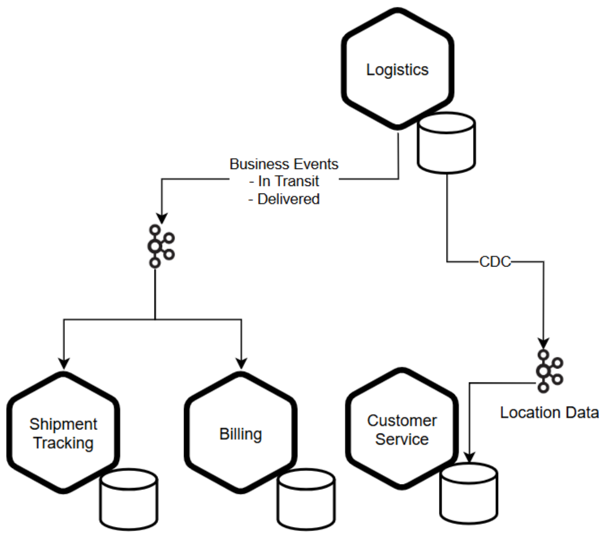

Figure 5: The proposed hybrid approach implementation in Shipping Induatry

3.7.2. Payment Processing:

- **Scenario:** A payment gateway must verify the validity of a customer's credit card and initiative authorization. The card details from “Customer Accounts” are needed in “Fraud Detection” domain as lookup data. The settled transaction details are needed to be in the "Settlement" and “Reconciliation” domain. The "Fraud Detection" domain needs access to each transaction data to assess the risk of a transaction.
- **Data Access Pattern:** The "Fraud Detection" domain needs to access specific card details (e.g., card number, expiry date) for each transaction. This is a high-volume, real-time access pattern. It also needs each transaction details from Payment gateway.
- **Inter-Domain Communication:** For real-time fraud checks, replicated lookup data is crucial. CDC could be used to replicate relevant card details to a read-optimized database accessible by the "Fraud Detection" domain. This minimizes transaction processing latency.
- **Event-Driven Need:** When a customer's card details are updated (e.g., new address), the "Customer Accounts" domain publishes an event. The "Fraud Detection" domain, and other relevant domains like "Billing," subscribe to this event to update their systems. And end of day settled transaction details can be sent to other domains. Each transaction details must be sent to “Fraud Detection Domain” at extremely low latency to complete authorization.
- **Hybrid Justification:** Real-time fraud checks require low-latency access to card details, making direct replication appropriate. However, broader updates related to customer accounts are handled via events.

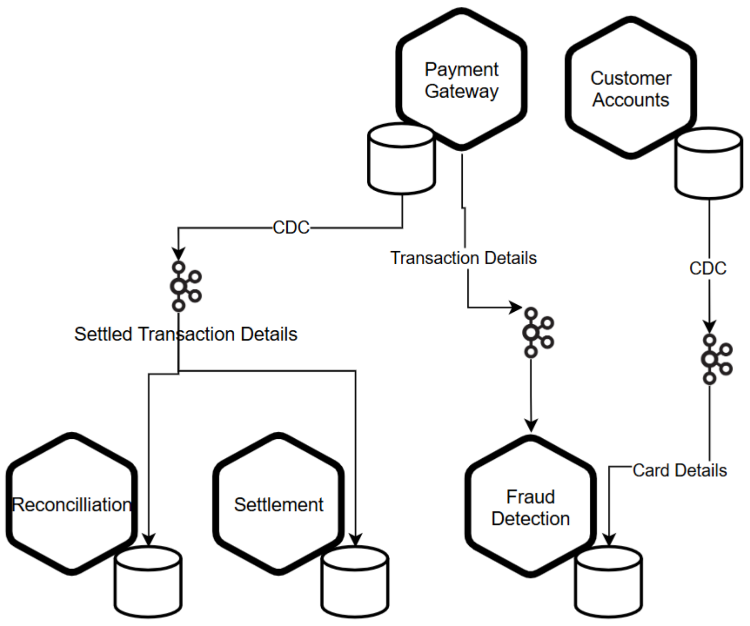

Figure 6: The proposed hybrid approach implementation in Payment Processing

3.7.3. Human Capital Management:

- **Scenario:** An employee "Payroll" system needs "Compensation", "Benefits" and "Time Management" info to accurately calculate payroll.
- **Data Access Pattern:** The "Payroll" domain frequently needs to access employee salary and Benefit data for every payroll calculation however these are fairly static. The time Management data is dynamic especially if it needs to work off of timecards.
- **Inter-Domain Communication:** Fivetran (or similar) can replicate the "Compensation" and "Benefits" data to Payroll domain.
- **Event-Driven Need:** When an employee's time card event occurs, the payroll must subscribe to timecard swipe events to accurately calculate payroll
- **Hybrid Justification:** Replication is suitable for the frequent but not real-time salary lookups needed by the "Payroll" system. Events are used for more frequent timecard updates.

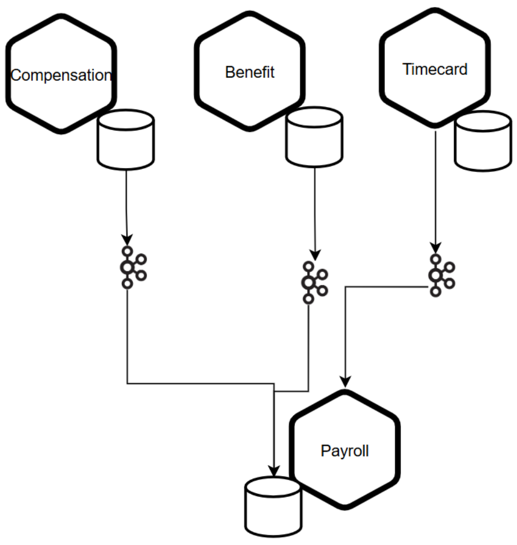

Figure 7: The proposed hybrid approach implementation in Human Capital Managment

3.7.4. Consumer Banking:

- **Scenario:** A mobile banking app needs to display the customer's account balance. This data is owned by the "Core Banking" domain. The "Mobile Banking" domain needs access to this information for every app refresh.
- **Data Access Pattern:** The "Mobile Banking" domain performs frequent, real-time lookups of account balance data.
- **Inter-Domain Communication:** Replicated read-only copy is essential. Fivetran or similar can replicate account balance data to the "Mobile Banking" domain's read database.
- **Event-Driven Need:** When a transaction occurs, the "Core Banking" domain publishes an event. Other domains, such as "Investment Management" (to update portfolio balances) or "General Ledger" (to update ledger), subscribe to these events.
- **Hybrid Justification:** Real-time balance display requires efficient lookup data replication. Broader transaction updates are handled via events.

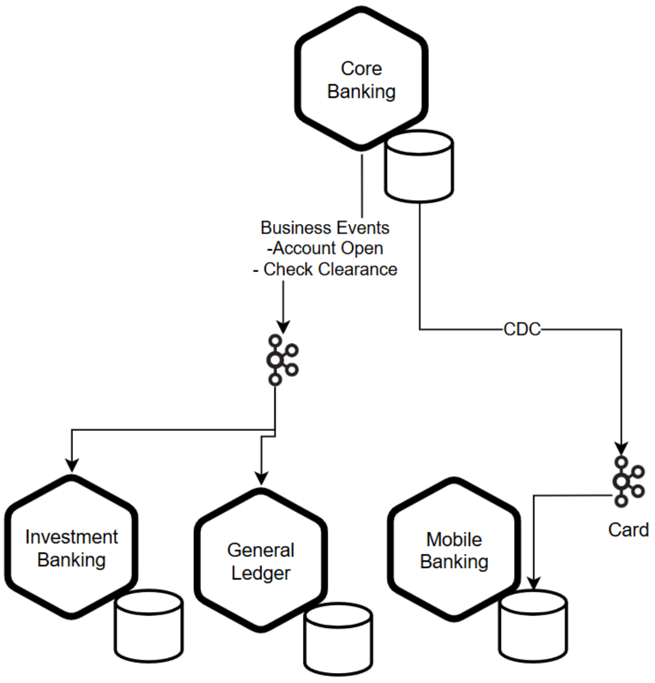

Figure 8: The proposed hybrid approach implementation in Banking

3.8 Wardley Mapping: We utilize Wardley Mapping to visualize the component evolution of the integration framework, analyzing the maturity of different technologies and their strategic implications. This helps us understand the current landscape and anticipate future trends in data management for microservices.

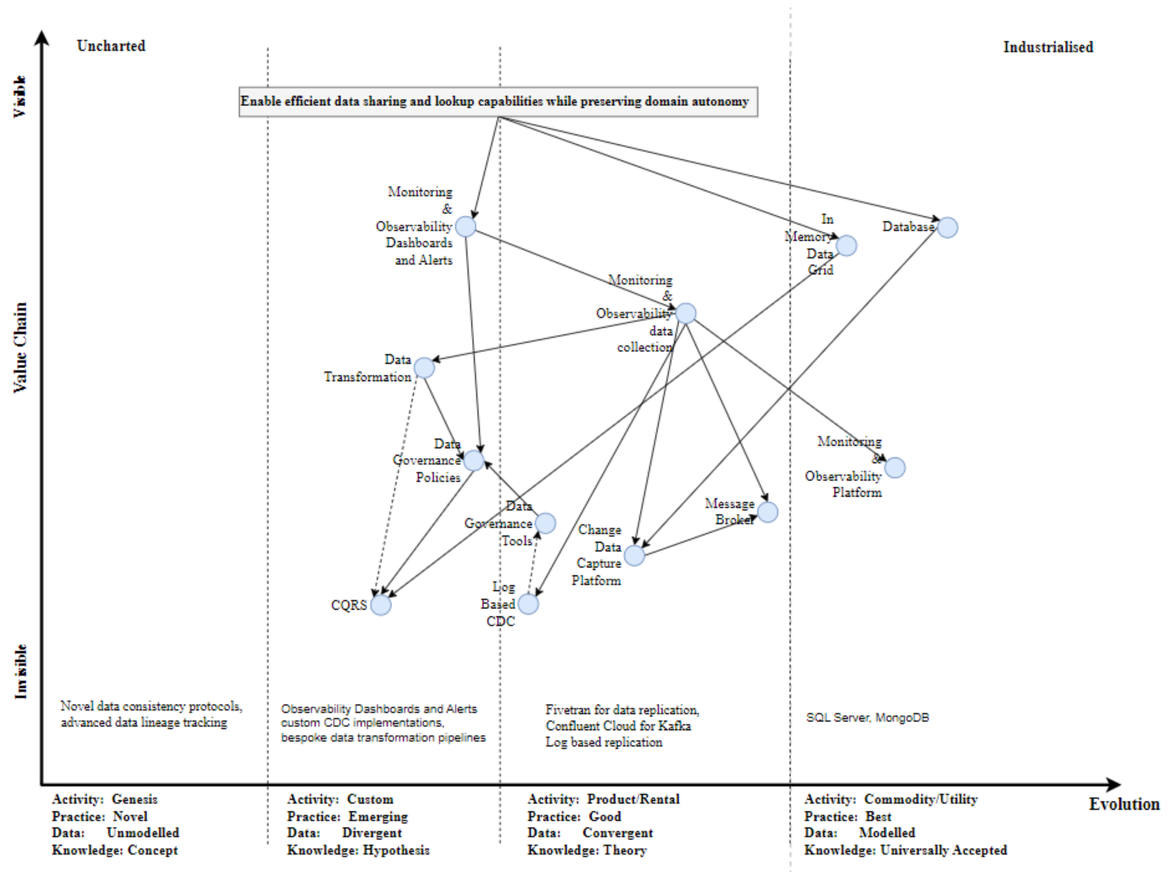
Uncharted
Industrialised
Visible
Value Chain
Invisible
Enable efficient data sharing and lookup capabilities while preserving domain autonomy
Monitoring & Observability Dashboards and Alerts
Monitoring & Observability data collection
In Memory Data Grid
Database
Data Transformation
Data Governance Policies
Data Governance Tools
Monitoring & Observability Platform
Message Broker
Change Data Capture Platform
CQRS
Log Based CDC
Novel data consistency protocols, advanced data lineage tracking
Observability Dashboards and Alerts custom CDC implementations, bespoke data transformation pipelines
Fivetran for data replication, Confluent Cloud for Kafka Log based replication
SQL Server, MongoDB
Evolution
Activity: Genesis
Practice: Novel
Data: Unmodelled
Knowledge: Concept
Activity: Custom
Practice: Emerging
Data: Divergent
Knowledge: Hypothesis
Activity: Product/Rental
Practice: Good
Data: Convergent
Knowledge: Theory
Activity: Commodity/Utility
Practice: Best
Data: Modelled
Knowledge: Universally Accepted

Figure 9: The evolution of technical stack in data sharing and lookup through Wardley Map

4. Results and Analysis

My implementation experience and evaluation of the hybrid approach yielded the following key results:

4.1 Reduced Database Load: Traditional CDC implementations often place a substantial burden on the source database due to the continuous polling or trigger-based mechanisms they employ to capture changes. This overhead can lead to performance bottlenecks, especially in systems that handle a large volume of transactions. Log-based CDC, on the other hand, leverages the database's transaction log, which is a built-in mechanism for recording all changes made to the data. By reading the transaction log, Log-based CDC can capture changes with minimal impact on the database's performance.

Furthermore, traditional CDC methods can sometimes introduce database version-specific dependencies, making it challenging to maintain compatibility across different database versions or migrate to new database platforms. Log-based CDC, by relying on the standardized transaction log format, often avoids such dependencies, ensuring greater flexibility and portability. Although each database platform has its own set of challenges to overcome, Fivetran Data Replication emerges as the winner compared to Debezium based replication.

4.2 Efficient Lookup Data Access: Replicating reference data across various consuming domains provided a significant advantage in terms of speed and efficiency for data retrieval. This streamlined process greatly enhanced the performance of operations that heavily relied on lookups, as the data was readily available within the domain itself. This eliminated the need for time-consuming cross-domain communication or data fetching, resulting in faster response times and improved overall system performance. This also gives an interim step to a target state architecture where event source based events are propagated asynchronously to all consuming domains.

4.3 Improved Domain Autonomy: In an event-driven architecture, domains are loosely coupled, which means they can operate and evolve independently without being tightly integrated. This loose coupling is achieved through the use of events as a communication mechanism. When a significant change occurs within a domain, it publishes an event. Although the determination of the payload (event notification, event carried state transfer or Event Sourced) introduces complexity, its value in just communicating the state change of objects in near real time fashion can not be undervalued. Other domains that have subscribed to that event can then react accordingly, ensuring that data remains consistent across the system. This approach fosters flexibility and maintainability, as changes within one domain do not necessitate widespread modifications in other domains. As long as the published events and their associated data structures remain compatible, the system can adapt to evolving requirements and technological advancements. AsyncAPI specification provides guidelines on various payload formats and broker protocol specific message binding that will be used to govern and evolve Async event contracts.

4.4 Scalability and Fault Tolerance:.The implementation of Kafka as the chosen message broker significantly enhanced the system's scalability and fault tolerance. This ensured that messages were reliably delivered and data remained synchronized across all services, even when faced with potential system failures or individual service outages. Kafka's inherent ability to handle high message volumes and its distributed architecture made it an ideal choice for managing the complexities of inter-domain communication.

4.5 Effective Data Synchronization with CQRS: In a system designed around the CQRS pattern, efficient event driven messaging was achieved by initiating the process with commands. These commands, upon execution, would produce corresponding events to the Kafka topic. These events were then responsible for propagating the changes made within one domain to other domains as the Kafka Consumer interface would trigger commands to establish the corresponding state in consuming domains, even when those domains utilized disparate data structures, ensuring data consistency across the entire system.

4.6 Trade-offs and Complexities: While the hybrid approach offered significant benefits, it also introduced some complexities in terms of managing different data synchronization mechanisms and ensuring data consistency across multiple data stores.

The results of this research demonstrate the viability and effectiveness of the hybrid approach for managing source of truth and refdata in a microservices architecture, incorporating Data Mesh principles and CQRS. By combining the strengths of log-based CDC, EDA, and domain-oriented data ownership, we achieved a balance between domain autonomy, data consistency, and performance.

However, the hybrid approach also introduces complexities that must be carefully managed. Ensuring data consistency across multiple data stores requires robust mechanisms for handling updates and resolving potential conflicts. Implementing and maintaining different data synchronization mechanisms can also add complexity to the overall architecture.

This research also highlights the limitations of traditional database replication techniques like SQL Server Always On Availability Groups, which are primarily designed for disaster recovery and often lead to excessive data replication. Our hybrid approach overcomes these limitations by enabling selective data propagation and promoting domain autonomy, as evidenced by the reduced database load and improved performance observed in our case study implementations.

Furthermore, this research explores the applicability of Data Mesh principles to operational data in a microservices context. By treating data as a product and promoting domain ownership, we can achieve a more decentralized and agile approach to data governance. This aligns with the broader trend of democratizing data access and empowering domains to manage their data independently.

Future research could explore more sophisticated techniques for managing data consistency and optimizing the performance of the hybrid approach. Further investigation into the trade-offs between different data synchronization strategies and their applicability to various use cases would also be valuable.

5. Conclusion

Research provides a valuable contribution to the field of microservices architecture, DDD, EDA, and Data Mesh by offering a pragmatic and nuanced approach to managing source of truth and lookup data for real-time operational and transactional needs. Our proposed hybrid methodology, combining log-based CDC, event-driven communication with CQRS, and domain-oriented data ownership, addresses the limitations of existing solutions while leveraging their strengths.

The findings of this research have significant implications for architects and developers building distributed systems. By adopting a hybrid approach and carefully considering the trade-offs involved, organizations can achieve domain autonomy, maintain data consistency, and optimize performance in their microservices architectures.

This research opens up new avenues for future exploration, including the development of more sophisticated data consistency mechanisms, the optimization of data synchronization strategies, and the application of this methodology to a wider range of industries and use cases. As microservices architectures continue to evolve, the need for effective data management solutions that embrace domain autonomy and real-time data sharing will only become more critical. This research provides a solid foundation for future innovation in this domain.

References

[1] Pautasso, C., Zimmermann, O., & Amundsen, M. “Microservices in Practice, Part 1: Reality Check and Service Design”, *IEEE Software, 34(1), 91-98.* 2017

[2] Evans, E., “Domain-Driven Design: Tackling Complexity in the Heart of Software”, *Pearson,* 2008

[3] Brandolini, A., *“*Strategic Domain-Driven Design with Context Mapping*”, In Proceedings of the 1st International Conference on Software Architecture Companion pp. 15-18. IEEE Press. 2019*

[4] Bellemare, A., *“*Building an Event-Driven Data Mesh”, *O'Reilly Media. 2023*

[5] Maia, T. and Correia, F., “Service Mesh Patterns”, *ACM EuroPLop '22: Proceedings of the 27th European Conference on Pattern Languages of Programs* (2), 1-12, *2022*

[6] *O'Reilly, B.,* “The Architect’s Paradox”, *#45, The complexity Lounge, https://youtu.be/Qq8x7KIV4W8?si=PRgIF4JezPQUsMbc* 2025

[7] O'Reilly, B. “Residuality and Representation: Toward a Coherent Philosophy of Software Architecture”, *Science Direct, Procedia Computer Science, 224, 91-97.* 2023

[8] Microsoft, “Always On Availability Groups”, *https://learn.microsoft.com/en-us/sql/database-engine/availability-groups/windows/always-on-availability-groups-sql-server?view=sql-server-ver16* , 2024

[9] Snowden, D. J., & Boone, M. E., “A Leader's Framework for Decision Making”. *Harvard Business Review, 85(11), 68-76.* 2007

[10] Devopedia, “Inter-Service Communication for Microservices”, https://devopedia.org/inter-service-communication-for-microservices, 2022.

[11] Richardson, C., “Microservices: Decomposing Applications for Deployability and Scalability”, *In Proceedings of the 38th International Conference on Software Engineering Companion (pp. 243-252). IEEE Press,* 2017.

[12] Kleppmann, M., “Designing Data-Intensive Applications”, *O'Reilly Media. 2017*

[13] Pathirana, S., & Perera, A., “A Survey of Change Data Capture Technology for Efficient Data Replication”, *Journal of Information Technology Review, 7(1), 1-18. 2015*

[14] Wardley, S., “Wardley Mapping”. *https://learnwardleymapping.com/* 2014

[15] Confluent, “Sync Databases and Remove Silos with Kafka CDC”, *https://www.confluent.io/blog/sync-databases-and-remove-silos-with-kafka-cdc/*

[16] Vettor, Robert & Smith, Steve, “Cloud-Native .NET apps for Azure v1.0”, *https://dotnet.microsoft.com/download/e-book/cloud-native-azure/pdf*