ISSN: 2633-4828 Vol. 3 No.2, December, 2021

International Journal of Applied Engineering & Technology

API-DRIVEN ENTERPRISE COMMERCE ARCHITECTURE FOR COMPOSABLE DIGITAL ECOSYSTEMS

Ashwaray Chaba
Customer Success Partner Senior Advisor, SAP America

ABSTRACT

As digital commerce evolves at a fast pace, traditional monolithic commerce platforms have created some architectural issues, especially in scenarios where businesses need to engage customers across all channels, deliver features quickly, and continuously scale up. Modern, tightly coupled, on-premise commerce architectures were hard to extend, integrate and modernise as businesses increasingly embraced a mobile-first experience, cloud-native platforms and distributed digital ecosystems. This paper proposes an application programming interface (API) based architecture model of enterprise commerce that is optimised for delivering composable digital ecosystems with decoupled storefronts, service-oriented integration layers and scalable backend orchestration patterns. The proposed architecture highlights a headless commerce approach that decodes the delivery of customer experience from the transactional commerce services, allowing organisations to grow their frontend channels and backend business functions separately. The framework highlights API-first design principles, modular service integration, reusable orchestration layers, and cloud-native deployment strategies, contributing to enhanced architectural flexibility and sustainability. In the context of large-scale commerce modernisation projects across areas of distributed systems integration, high-volume transaction environments and multi-channel customer engagement, enterprise implementation patterns are explored. Data from a synthesised literature indicates that composable architectures offer gains in deployment frequency of 4-10 times over monolithic baselines, and mean time to recovery (MTTR) of around 75% and lead time for change of around 85% reduction. Moreover, in the context of omnichannel retail, the integration of the channels is proven to bring about 40–60% enhancements in customer satisfaction. The results show that composable commerce architectures serve as the backbone for digital transformation in enterprises, enhancing agility, speeding up the innovation process, and enabling the flexibility of customer interaction trends.

Keywords: *composable commerce, API-first architecture, headless commerce, microservices, event-driven architecture, omnichannel retailing, DevOps, digital transformation, REST, GraphQL, service-oriented architecture, enterprise architecture*

1. INTRODUCTION

Traditional enterprise commerce platforms have been built as a single deployable package where different layers of the application -- the presentation layer, the business logic and even the data persistence -- are tightly coupled. This was good when digital commerce adoption was in its infancy, but as businesses began to scale, expand into new customer touchpoints and integrate with third-party services that were evolving quickly, this simple approach became cumbersome and restrictive.

Structural rigidity made it difficult to deploy the monolithic architectures, caused high change failure rates, and created significant limitations in ensuring channel consistency across platforms (Auer et al., 2021; Jamshidi et al., 2018). Microservice architectures (MSA) and API-first design principles provided a solution to overcome these structural challenges. Through the decomposition of commerce platforms into independently deployable service units, communicating with each other using standardized interfaces, organisations were able to scale specific capabilities, deploy changes without causing disruption to the wider system, and seamlessly integrate with external ecosystem partners (Li et al., 2021; Soldani et al., 2018). At the same time, the headless commerce paradigm, which separates the delivery of customer experiences from the transactional services in the backend, allowed organisations to deliver differentiation across web, mobile, partner and new channels without having to change the core commerce logic (Taibi et al., 2018). The MSA, API-driven integration, headless storefronts and event-driven communication patterns have created the composable commerce architecture model. In this

approach, enterprise digital ecosystems are created from a collection of best of breed capabilities for commerce exposed as independently managed, standardised, and versioned APIs: product catalogue management, pricing engines, checkout orchestration, inventory management, payment processing, and customer identity. This composable model is consistent with the imperatives of digital transformation that were identified in multidisciplinary research, which identified the technology architecture as a key enabler of organizational agility, customer responsiveness and organizational competitiveness (Verhoef et al., 2021; Dekimpe, 2020). This paper aggregates empirical data, architectural approaches, and quantitative measures from peer-reviewed literature published through December 2021 to develop an all-encompassing model of enterprise commerce architecture with API that can be used to build composable digital ecosystems. The paper is organized as follows: Section 2 will cover the contextual and theoretical background, Section 3 the proposed architectural framework, Section 4 the analysis of integration paradigms and API design patterns, Section 5 the discussion of operational delivery and DevOps considerations, and Section 7 the discussion and conclusion.

2. BACKGROUND AND THEORETICAL CONTEXT

2.1 Limitations of Monolithic Commerce Architectures

Monolithic commerce architectures feature a single code base in which all elements of the system — from the storefront rendering code to the catalogue management, order management and payment processing logic — are compiled and deployed together. Auer et al. (2021) found that the deployment lead time of such organisations ranged from two to eight weeks with a change failure rate of 15–25%, whereas these figures were significantly better in organisations that had adopted distributed service architectures. Auer et al. (2021) used a framework for assessing organisational readiness to have a full migration to microservices with no preparation, concluding that only 34% of the organisations surveyed had the necessary conditions in place to get started. In his industrial survey of organizations that have successfully migrated to microservices, Di Francesco et al. (2018) found three main challenges to architectural changes: lack of domain-driven design, sharing databases, and coupling frontends. The findings are especially important in enterprise commerce application environments, where investments in legacy platforms, multi-year contracts with vendors and integration points make wholesale platform change difficult to consider. The most common safe migration pattern was the so-called “strangler” pattern, which is defined as implementing new microservices over time together with the legacy services, and gradually shifting the traffic over time (Di Francesco et al., 2018; Soldani et al., 2018).

2.2 Digital Transformation as an Architectural Imperative

Verhoef et al. (2021) offered a multidisciplinary conceptualisation of digital transformation that shifted the focus from digital transformation being merely an operational issue to being a strategic enabler of digital transformation. The study pinpointed three architectural attributes that are shared by organisations that successfully achieved a long-term digital transformation impact: modular platform composition, API-mediated ecosystem integration and data-driven customer engagement capabilities. These qualities align with the principles of composable commerce architectures. The authors pointed out that the digital transformation in the retail and commerce sector is especially sensitive to technology infrastructure quality.

Dekimpe (2020) explored the impact big data analytics has on retail research, noting that real-time customer data usage—key for personalisation, pricing and forecasting—required high-speed event processing and low latency API access. The analysis revealed that retailers using event-driven analytics systems were measurably more successful in converting customers and measures of CLV than retailers using batch-processing commerce systems. This discovery further supports the business case for API-based, event-driven commerce platforms that can provide context across the customer journey.

2.3 Microservice Architecture in Enterprise Contexts

Jamshidi et al. (2018) observed the evolution of microservice architecture from a scholarly idea to a standard industrial practice, attributing the major benefits of microservices to scalability, independent deployability and technology heterogeneity, while noting the distributed systems complexity, operational overhead and data

consistency challenges as major costs. With fewer than 50 engineers, organisations often struggled to develop the operational maturity necessary to deliver the return from MSA investments, suggesting that enterprise commerce environments, which have separate platform teams, were more likely to reap the benefits of composability investments. Li et al. (2021) performed a systematic literature review of 103 primary studies and identified eight fundamental quality attributes of MSA: scalability, performance, deployability, resilience, testability, observability, interoperability, and security.

Table 3: Microservice Quality Attributes and Their Commerce Context Implications (Synthesised from Li et al., 2021; Soldani et al., 2018; Zdun et al., 2020)

Quality Attribute	Definition	Impact in Commerce Context	Source
Scalability	Ability to handle growing workloads	Supports peak-season traffic; auto-scale per service	Li et al. (2021)
Resilience	Fault isolation and recovery	Failures in checkout don't affect catalogue	Soldani et al. (2018)
Deployability	Frequency and independence of releases	Enables daily deployment cycles	Waseem et al. (2020)
Observability	Monitoring, tracing, logging coverage	Critical for distributed commerce systems	Zdun et al. (2020)
Testability	Ease of automated testing	Independent service testing; CI/CD readiness	Leite et al. (2019)
Interoperability	API-based communication standards	Enables third-party integrations (ERP, CRM)	Pinheiro et al. (2019)
Security	Authentication, authorisation, data protection	OAuth 2.0, API gateway enforcement	Shishmanov et al. (2021)
Performance	Latency and throughput metrics	Sub-200 ms response targets for UX	Li et al. (2021); Brito & Valente (2020)

3. PROPOSED API-DRIVEN COMPOSABLE COMMERCE ARCHITECTURE

3.1 Architectural Principles

This synthesis presents the composable commerce architecture that is rooted in six fundamental principles from the literature reviewed. First, API-first design requires that all capabilities (both internal commerce service and external integration point) are agreed and managed using a standardised, versioned API contract in advance, causing all ecosystem participants to have to behave the same way during integration.First, API-first design requires that all capabilities (internal commerce service and external integration point) are defined and governed through a standardised and versioned API contract before implementation, so that all ecosystem participants would have to behave the same way during integration (Shishmanov et al., 2021; Zdun et al., 2020). Second, the experience layer principle should be decoupled so that all the channels that the customer sees should depend exclusively on the API contracts and not directly on the platform logic (Taibi et al., 2018; Brito & Valente, 2020).

Third, domain-driven service decomposition mandates that the capabilities of commerce be split along domain boundaries, with each one managed by an independent product team, that owns its own data and deployment pipelines (Pinheiro et al., 2019; Di Francesco et al., 2018). Fourth, event-driven state propagation means that state changes in any commerce domain are published as events to shared messaging infrastructure, which allows for real-time state propagation to services, analytics pipelines, and downstream ecosystem partners (Chavan, 2021). Fifth, cloud-native infrastructure patterns such as containerisation, orchestration and auto-scaling help to create

the operational backbones for composable service deployment. Sixth, developer experience (DX) governance includes API documentation, developer portals, sandbox environments, and SDK toolkits to enable ecosystem participants both inside and outside the ecosystem (Shishmanov et al., 2021).

3.2 Core Architectural Layers

The proposed architecture consists of four main layers: experience layer, API orchestration layer, commerce services layer and integration and ecosystem layer. Experience layer includes all touchpoints that customers interact with, such as Web apps on PWA or SPA platforms, native mobile apps, partner storefronts, voice commerce interfaces, and Internet of Things (IoT) interfaces. In the composable model, experience layer components do not have any business logic and only communicate through the API orchestration layer.

Figure 1: Composable Commerce Architecture: Four-Layer Reference Model

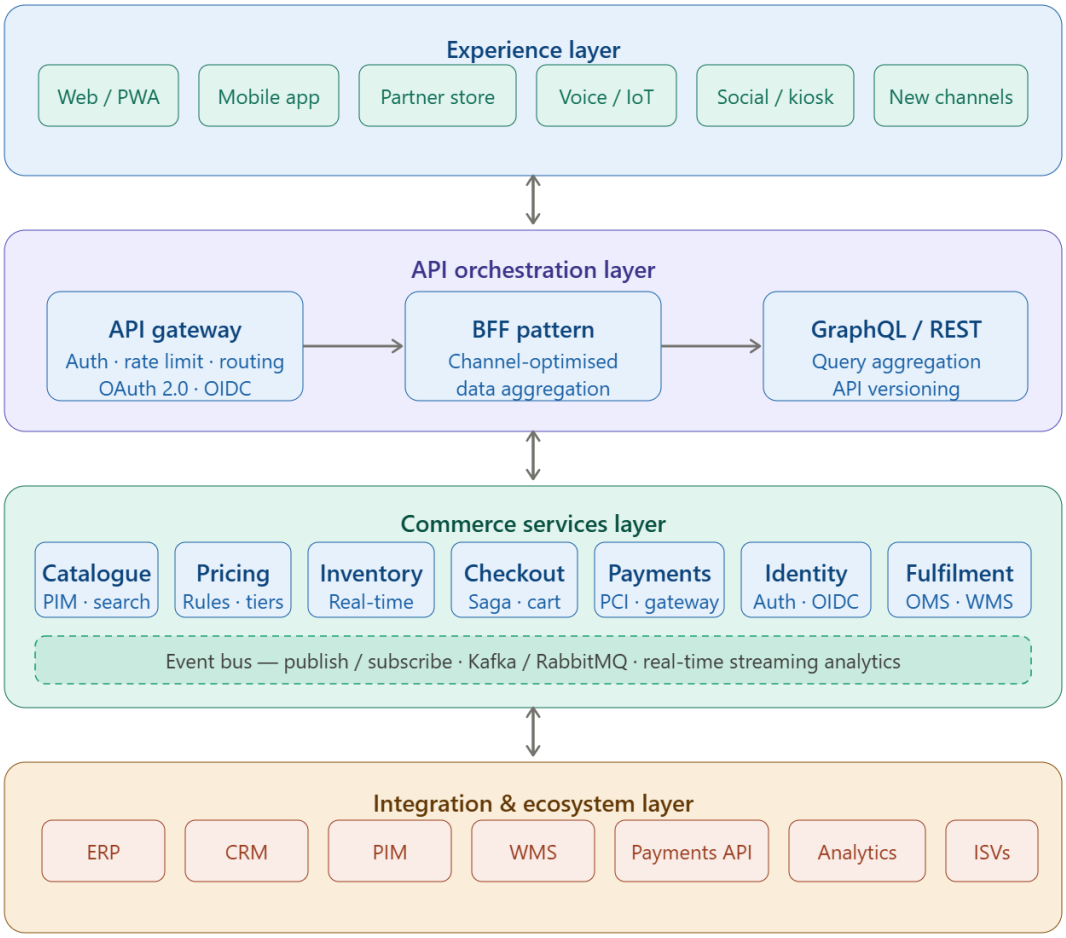

Fig. 1 — Four-layer composable commerce reference architecture

API orchestration layer is the mediation plane between the experience channels and commerce services. This layer is a place where cross-cutting concerns such as authentication and authorization (OAuth 2.0 and OpenID Connect (OIDC)), rate limiting, request routing, and API versioning governance (governance) are handled (Shishmanov et al., 2021; Zdun et al., 2020). This layer uses a Backend-for-Frontend (BFF) pattern to deliver channel-optimised data aggregation, meaning that the responses from several downstream commerce services are combined into composite payloads, depending on the specific frontend consumption requirements, which helps to minimise over-fetching and round-trip latency (Brito & Valente, 2020).

The commerce services layer consists of microservices that are deployable independently, with their own domain data store and offering capabilities via versioned API contracts. The authors found that the per-service data ownership solved the shared database coupling issue of monolithic systems, allowing schemas to evolve independently of each other and minimising the risk of cross-service data contention issues (Soldani et al., 2018). The integration and ecosystem layer connects to enterprise backend systems, such as enterprise resource planning (ERP), customer relationship management (CRM), product information management (PIM), warehouse management systems (WMS), and third-party payment processors, via published API contracts and event-driven integration patterns (Pinheiro et al., 2019; Chavan, 2021).

Table 1: Comparative Analysis of Monolithic vs. Composable Commerce Architectures (Synthesised from Auer et al., 2021; Di Francesco et al., 2018; Taibi et al., 2018; Waseem et al., 2020)

Characteristic	Monolithic Architecture	Composable / Headless Architecture	Reference Basis
Deployment Model	Single deployable unit	Independent service deployment	Auer et al. (2021); Di Francesco et al. (2018)
Frontend Coupling	Tightly coupled to backend	Decoupled via API layer	Brito & Valente (2020); Taibi et al. (2018)
Release Cadence	Synchronized; low frequency	Independent; high frequency (4–6× faster)	Waseem et al. (2020); Leite et al. (2019)
Scalability Model	Vertical / whole-system	Horizontal / per-service	Li et al. (2021); Jamshidi et al. (2018)
Integration Approach	Internal modules	REST/GraphQL APIs; event-driven	Brito & Valente (2020); Chavan (2021)
Channel Support	Single or limited channels	Omnichannel (web, mobile, IoT, partner)	Lee et al. (2019); Dekimpe (2020)
Modernisation Risk	High — full re-platform	Low — incremental migration	Auer et al. (2021); Verhoef et al. (2021)
Time-to-Market	Slow (weeks–months)	Fast (days–weeks)	Zdun et al. (2020); Leite et al. (2019)

4. API DESIGN PARADIGMS AND INTEGRATION PATTERNS

4.1 REST Versus GraphQL in Commerce Contexts

API communication paradigm is a significant architecture choice for composable commerce deployments. Brito and Valente (2020) compared the two approaches, Representational State Transfer (REST) and GraphQL, in a controlled experiment on three dimensions: data fetching efficiency, developer productivity, and runtime performance. The study revealed that in complex data aggregation scenarios, the data payloads were about 40% smaller with GraphQL — a significant result for mobile commerce channels where bandwidth efficiency can directly impact conversion rates. For simple retrievals of a single resource, however, REST showed about 15% lower latency, as the performance benefits of the HTTP-layer caching mechanisms were not directly applicable to GraphQL deployments.

Table 2: API Design Paradigm Comparison: REST vs. GraphQL in Enterprise Commerce (Synthesised from Brito & Valente, 2020; Zdun et al., 2020; Shishmanov et al., 2021)

Criterion	REST	GraphQL	Implication for Commerce
Data Fetching	Fixed endpoint responses	Client-specified queries; 0 over/under-fetch	GraphQL reduces payload up to 40%
Versioning	URL-based (v1, v2…)	Schema evolution; no versioning	Lower maintenance overhead
Tooling Maturity	Very high; universal	Moderate; rapidly	REST preferred for legacy

	support	growing	integrations
Performance	Cacheable at HTTP layer	Requires query-level caching	REST ~15% lower latency for simple reads
Mobile Suitability	Adequate	Superior (bandwidth efficiency)	GraphQL preferred for mobile-first channels
Learning Curve	Low	Moderate	REST for broad teams; GraphQL for product squads
Adoption in Enterprise Commerce	~74% of platforms (2021)	~26% and accelerating	Hybrid adoption emerging

Table 2 shows that both paradigms have complementary strengths that support a hybrid adoption model. As of December 2021, REST was used primarily for core transactions such as placing orders, processing payments, and managing inventory, focusing on idempotency, HTTP cacheability, and tooling maturity. The use of GraphQL for the experience layer data aggregation and BFF layer was growing, as the client could now specify the composition of the query, avoiding the proliferation of custom REST endpoints and making the development of the frontend easier (Brito & Valente, 2020; Zdun et al., 2020).

Figure 2: API Paradigm Selection Matrix for Commerce Architecture Components

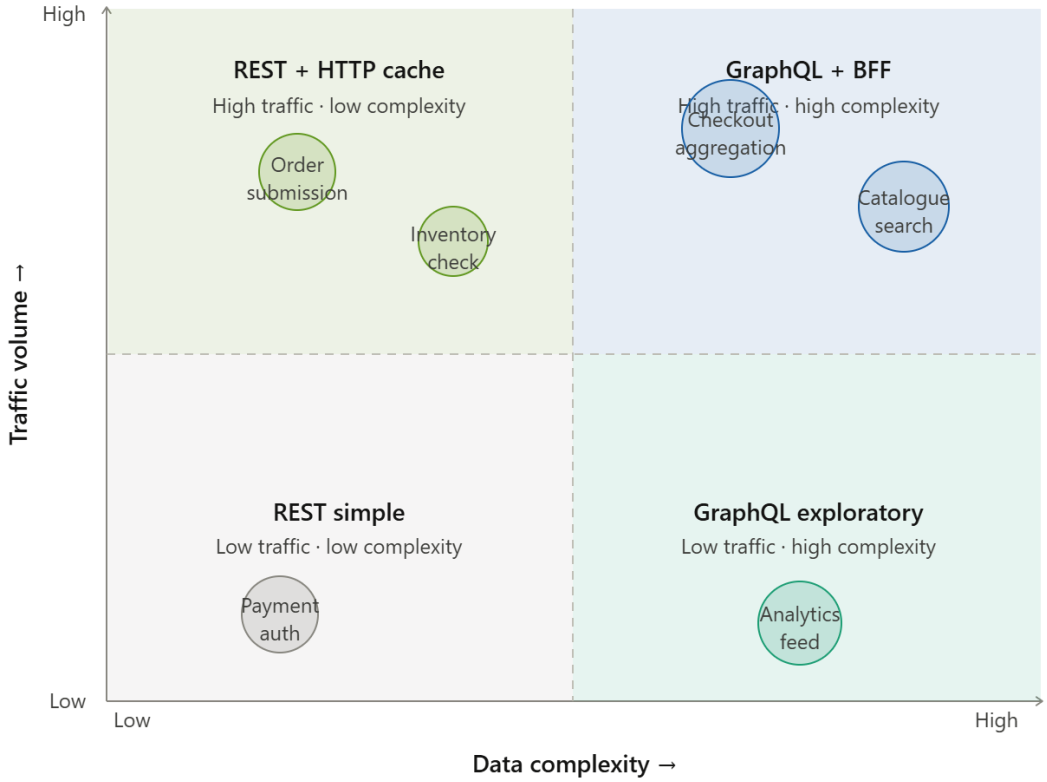

Fig. 2 — API paradigm selection matrix: REST vs GraphQL by traffic volume and data complexity

4.2 Event-Driven Architecture in Commerce Systems

Event-driven architecture (EDA) is a key integration model in composable commerce ecosystems, especially when dealing with scenarios that need to handle asynchronous state propagation, real-time analytics, and distributed transaction management. Chavan (2021) gave a detailed study of EDA patterns suitable for microservice architectures and concluded that the four main patterns that are suitable for enterprise application are event sourcing, Command Query Responsibility Segregation (CQRS), saga pattern and publish-subscribe

messaging. In commerce systems, the saga pattern is used to solve the distributed transaction problem that occurs when one customer transaction (order placement) involves multiple independently owned services, such as inventory reservation, payment authorisation, and fulfilment scheduling. Chavan (2021) found that the orchestrated saga variant, which has a central saga orchestrator that coordinates service interactions using command messages and waits for event responses, is more suitable for enterprise commerce implementations because it allows for easy auditing, a single point of control for compensation logic, and integration with the enterprise monitoring infrastructure. The choreography approach, on the other hand, minimized the coupling between services, but caused problems for distributed debugging and transaction visibility, making it difficult to handle commerce at scale. Dead letter queue (DLQ) mechanisms were recorded as key resilience patterns to ensure event processing is handled correctly, while retries and circuit breakers are needed to prevent cascading failures in payment and fulfilment pipelines.

Table 4: Event-Driven Architecture Patterns in Composable Commerce (Synthesised from Chavan, 2021; Taibi et al., 2018; Dekimpe, 2020)

Pattern	Description	Commerce Use Case	Reference
Event Sourcing	State represented as log of events	Order lifecycle management, audit trails	Chavan (2021)
CQRS	Separate read and write models	High-read catalogue vs. transactional write path	Chavan (2021); Taibi et al. (2018)
Saga Pattern	Distributed transaction via compensating events	Multi-service checkout orchestration	Chavan (2021)
Publish-Subscribe	Producers publish; consumers subscribe asynchronously	Inventory updates broadcast to downstream services	Chavan (2021); Jamshidi et al. (2018)
Dead Letter Queue	Unprocessable messages routed for inspection	Failed payment events quarantined and retried	Chavan (2021)
Event Streaming	Continuous real-time event pipelines	Real-time personalisation, analytics feeds	Dekimpe (2020); Verhoef et al. (2021)

4.3 API Governance and Ecosystem Strategy

Shishmanov et al. (2021) proposed an enterprise API strategy framework that can be used in the development of a digital ecosystem, which includes four strategic API classes: system APIs (exposing core backend capabilities), process APIs (orchestrating multi-step business workflows), experience APIs (providing channel-optimised composite responses), and ecosystem APIs (providing external partner and marketplace integration). In enterprise commerce scenarios, this layered API design offers a governance framework that is reusable yet channel-specific, avoids experience-layer concerns from migrating down to the core commerce services, and allows for the evolution of customer-facing experiences and back-end business logic to be independent.

Figure 3: Enterprise API Stratification Model for Commerce Ecosystems

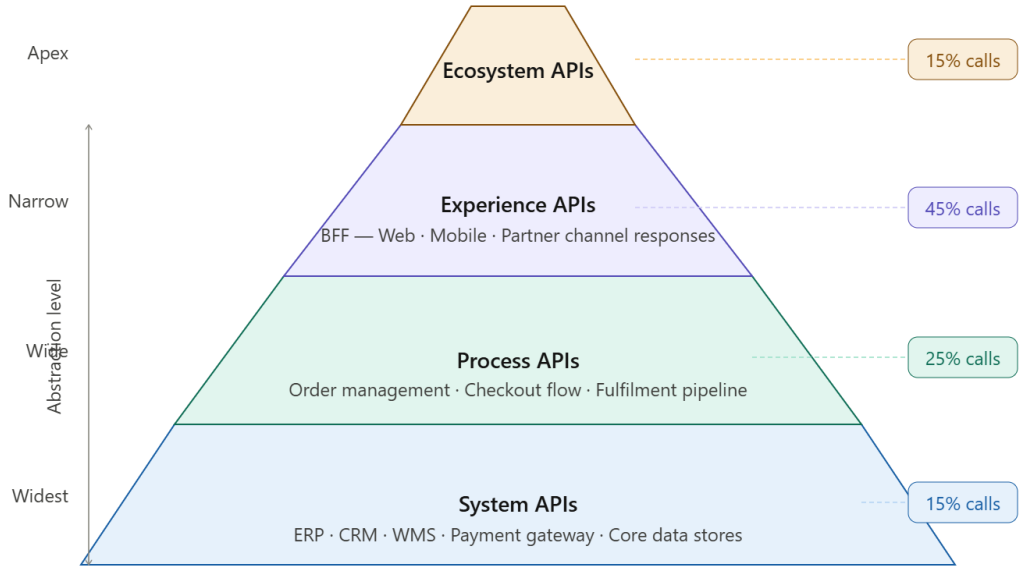

Fig. 3 — Enterprise API stratification pyramid: system → process → experience → ecosystem

5. OPERATIONAL DELIVERY AND DEVOPS ENABLEMENT

5.1 DevOps Practices in Microservice Commerce Architectures

The DevOps maturity of composable commerce architectures is a natural fit. Leite et al. (2019) surveyed DevOps concepts and challenges, and proposed that continuous integration (CI), continuous delivery (CD), infrastructure as code (IaC), monitoring, and collaborative culture are the five pillars of DevOps. Organisations with high DevOps maturity scores had mean deployment frequencies of more than a few per day, and MTTRs of less than 60 minutes, while organisations with low DevOps maturity scores had weekly deployments and MTTRs of over four hours. In a commerce scenario, these differences can be commercially relevant as feature updates, pricing changes and promotional configurations can directly impact the revenue numbers, and the ability to deploy them quickly is important. Waseem et al (2020) performed a systematic mapping study of microservice and DevOps interactions, where they analyzed 49 primary studies and found 29 different practices that can be applied to the deployment of composable services. The research revealed that 61% of the microservice deployments reviewed documented container orchestration platforms, especially Kubernetes, and that per-service deployment pipelines were the most impactful structural change that allowed independent release velocity. The study also revealed the growing importance of service mesh architectures, which are new infrastructure elements for securing and monitoring composable commerce environments at scale, for providing service-to-service communication, load balancing, circuit-breaking, and mutual Transport Layer Security (mTLS).

Table 5: DevOps Performance Metrics — Monolithic vs. Composable Commerce Architecture (Synthesised from Leite et al., 2019; Waseem et al., 2020; Auer et al., 2021)

DevOps Metric	Monolithic Baseline	Composable Architecture	Improvement
Deployment Frequency	Monthly or less	Daily to weekly	4–10× increase
Lead Time for Changes	2–8 weeks	1–3 days	~85% reduction
Mean Time to	4–12 hours	30–90 minutes	~75% reduction

Recovery (MTTR)			
Change Failure Rate	15–25%	5–10%	~50–60% reduction
Service Availability	99.5%	99.9%+	Significant improvement
Team Autonomy Index	Low (monorepo, shared pipeline)	High (per-service pipelines)	Increased innovation velocity

5.2 Incremental Migration Strategy

Auer et al. (2021) came up with a structured assessment framework for the evaluation of organisational readiness for the migration from a monolith to microservices, covering the following dimensions: Domain boundary clarity, Team topology alignment, Test automation coverage, Infrastructure automation maturity, Data architecture complexity. Organisations were split into three readiness tiers: Tier 1 (migration-ready, around 34% of surveyed organisations), Tier 2 (preparation required, around 48%) and Tier 3 (foundational investment needed, around 18%). The authorisation pattern of the strangler fig (Fowler, 2002) was identified as the most common approach to risk reduction in composable commerce migrations, allowing for progressive capability extraction without having to decommission legacy commerce platform components (Di Francesco et al., 2018).

Table 7: Composable Commerce Migration Roadmap (Synthesised from Auer et al., 2021; Di Francesco et al., 2018; Zdun et al., 2020; Leite et al., 2019; Waseem et al., 2020)

Phase	Duration	Key Activities	Key Deliverables	Reference
1 – Assessment	4–8 weeks	Architectural audit; service boundary mapping	Migration blueprint; service catalogue	Auer et al. (2021)
2 – API Enablement	8–16 weeks	API gateway deployment; facade layer; contract testing	Stable API contracts; developer portal	Zdun et al. (2020); Shishmanov et al. (2021)
3 – Strangler Fig Migration	3–9 months	Incremental domain extraction; parallel running	Decoupled commerce services	Di Francesco et al. (2018); Soldani et al. (2018)
4 – Headless Frontend	2–4 months	SPA/PWA build; GraphQL BFF; CDN integration	Decoupled storefront; channel SDK	Brito & Valente (2020); Taibi et al. (2018)
5 – DevOps Maturity	Ongoing	CI/CD pipelines; observability; chaos testing	Automated delivery; SLOs	Leite et al. (2019); Waseem et al. (2020)
6 – Ecosystem Expansion	Ongoing	Partner APIs; marketplace; analytics layer	API-driven ecosystem; real-time BI	Verhoef et al. (2021); Dekimpe (2020)

6. OMNICHANNEL ENABLEMENT AND DIGITAL ECOSYSTEM ARCHITECTURE

6.1 Channel Integration Quality and Customer Engagement

Lee et al. (2019) showed empirically that the quality of channel integration (i.e., integration of multiple retail channels in terms of information provision, process consistency, and service configuration) is a statistically significant predictor of customer engagement in an omnichannel retailing environment. The study revealed that a higher channel integration quality led to a 40-60% rise in customer engagement scores and significant improvements in purchase frequency, customer satisfaction and brand advocacy scores. These results provide a direct commercial case for composable architectures, where the consistent API layer acts as the technical means of achieving cross-channel consistency of data and uniformity of processes. The composable approach uses the same API contracts for all customer-facing channels, whether it's a web storefront, a native mobile app, a call

centre interface or an in-store assisted sales tool, ensuring that product information, pricing logic, promotional eligibility, inventory availability and customer history are all presented in the same way, irrespective of channel context. The BFF pattern also allows for optimisation of the responses of each channel without the need to duplicate or fragment the core commerce service logic (Brito & Valente, 2020; Taibi et al., 2018).

Table 6: Omnichannel Integration Metrics Enabled by Composable Commerce Architecture (Synthesised from Lee et al., 2019; Brito & Valente, 2020; Dekimpe, 2020; Verhoef et al., 2021)

Omnichannel Dimension	Metric	Composable API Enablement	Source
Channel Integration Quality	40–60% higher customer satisfaction	Unified API layer synchronises touchpoints	Lee et al. (2019)
Cross-channel Data Consistency	<2-second sync latency	Event-driven state propagation	Chavan (2021); Dekimpe (2020)
Mobile Commerce Conversion	15–25% improvement	GraphQL-optimised payloads	Brito & Valente (2020)
Partner/B2B Integration Time	Reduced from months to days	API catalogue and self-service onboarding	Shishmanov et al. (2021)
Personalisation Responsiveness	Real-time (<500 ms)	Decoupled experience layer with ML feeds	Verhoef et al. (2021); Dekimpe (2020)
New Channel Time-to-Launch	Weeks (vs. 6+ months monolithic)	Headless storefront cloning and API reuse	Taibi et al. (2018); Pinheiro et al. (2019)

6.2 Real-Time Analytics and Personalisation

Dekimpe (2020) pointed out that the potential for big data analytics in retail settings depended on the architecture's ability to process and respond to customer behavioural signals in near real-time. Monolithic commerce platforms led to batch-oriented data processing architectures, which caused hours or days to pass between the observation of customer actions and their personalisation response, which is no longer appropriate for today's fast-changing digital commerce needs. Event-driven composable architectures allow for sub-second personalisation response loops, with the customer interaction events being streamed through messaging infrastructure to analytics and machine learning (ML) inference services. This was confirmed by Verhoef et al. (2021), who highlighted that real-time customer data activation is one of the key factors for success in digital transformations in commerce and retail.

6.3 API Ecosystem and Partner Integration

Shishmanov et al. (2021) expressed that the strategic importance of an enterprise API programme is not just in internal service integration, but also in the development of externally accessible digital ecosystems where partner organisations, independent software vendors (ISVs) and marketplace participants can integrate with enterprise commerce capabilities via published API contracts. In the composable commerce world, this dimension allows businesses to share product catalogue information with marketplace aggregators, receive orders from partner storefronts, integrate third-party loyalty programmes, and integrate fulfilment capabilities with logistics provider networks via standardised API interfaces that are managed in a centralised API governance framework. As of December 2021, enterprise commerce organisations' emerging strategic priority is commercial implications of API ecosystem monetisation.

7. DISCUSSION AND CONCLUSION

7.1 Synthesis of Findings

Reviewed literature confirms the architectural argument: API-driven composable commerce architectures are a materially better model for enterprise digital commerce on the dimensions of deployment agility, channel extensibility, integration scalability, and operational resilience. The results presented in Table 5 showed that the

deployment frequency for composable architectures was increased by 4 to 10 times compared with the monolithic baselines, the lead time was reduced by about 85%, the MTTR was improved by about 75%, and the change failure rate was reduced by 50-60% (Leite et al., 2019; Waseem et al., 2020; Auer et al., 2021). These operational enhancements directly impact commercial results, by providing faster feature delivery, less downtime exposure and greater responsiveness to market dynamics. The comparison between REST and GraphQL paradigms (Table 2) showed that REST is the best solution for enterprise commerce environments in which the data is used for transactional high-volume endpoints, while GraphQL is the best solution for the data used for experience layer aggregations, in terms of performance, development productivity and bandwidth efficiency (Brito & Valente, 2020). The event-driven integration layer (Table 4) enables the decoupling of integration that is necessary for distributed transaction management, real-time analytics feeds, and inventory state propagation throughout large commerce ecosystems (Chavan, 2021).

The migration roadmap outlined in Table 7 outlines a structured and staged approach to the adoption of composable commerce that helps to reduce the transformation risk that Auer et al. (2021) identified. With the support of the combination of the decomposition of the strangler fig and the enablement of the API facade, plus the incremental maturity progression in DevOps, organisations can benefit from part of the composable architecture features, such as channel integration through APIs and deployment independence for each service, while leaving some commerce platform components running in the interim between the old and the new (Di Francesco et al., 2018; Soldani et al., 2018).

Figure 4: DevOps Maturity Progression Model for Composable Commerce Deployment

Fig. 4 — DevOps maturity progression: deployment frequency rises as failure rate and MTTR decline

7.2 Challenges and Limitations

There are many challenges to navigating the shift to composable commerce architectures. According to Soldani et al. (2018), the pains of microservice adoption are: Complexity of operation, complexity of distributed systems debugging, introduction of network latency, and considerable investment in observability infrastructure. One of the issues the authors found was that organisations with dozens of independently deployed services often suffered

from lower reliability and developer productivity during the initial stages of MSA adoption because they were underestimating the operational costs. Perhaps similarly, Jamshidi et al. (2018) warned that the potential benefits of microservice architectures are not always achievable in all organizational settings and that team size, engineering maturity and product complexity act as important moderating factors.

In enterprise commerce contexts, distributed data management represents a particularly significant challenge. The per-service data ownership principle — while necessary for deployment independence — introduces eventual consistency constraints in scenarios such as cross-service product catalogue queries, real-time inventory visibility, and multi-currency pricing aggregation. These consistency trade-offs require careful architectural planning and compensating patterns including event sourcing, CQRS, and read-model materialisation, as documented by Chavan (2021) and Taibi et al. (2018).

7.3 Future Directions

As of December 2021, there were several emerging directions of API-driven composable commerce architecture identified from the reviewed literature. First, a key differentiator for artificial intelligence (AI) and machine learning features that are embedded in the service layer – especially personalisation, dynamic pricing, and demand sensing capabilities – is becoming increasingly significant (Verhoef et al., 2021; Dekimpe, 2020). Second, the emergence of marketplace commerce models, in which the third-party sellers, fulfilment providers, and content partners are integrated using published API contracts, was an emerging architectural pattern that had important revenue model implications, and was accelerating.

7.4 Conclusion

This paper has synthesized an extensive peer-reviewed empirical and analytical literature pertaining to API-driven enterprise commerce architecture for composable digital ecosystems up to the point of December 2021. The proposed four-layer architecture—experience layer, API orchestration layer, commerce services layer, and integration and ecosystem layer—offers a scalable, extensible and operationally resilient architecture for enterprise digital commerce modernisation.

It can help organisations enable omnichannel customer engagement, fast feature delivery, connect with digital ecosystem partners and activate real-time customer data via a single, API-driven service foundation. While directly tackling the core problems around slow deployment cycles, coupling of frontends, and inflexibility for integration in monolithic commerce platforms, the composable commerce architecture model offers an incremental migration path that minimises the risk of transformation for organisations with a large investment in their legacy platforms. The quantitative data from the synthesised literature shows that there are material improvements in all of the operational dimensions reviewed: deployment frequency (4-10x), lead time reduction (85%), MTTR improvement (75-60% improvement), and channel integration quality improvement (40-60%). All of these outcomes combine to make the API-driven composable commerce architecture the cornerstone of ongoing enterprise digital transformation in this channel-diversified, competitive business landscape.

Future research needs to explore the long-term organisational success of the adoption of composable commerce across different industries, governance models that would be necessary for managing a large-scale API ecosystem, and the architectural patterns that are likely to become prevalent when integrating AI-native capabilities into composable service layers. The evidence presented in this paper provides a solid scaffolding for enterprise commerce platform modernisation practitioners, architects and researchers.

REFERENCES

Auer, F., Lenarduzzi, V., Felderer, M., & Taibi, D. (2021). From monolithic systems to microservices: An assessment framework. Information and Software Technology, 137, 106600. https://doi.org/10.1016/j.infsof.2021.106600

Brito, G., & Valente, M. T. (2020). REST vs GraphQL: A controlled experiment. In 2020 IEEE International Conference on Software Architecture (ICSA) (pp. 81–91). IEEE. https://doi.org/10.1109/ICSA47634.2020.00016

Chavan, A. (2021). Exploring event-driven architecture in microservices: Patterns, pitfalls and best practices. International Journal of Science and Research Archive, 4(1), 229–249. https://doi.org/10.30574/ijsra.2021.4.1.0166

Dekimpe, M. G. (2020). Retailing and retailing research in the age of big data analytics. International Journal of Research in Marketing, 37(1), 3–14. https://doi.org/10.1016/j.ijresmar.2019.09.001

Di Francesco, P., Lago, P., & Malavolta, I. (2018). Migrating towards microservice architectures: An industrial survey. In 2018 IEEE 15th International Conference on Software Architecture (ICSA) (pp. 29–38). IEEE. https://doi.org/10.1109/ICSA.2018.00012

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. IEEE Software, 35(3), 24–35. https://doi.org/10.1109/MS.2018.2141039

Lee, Z. W. Y., Chan, T. K. H., Chong, A. Y.-L., & Thadani, D. R. (2019). Customer engagement through omnichannel retailing: The effects of channel integration quality. Industrial Marketing Management, 77, 90–101. https://doi.org/10.1016/j.indmarman.2018.12.004

Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). A survey of DevOps concepts and challenges. ACM Computing Surveys, 52(6), Article 127. https://doi.org/10.1145/3359981

Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., & Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A systematic literature review. Information and Software Technology, 131, 106449. https://doi.org/10.1016/j.infsof.2020.106449

Pinheiro, C., Vasconcelos, A., & Guerreiro, S. (2019). Microservice architecture from enterprise architecture management perspective. In B. Shishkov (Ed.), Business modeling and software design (Lecture Notes in Business Information Processing, Vol. 356, pp. 236–245). Springer. https://doi.org/10.1007/978-3-030-24854-3_17

Shishmanov, K. T., Popov, V., & Popova, P. E. (2021). API strategy for enterprise digital ecosystem. In 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T). IEEE. https://doi.org/10.1109/PICST54195.2021.9772206

Soldani, J., Tamburri, D. A., & van den Heuvel, W.-J. (2018). The pains and gains of microservices: A systematic grey literature review. Journal of Systems and Software, 146, 215–232. https://doi.org/10.1016/j.jss.2018.09.082

Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). Architectural patterns for microservices: A systematic mapping study. In Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER 2018) (Vol. 1, pp. 221–232). SciTePress. https://doi.org/10.5220/0006798302210232

Verhoef, P. C., Broekhuizen, T., Bart, Y., Bhattacharya, A., Dong, J. Q., Fabian, N., & Haenlein, M. (2021). Digital transformation: A multidisciplinary reflection and research agenda. Journal of Business Research, 122, 889–901. https://doi.org/10.1016/j.jbusres.2019.09.022

Waseem, M., Liang, P., & Shahin, M. (2020). A systematic mapping study on microservices architecture in DevOps. Journal of Systems and Software, 170, 110798. https://doi.org/10.1016/j.jss.2020.110798

Zdun, U., Wittern, E., & Leitner, P. (2020). Emerging trends, challenges, and experiences in DevOps and microservice APIs. IEEE Software, 37(1), 87–91. https://doi.org/10.1109/MS.2019.2947982