LTIMindtree

Redefining Retail & Consumer Commerce:

The Shift from Monoliths to Composable Architecture

Table of Contents

Executive Summary

The whitepaper underscores the critical need to transition from monolithic to composable commerce architecture in B2B commerce, especially within the retail and consumer business sectors. Monolithic systems stifle scalability and agility, making it challenging to keep pace with technological advancements. In contrast, composable commerce empowers businesses to create tailored platforms with independent components, enhancing flexibility and enabling swift updates.

This transition is driven by the necessity for flexibility, scalability, and innovation to meet evolving customer demands. B2B models grapple with complex workflows, diverse customer segments, and integration with various systems. The whitepaper outlines a three-phase approach for transitioning, focusing on foundational elements and incremental migration strategies. Effective data migration and synchronization between legacy and new systems are crucial. Moreover, integrating DevSecOps and cloud architecture is vital for optimizing the total cost of ownership and ensuring security and resilience in the new composable commerce environment.

Introduction

Business-to-business (B2B) commerce is rapidly growing due to the need for constant innovation and quick delivery of new features. Unlike business-to-consumer (B2C) commerce, B2B platforms are complex as they integrate multiple organizations' business processes into a seamless workflow. Organizations in B2B distribution are leveraging collaborative commerce to streamline information sharing among customers, suppliers, and manufacturers. Strategies like just-in-time (JIT), make-to-order (MTO), and build-to-order (BTO) help optimize inventory and reduce costs. B2B distribution must handle bulk procurement, demand fluctuations, contractual obligations, and high customer expectations regarding delivery times, product quality, and domain expertise.

The traditional monolithic IT infrastructures, which are built as a single, interconnected platform, are known for their limited scalability, lack of agility, longer time-to-market, difficulty in introducing new functionalities, complexity implanting technology updates, and application outages, among others. It is challenging for businesses to meet the evolving customer demands and technological advancements with monolithic commerce. Therefore, transitioning to a composable commerce platform gives businesses a competitive edge as it offers greater flexibility, scalability, and innovation. What is composable commerce? Composable commerce is a modern approach to building a custom platform designed to meet specific business needs. It empowers businesses to build an omnichannel platform with "composable components" that can be developed, scaled, and updated independently, making the platform agile and flexible.

Seamless shopping experiences are crucial in the retail and consumer industry. To enable seamless shopping experiences, businesses need a robust B2B commerce platform to handle complex supply chain operations, manage large data volumes, and deliver personalized customer experiences.

While there are many similarities between B2B and B2C models, there are some fundamental differences in customer models, product features, and workflows in B2B models. B2B commerce platforms must handle complex workflows, diverse customer segments, and integration with various systems. Unlike B2C, where the focus is on individual consumers, B2B involves multiple stakeholders, bulk orders, and long-term contracts. This complexity requires a robust and flexible architecture to meet the unique demands of B2B commerce, particularly in the Retail and Consumer business sectors.

While Both B2B and B2C follow a standardized approach for order capture, as illustrated below but

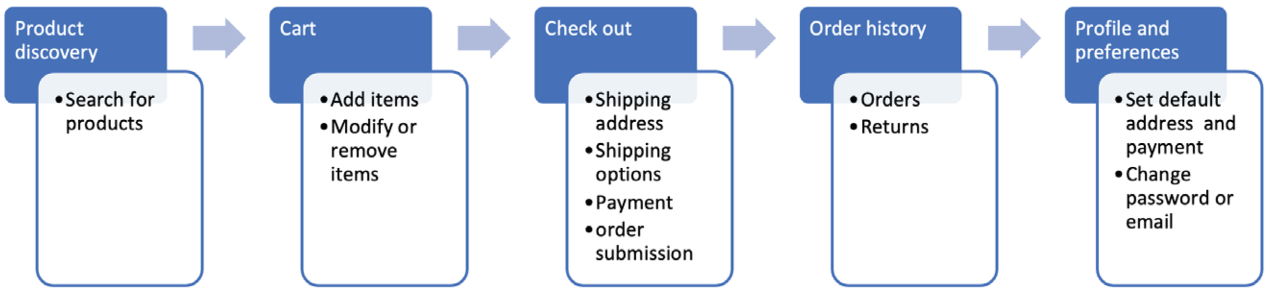
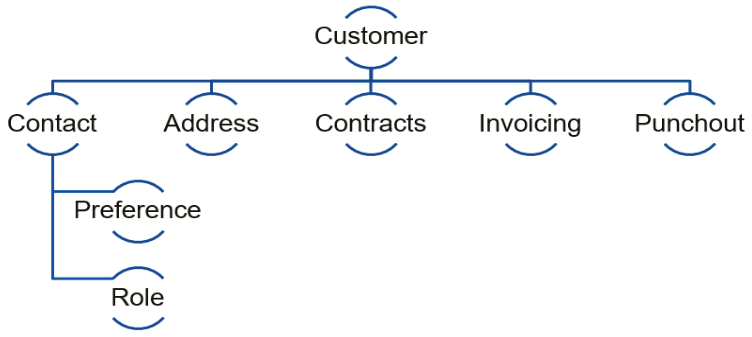


Figure 1: B2B and B2C standardized approach for order capture

B2B commerce model will have typically these core features

1. **Customers:** Organizations can register as customers, enabling multiple employees to access the B2B commerce portal and place orders on their behalf.
2. **Contracts:** Customers receive contracted prices through the portal and can leverage features like bulk order discounts and item restrictions.
3. **Invoicing:** Order payment is decoupled from the order process with this module.



Challenges of monolith infrastructures for B2B commerce growth

In the early 2000s, organizations quickly built ecommerce storefronts, followed by B2B commerce for agility and convenience of the business . Initially, tightly coupled architectures with ERP systems posed challenges like heavy customizations and complexity. Three-tier architecture offered some decoupling, but maintaining UI and business logic independently was difficult. Service-oriented architectures and middleware models laid the foundation for modern microservices, enabling continuous adaptation. B2B commerce faces diverse customer segments, making feature rollout and personalization challenging. B2B Distributors must address evolving customer needs and internal cost optimizations, integrating efficient sales and logistics models with the e-commerce experience.

According to a McKinsey survey[2] e-commerce and account manager-based sales compete for the top sales channel position in B2B. This necessitates a sophisticated and feature-rich B2B Commerce experience. Omnichannel experience demands a decoupled design. Monoliths have tight coupling, causing unexpected behavior and delivery contention. They have longer release cycles, making scaling and maintenance challenging.

Advantages of composable commerce for B2B

To optimize **B2B commerce**, it's essential to use **best-in-class products** tailored to specific needs, allowing for **plug-and-play features** without the limitations of generic e-commerce packages or complex home-grown frameworks. **Microservice abstraction** helps remove product dependency, enabling rapid value-add changes and preventing vendor lock-in, as highlighted by a **Forbes study[3] . Composable commerce** also allows the integration of business-centric tools, enabling workflow-based solutions for real-time demand shifts without relying on IT teams. **Headless services,** which distinguish business services from user experience, facilitate rapid UI changes and integration across different stacks in an **omnichannel** scenario. This approach reduces **total cost of ownership (TCO)** by focusing on on-demand scaling and allowing for rapid product exits if costs become prohibitive. **Faster time to market** is achieved as teams focus on core competencies and small feature iterations, reducing delivery contention as long as service contracts remain unchanged and sufficient regression tests are in place.

Scalability is enhanced through microservices, allowing parallel development and resource scaling on demand. Calculating ROI for such transformation projects is complex, but the real benefit lies in setting up an environment for rapid innovation and reduced time to market. This is achieved by deploying parallel development teams and fast-tracking the innovation-feedback cycle.

Navigating the risks and challenges in B2B commerce digital transformation

Digital transformation into composable commerce offers significant benefits when planned and executed properly.

- Focus on agility, innovation, and scalability to ensure a successful transition without delaying time to market.
- Assess your current architecture; if the modular monolith is mature and teams are well-integrated, transitioning to microservices may not be necessary.
- If moving forward with a transformation, select products and tools based on clear acceptance criteria and proof of concepts.
- Prioritize foundational elements during development to minimize risks Focus on agility innovation, and scalability to ensure a successful transition without delaying time to market.
- Assess your current architecture; if the modular monolith is mature and teams are well-integrated, transitioning to microservices may not be necessary.
- If moving forward with a transformation, select products and tools based on clear acceptance criteria and thorough proof of concepts.
- Prioritize foundational elements during development to minimize risks and ensure a strong base for future growth.

Steps for digital transformation from monoliths to composable commerce for B2B

There are several foundational elements that must be identified first. It is best to have a three-phase approach to achieve the successful transformation from monoliths to composable commerce for B2B platforms.

1. Identify foundation elements about products, tools, and high level of architecture.

a) Products:

- Digital experience platform
- Cloud platform and data storage
- Search engines
- Commerce engine: Driven by headless systems like commerce tools, custom-built, or hybrid.
- Customer identity and access management (CIAM)
- Frameworks for UI and API
- API gateway

b) Tools:

- Source repository
- Artifact registry
- Build and pipeline tools
- Security scanners
- Code quality scanners
- Test automation tools
- Agile PM tools
- Technical documentation tools

c) Architectural approaches and designs for cross-cutting concerns must be addressed to ensure consistency. Focus on establishing uniform development standards and guidelines, while documenting and aligning them with clearly defined architectural strategies.
Some of the strategies are:

- Content management
- UX, UI, and SEO
- API service architecture
- Caching and performance
- Identity management and access control
- Test automation
- Cloud architecture and dev-ops

2. Planning

There are two main approaches for migrating a monolith to microservices:

- The first is building a new platform from scratch, but this waterfall approach may not be acceptable due to long wait times.
- The preferred approach is incremental migration, which decomposes the monolith. To avoid rework, it's crucial to plan foundational and cross-cutting architecture elements like CIAM, Cloud, DXP, and DevOps pipelines first. Functional features can then be sequenced for seamless production migration with minimal impact.

3. Execution

During the planning phase, an agile organization structure with smaller scrum teams should be put in place to achieve parallel development of functional features. Designs of individual features should be covered here.

Approaches for incremental decomposition

Domain-driven design (DDD) is key for e-commerce architecture. In DDD, bounded contexts are related to business functions with clear interfaces. For B2B, domains like catalog, cart & checkout, and Invoicing convert to microservices. These are loosely coupled, reusable, and ensure autonomy and scalability. Decomposing complex monoliths like B2B commerce platforms is challenging due to module interdependencies. Bounded contexts help identify features for migration. A customer and segment-based migration model ensures a universal experience and avoids data corruption.

The following patterns support the incremental migration of features:

1. Strangler fig pattern:

Using a proxy pattern, client requests can be redirected to either the old or new experience. The proxy can be a server acting based on specific rules or a programmatically implemented wrapper over existing APIs. Consider the search function in an e-commerce program that calls a search engine. There are two scenarios:

a. If the legacy application is based on SOA and has a search service calling the search engine, apply the strangler pattern at the service layer:

i. Create a new search service following REST principles with a transformation layer to accept requests in the old format, using API Gateways or programming.

ii. Introduce a proxy layer to switch between old and new services based on an inbound identifier, such as a customer ID cookie or A/B test cookie.

iii. Switch the legacy application to point to this proxy layer, requiring minimal changes.

b. If the legacy application lacks SOA and tightly couples UI and Search Functions, apply the strangler pattern at the UI Layer:

i. Create a new search service and search UI that integrates with the search engine.

ii. Introduce a proxy using an HTTP server/load balancer to redirect between legacy and modern UI-based on defined rules.

iii. Modify the legacy application for a transitional user experience, easing customer acceptance of the new interface.

During transactional scenarios, synchronizing data between both systems becomes more challenging. This will be covered in the data migration section.

2. Branch by abstraction Pattern: This pattern is used when the logic that has to be extracted is too tightly coupled with business logic for other domains or UI.

In the above scenario, if we still need to retain the legacy UI but still calling the search service, we will choose the branch by abstraction pattern.

a. Create an abstract layer for the search service interface for both old and new services.

b. Refactor the UI code so it can call this new abstract search service interface.

c. Now, switches can be used to control the calls between legacy and new search services.

UI as micro front ends: Similar to microservices, UI must also be decomposed page by page. The micro front-end model helps build individual pages and hook them into an existing web portal so that a seamless user experience can be achieved. Each page or micro app can be independently developed and deployed and loosely coupled with other pages.

3. Switch-based migration pattern

During incremental migration, components of the legacy system and transformed system may co-exist and become part of a single-user story. Customers may move seamlessly between the legacy and new systems. To facilitate this, a customer + feature-based switch combination is recommended.

The matrix below depicts how the switch pattern can work:

Customer (Organization)	Feature name	Switch value
Customer1	Search	New system
Customer 2	Invoicing	Legacy system

Table 1: Depicting switch pattern

Data migration and management

Data decomposition for monolith to microservice migration has two phases: Initial data migration and data synchronization.

1. Initial data migration

Migrating B2B commerce data is more complex than B2C because of the tightly coupled functionalities and the need for seamless interoperability across numerous features. To address this, follow these steps:

1. Identify customers and features for migration. Some customers may use features on both legacy and new systems during incremental migration. Build a modular data migration program to support customer and feature-based data migration.
2. Prepare data transformation programs based on source and destination formats. For example:
   - Customer data may flow into CIAM
   - Previous and in-progress order data may flow into the OMS and order history database
   - Other B2B data may be stored in a relational or NoSQL database
3. Prepare validation programs to ensure all data is successfully transformed and inserted into the new system.

Confidential data like passwords and credit cards should not be migrated to comply with international regulations. For example, if we introduce a new CIAM system, we will provide customers with the option to reset their passwords rather than decrypting and re-encrypting current passwords.

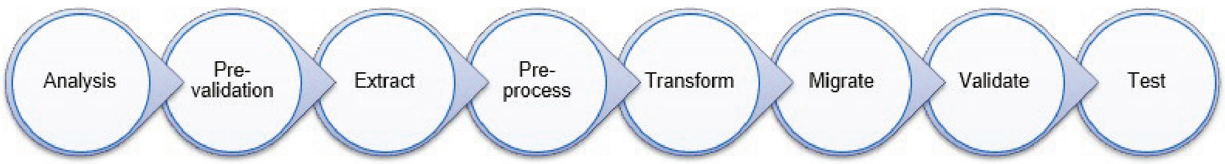


Figure 3: Typical data migration program

2. Data synchronization

To support failover scenarios, synchronize data between the new and legacy systems using event-based systems, database triggers, or messaging queues. Avoid exposing legacy features to migrated customers. If necessary, enable two-way synchronization, which can cause issues like locking and data inconsistency. Irreversible actions, such as moving to a new CIAM system, can block failover.

3. Saga pattern and compensating transactions

Microservices can call other microservices to retrieve or update information, with some services aggregating multiple workflow steps for centralized state control. For example, a checkout service can handle creating or selecting a shipping address, adding payment, and submitting a cart by invoking relevant services. These are known as orchestrator services and implement the Saga pattern, which coordinates data changes across multiple services. The orchestrator pattern uses a central coordinator, while the choreography pattern relies on event triggers. To ensure data consistency and integrity during failures, compensating transactions must be programmed for each service to roll back transactions to the point of failure.

DevSecOps and migration of DevOps tasks strategy

DevSecOps deals with the aspects of integrating and automating pre-release operations, code quality, and security auditing with the development and testing life cycle of a product.

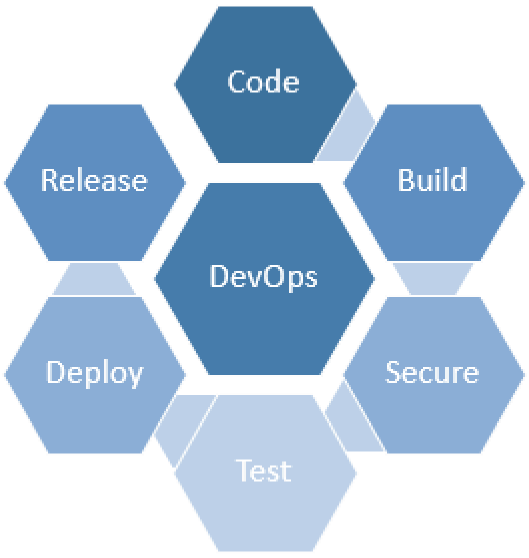


Figure 4: Typical DevSecOps cycle

Cloud architecture is a key to optimizing the total cost of ownership. Some factors to be considered when optimizing cloud architecture are:

- Regions of operation and latency
- Propensity towards completely cloud-native models or preference towards cloud vendor-agnostic models
- Alignment of products in composable architecture aligned to the same cloud provider
- Security, high availability, and resilience
- Build cloud-ready APIs using Docker/Kubernetes, ensuring they are stateless and session-free.
- Start with lift-and-shift to VMs with containers, then move to Kubernetes, which is fully cloud-managed.
- Invest in making applications that are cloud-native to save effort.
- Choose a modern source control system with redundancy and pipeline automation.
- Set up CI/CD pipelines for approval-based deployments integrated with a ticketing service.
- Automate Swagger documentation, code quality scanners, SAST/DAST security scanners, unit tests, and coverage analysis.
- Leverage managed services for storage, sensitive data, configuration management, and cloud-managed deployments.
- Automate infrastructure creation using Terraform and implement security scanning to identify vulnerabilities.

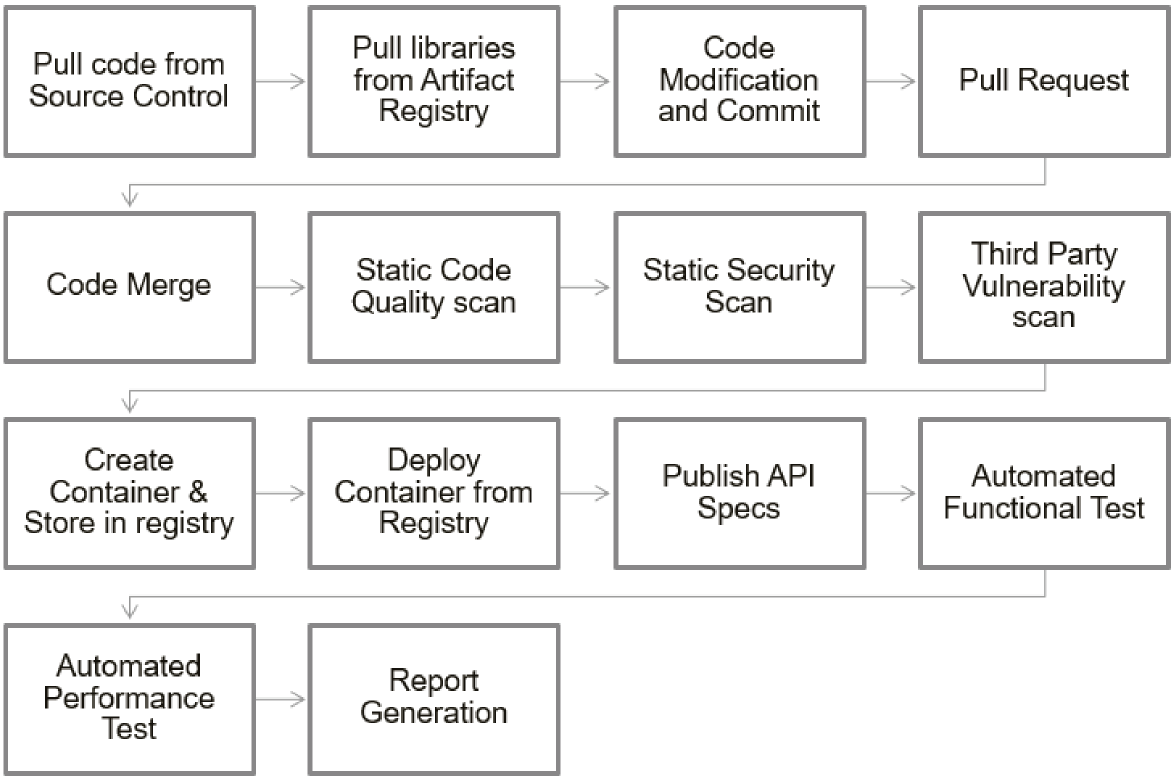

Pull code from Source Control
Pull libraries from Artifact Registry
Code Modification and Commit
Pull Request
Code Merge
Static Code Quality scan
Static Security Scan
Third Party Vulnerability scan
Create Container & Store in registry
Deploy Container from Registry
Publish API Specs
Automated Functional Test
Automated Performance Test
Report Generation

Figure 5: Migration of DevOps tasks

Conclusion

Digital transformation of a B2B commerce application into composable commerce brings significant benefits if planned and executed properly. Ensure the move is driven by agility, innovation, scalability, and time to market. If your current modular monolith architecture is mature and teams are in synergy, a transformation to microservices may not be necessary. However, if proceeding, select products and tools based on acceptance criteria and proof of concepts. Optimize development by implementing foundational elements first. Organizational transformation is also required, with teams adopting an agile mindset for continuous improvement and innovation. Digital transformation in the retail industry is crucial for seamless shopping experiences and enhanced customer engagement.

Citations

[1]E-Commerce 2000: The Year of Living Dangerously, Jon Weisman, eCommercetimes, December 29, 2000: *https://www.ecommercetimes.com/story/e-commerce-2000-the-year-of-living-dangerously-6380.html*

[2] Busting the five biggest B2B e-commerce myths, Manu Bangia, Liz Harrison, Candace Lun Plotkin, and Kate Piwonski, McKinsey, January 26, 2022: *https://www.mckinsey.com/capabilities/growth-marketing-and-sales/our-insights/busting-the-five-biggest-b2b-e-commerce-myths*

[3] Outpace The Competition With Composable Commerce, Robert Harbols, Forbes, September 06, 2022:

https://www.forbes.com/councils/forbesbusinesscouncil/2022/09/06/outpace-the-competition-with-composable-commerce/

Authors

Visakh Sankaranarayanan

Senior Principal – Architecture

Visakh has over 22 years of experience in digital and e-commerce programs, with deep expertise in HCL Commerce and Commerce Tools. He specializes in B2C and B2B customizations as well as B2B sales channel integrations.

Atish Roy

Senior Principal – Architecture

With over 17 years of experience, Atish specializes in digital and e-commerce programs, with expertise in Commerce Tools, HCL Commerce, MACH architecture, cloud architecture, DevSecOps, and CI/CD strategies, with a strong focus on the Google Cloud Platform.

Akshatha Kamath

Principal Architect

With 16 years of experience in digital e-commerce for Retail CPG and Manufacturing, including 13 years at LTIMindtree, Akshatha specializes in e-commerce and Google Services.

Dr. Rajesh Singh

Associate Vice President, Retail CPG

Dr. Rajesh Singh, a Ph.D. graduate from the Indian Institute of Management, specializes in Big Data and Finance. With over 26 years of experience, he has led Digital Transformation Programs in Retail CPG, leveraging expertise in Digital Technologies, Cloud Computing, AI/ML/NLP, and Data Science.

Abhishek Kaushik

Principal Director, Retail and CPG Industry, LTIMindtree

Abhishek leads digital transformation strategic accounts at LTIMindtree, partnering with clients to solve business challenges and drive growth across BSFI, Manufacturing, Retail, and CPG industries.

LTIMindtree