NovaRetail Enterprise Architecture Modernization Plan

Prepared for: NovaRetail CTO & Senior Leadership

Prepared by: Nithin Mohan T K

Date: November 22, 2025

Document Version: 0.1

Document Status: First Draft

Confidentiality: Internal Use Only

Executive Summary

The Challenge

NovaRetail operates 12 independent e-commerce platforms across different countries, each built with disparate technology stacks and isolated data systems. This fragmentation creates critical business challenges:

- Inconsistent customer experience across markets
- Unable to recognize customers across borders (8 separate databases)
- Slow feature deployment (months instead of weeks)
- Operational inefficiency from batch processing and manual data management
- High operational costs from maintaining multiple technology stacks
- Scalability limitations preventing rapid market expansion

The Solution

A cloud-first, event-driven microservices architecture that unifies customer identity, enables real-time order processing, and provides a composable platform for rapid feature deployment. This transformation will be executed over 3 years using proven migration patterns to minimize business disruption.

Expected Business Outcomes

Metric	Current State	Target State (Year 3)
Time to Market (New Features)	3-6 months	2-4 weeks
Customer Data Accuracy	~60% (fragmented)	95%+ (unified)
Order Processing Latency	12-24 hours (batch)	<5 minutes (real-time)
System Availability	95% average	99.9% SLA
Infrastructure Cost	High (mixed on-prem/cloud)	-30% (cloud-optimized)
Developer Productivity	Low (tech debt)	+50% (standardized)

Table 1: Expected business impact metrics

Why This Approach

✓ **Proven Migration Patterns:** Strangler Fig pattern enables gradual, low-risk migration
✓ **Business Value Focus:** Delivers value incrementally, not as "big bang" at year 3
✓ **Cloud-Native Architecture:** Leverages modern patterns for scalability and resilience

- ✓ **Event-Driven Integration:** Enables real-time processing and loose coupling
- ✓ **Domain-Driven Design:** Aligns technical architecture with business domains
- ✓ **Technology Standardization:** Reduces complexity while preserving team autonomy

Investment: Estimated $4-6M over 3 years with measurable ROI starting Q2 Year 1.

1. Current State Assessment

High-Level Architecture Overview

System Landscape Description

The current Nova Retail architecture consists of 12 independent country-specific e-commerce platforms with no standardization:

Application Layer:

- 12 separate web applications (.NET, Java/Spring, PHP, Node.js)
- Each country team independently developed their solution
- No shared components or services
- Direct database access from application code

Data Layer:

- 8 different customer databases with overlapping but inconsistent schemas
- Product data maintained in Excel spreadsheets
- Manual data uploads and synchronization
- No master data management (MDM)
- Order data consolidated via nightly batch jobs

Integration Layer:

- Point-to-point integrations between systems
- Synchronous HTTP calls creating tight coupling
- No centralized API gateway or service mesh
- Cascading failures due to direct dependencies

Infrastructure Layer:

- Mixed deployment: some on-premises, others on AWS, some on Azure
- No containerization or orchestration
- Limited monitoring and observability
- Manual deployment processes

Current State Architecture Diagram

[Reference: NovaRetail-Complete-Architecture-v0.1.drawio/C1-System-Context-Current]

Key Architectural Issues and Risks (Prioritized)

Critical Priority (P0) - Immediate Business Impact

1. **No Unified Customer Identity (Risk Score: 9.5/10)**

Issue: Customer data exists in 8 isolated databases with no synchronization or master record.

- Business Impact:
    - Cannot track customer journey across countries
    - Duplicate marketing efforts and wasted spend
    - Poor customer experience (must re-register in each country)
    - Impossible to implement global loyalty programs
    - GDPR compliance risk (cannot fulfill data deletion requests)

2. **Batch Processing for Orders (Risk Score: 9.0/10)**

Issue: Order processing relies on nightly batch jobs with 12-24 hour delays.

- Business Impact:
    - Delayed order fulfillment affecting customer satisfaction
    - Inventory cannot be updated in real-time leading to overselling
    - Cannot support modern fulfillment (same-day delivery, click-and-collect)
    - Competitive disadvantage vs. real-time competitors

3. **Manual Product Data Management (Risk Score: 8.5/10)**

Issue: Product information managed in Excel spreadsheets with manual uploads.

- Business Impact:
    - High error rate in product data (pricing, descriptions, availability)
    - Cannot launch products simultaneously across markets
    - Time-to-market for new products: weeks instead of hours
    - Lost revenue from incorrect pricing or stock information

High Priority (P1) - Operational Efficiency

4. **Technology Fragmentation (Risk Score: 8.0/10)**

Issue: Four different technology stacks (.NET, Java, PHP, Node.js) across platforms.

5. **Point-to-Point Integration Pattern (Risk Score: 7.5/10)**

Issue: Direct synchronous calls between systems creating tight coupling.

6. **Infrastructure Inconsistency (Risk Score: 7.0/10)**

Issue: Mixed deployment across on-premises, AWS, and Azure with no standardization.

Medium Priority (P2) - Scalability and Maintainability

7. **Monolithic Architecture per Country (Risk Score: 6.5/10)**
8. **Limited Observability (Risk Score: 6.0/10)**
9. **No Automated Deployment Pipeline (Risk Score: 5.5/10)**

Stated Assumptions

1. **Business Assumptions:**
    a. NovaRetail plans to continue international expansion (3-5 new countries in next 3 years)
    b. Customer acquisition cost is high, making customer retention critical
    c. Competition is increasing with better-funded, technology-native competitors
    d. Leadership is committed to 3-year transformation timeline and investment

2. **Technical Assumptions:**
    a. Average traffic: 100K daily active users per country platform
    b. Peak traffic (Black Friday): 5-10x normal load
    c. Current system availability: approximately 95% (varies by country)
    d. Data volumes: ~10M customers total, ~50K orders per day globally
    e. Most customer data can be reconciled/deduplicated via email address

3. **Organizational Assumptions:**
    a. Development teams are willing to adopt new technologies and practices
    b. Some resistance expected from country teams (autonomy concerns)
    c. Current team size: ~80 developers across 12 country teams
    d. Budget available: $4-6M over 3 years for architecture modernization

4. **Regulatory and Compliance:**
    a. Must comply with GDPR (European markets) and local data protection laws
    b. PCI-DSS compliance required for payment processing
    c. Some countries have data residency requirements

2. Target Architecture Proposal

Target Architecture Vision

The future-state architecture for NovaRetail is designed as a cloud-native, event-driven microservices platform that enables business agility while maintaining operational excellence.

Core Design Principles

Principle	Description
Domain-Driven Design	Organize services around business domains (Customer, Order, Product, Inventory, Fulfillment) not technical layers
API-First Development	All services expose well-defined APIs; UI and integrations consume APIs only
Event-Driven Integration	Use asynchronous event messaging for inter-service communication; minimize synchronous coupling
Cloud-Native by Default	Design for containerization, orchestration, and cloud services; leverage managed services
Single Source of Truth	Each domain owns its data; no shared databases; data distributed via events
Autonomy with Governance	Country teams retain autonomy within standardized architecture guardrails
Security by Design	Zero-trust architecture; authenticate and authorize at every boundary
Design for Failure	Assume failures will occur; implement circuit breakers, retries, and graceful degradation
Observability First	Comprehensive logging, metrics, and tracing built into every service from day one
Evolutionary Architecture	Enable incremental changes; avoid big-bang migrations; strangler fig pattern

Data Residency	Support country-specific data residency requirements through regional deployments
Developer Experience	Provide self-service tools, clear documentation, and automated workflows

Table 2: Architecture principles for NovaRetail target state

Target Architecture Diagram (Logical View)

[Reference: NovaRetail-Complete-Architecture-v0.1.drawio/C2-Logical-View]

Target System Context Diagram

[Reference: NovaRetail-Complete-Architecture-v0.1.drawio/C1-System-Context-Target]

The target architecture represents *"NovaRetail OmniCommerce Platform .NET 10 Microservices on Azure + AWS DR Event-driven, API-first"* - a unified platform leveraging .NET 10 and Java Spring Boot microservices deployed on Azure (primary) with AWS as warm standby for disaster recovery.

Domain Model (Bounded Contexts)

Core Domains

Domain	Responsibilities	Key Entities
Customer Identity	Unified customer profile, authentication, preferences	Customer, Profile, Authentication, Consent
Product Catalog	Product information management, categorization, search	Product, Category, Attribute, Price, SKU
Order Management	Order lifecycle, order processing, order history	Order, OrderLine, Cart, Payment, Status
Inventory	Stock levels, reservations, availability	StockItem, Location, Reservation, Movement
Fulfillment	Shipping, delivery, returns	Shipment, Carrier, DeliverySlot, Return
Payment	Payment processing, refunds, transaction records	Transaction, PaymentMethod, Refund

Table 3: Core business domains

Supporting Domains

Domain	Responsibilities	Key Entities
Notification	Multi-channel messaging	Message, Template, Channel, Preference
Promotion	Pricing rules, discounts, campaigns	Campaign, Rule, Coupon, Discount
Analytics	Business intelligence, reporting	Metric, Report, Dashboard, KPI
Content Management	CMS for marketing content	Page, Asset, Campaign, Localization

Table 4: Supporting business domains

Domain Relationships and Event Flow Example

- **Customer Registration Flow:**

1. Customer registers via Web App
2. API Gateway → Customer Identity Service
3. Creates customer profile
4. Publishes: customer.registered event
5. Event consumed by Notification Service → Sends welcome email
6. Event consumed by Analytics Service → Updates customer acquisition metrics
7. Event consumed by Promotion Service → Assigns new customer offer

- **Order Placement Flow (Event-Driven Saga):**

[Reference: NovaRetail-Complete-Architecture-v0.1.drawio/C3-Order-Saga and NovaRetail-Complete-Architecture-v0.1.drawio/Sequence-Order-Saga]

1. Customer places order via Web/Mobile App
2. Order Management Service validates cart, creates order (status: PENDING), publishes: order.created event
3. Event consumed by Inventory Service → Reserves stock, publishes: inventory.reserved event
4. Event consumed by Payment Service → Processes payment, publishes: payment.completed event
5. Order Management Service consumes inventory.reserved + payment.completed, updates order status → CONFIRMED
6. Event consumed by Fulfillment Service → Creates shipment
7. Event consumed by Notification Service → Sends confirmation email/SMS
8. Event consumed by Analytics Service → Updates sales metrics

This event-driven approach eliminates batch processing and enables near real-time order processing.

3. Migration and Roadmap

Migration Strategy: Strangler Fig Pattern

We will employ the Strangler Fig pattern as our primary migration approach. This pattern allows us to incrementally replace legacy functionality with new microservices while both systems run in parallel, minimizing risk and delivering value continuously.

Why Strangler Fig?

- **Incremental Value Delivery:** Business benefits start immediately, not after 3 years
- **Risk Mitigation:** Can pause or adjust based on learnings and business priorities
- **Continuous Operation:** No "big bang" cutover; legacy systems remain operational
- **Team Learning:** Teams learn new patterns gradually, reducing adoption risk
- **Proven Success:** Widely adopted by Amazon, Netflix, Spotify for similar migrations

Implementation Approach

1. **Facade Layer:** Implement API Gateway as facade routing traffic to new/old systems
2. **Identify Seams:** Find natural boundaries in monoliths for service extraction
3. **Build New Services:** Implement functionality in new microservices (.NET 10 or Java Spring Boot)
4. **Route Traffic:** Gradually shift traffic from legacy to new services
5. **Retire Legacy:** Once all traffic migrated, decommission old system

3-Year Phased Migration Plan

Year 1: Foundation and Quick Wins (Q1 2026 - Q4 2026)

- **Q1 2026: Infrastructure Foundation**
  - Establish cloud landing zone (Azure primary, AWS DR)
  - Deploy Kubernetes clusters (AKS primary, EKS standby) in 3 regions (US, EU, APAC)
  - Implement API Gateway (Azure API Management) and service mesh (Istio)
  - Set up CI/CD pipelines (Azure DevOps/GitHub Actions) and observability stack
  - Deploy event streaming platform (Azure Event Hubs / AWS MSK)

  Deliverable: Cloud platform ready for service deployment

- **Q2 2026: Customer Identity Service (Wave 1)**
  - Build unified Customer Identity Service (.NET 10 / Java Spring Boot)
  - Migrate and deduplicate customer data from 8 databases
  - Implement OAuth2/OIDC authentication
  - Create customer data synchronization via CDC (Change Data Capture)
  - API Gateway routes authentication to new service

Deliverable: Single customer view across all platforms

Business Value: Enable cross-country customer recognition, GDPR compliance

- **Q3 2026: Product Information Management (Wave 2)**
  - Deploy centralized Product Catalog Service (PIM)
  - Migrate product data from spreadsheets to PIM
  - Implement product data APIs
  - Build admin UI for product management
  - Legacy systems consume product data via API (strangler fig)

  Deliverable: Centralized product management

  Business Value: Reduce time-to-market for new products from weeks to hours

- **Q4 2026: Order Management Foundation (Wave 3)**
  - Build Order Management Service
  - Implement real-time order processing (replace batch)
  - Deploy Inventory Service for stock management
  - Create event-driven order flow
  - Pilot in 2 countries

  Deliverable: Real-time order processing in pilot markets

  Business Value: Enable same-day fulfillment, reduce overselling

Year 2: Core Migration and Expansion (Q1 2027 - Q4 2027)

1. Q1 2027: Payment and Fulfillment Services (Wave 4)
2. Q2 2027: Country Platform Migration (Wave 5A) - 4 high-traffic countries
3. Q3 2027: Country Platform Migration (Wave 5B) - 4 medium-traffic countries
4. Q4 2027: Analytics and Promotion Services (Wave 6)

Year 3: Completion and Optimization (Q1 2028 - Q4 2028)

1. Q1 2028: Final Country Migration (Wave 7)
2. Q2 2028: Advanced Features (Wave 8)
3. Q3 2028: Performance Optimization
4. Q4 2028: New Market Expansion Ready

Key Dependencies and Risks

Dependency	Impact if Delayed	Mitigation
Cloud infrastructure setup	Blocks all service deployment	Start Q1, allocate dedicated team
Customer data deduplication	Cannot unify customer identity	Begin data analysis immediately, use ML for matching
API Gateway deployment	Cannot implement strangler pattern	Prioritize in Q1, use managed service
Event streaming platform	Cannot enable real-time processing	Deploy in Q1, choose managed service
Team upskilling	Slow adoption, quality issues	Training program in parallel with Q1

Table 5: Critical dependencies and mitigations

Risk	Probability	Impact	Mitigation
Data migration errors	Medium	High (customer trust)	Extensive testing, parallel run validation
Performance degradation	Medium	High (revenue loss)	Load testing, gradual rollout, quick rollback
Team resistance	High	Medium (delays)	Change management, early wins, training
Budget overrun	Medium	High (scope reduction)	Phased funding, value delivery gates
Vendor lock-in	Low	Medium	Multi-cloud strategy, use open standards
Security vulnerabilities	Low	Critical	Security reviews, pen testing, bug bounty

Table 6: Key migration risks

4. Reference Architecture and Standards

Reference Architecture for New Services

Service Template (Runtime View)

Every microservice at NovaRetail follows this standardized architecture, whether built with .NET 10 / ASP.NET Core or Java 17+ / Spring Boot:

- API Layer (RESTful APIs) - OpenAPI 3.0 specification, JSON request/response, JWT authentication, Input validation
- Business Logic Layer - Domain models, Business rules, Workflow orchestration
- Data Access Layer - Repository pattern, Database abstraction, Caching (Redis)
- Database (PostgreSQL/MongoDB/Cosmos DB) - Schema versioning, Backup and recovery
- Event Publishing/Consuming - Publishes domain events to Event Hubs/Kafka, Subscribes to relevant events, Event schema validation
- Cross-Cutting Concerns (via Service Mesh) - Health checks, Metrics endpoint, Distributed tracing, Structured logging, Circuit breaker

Integration Patterns

Pattern	Use Case	Implementation
Synchronous API Call	Read operations, queries, lookups	REST APIs via API Gateway, gRPC for internal service-to-service
Asynchronous Events	State changes, notifications, workflows	Azure Event Hubs / AWS MSK / Kafka topics with event schema registry
Saga Pattern	Distributed transactions (e.g., order placement)	Choreography-based saga using events
CQRS	Separate read/write models for high-scale domains	Command writes, event-sourced read models
API Composition	Aggregate data from multiple services	Backend-for-Frontend (BFF) pattern

Table 7: Integration patterns and usage

Data Patterns

- **Database per Service:** Each service owns its database; no shared databases

- **Event Sourcing:** For audit-critical domains (Order, Payment)
- **Change Data Capture (CDC):** Sync legacy data to new services during migration
- **Caching Strategy:**
- - L1 (Application): In-memory cache
- - L2 (Distributed): Redis for shared cache
- - CDN: Azure Front Door / CloudFront for static assets and API responses

Technology Standards

Programming Languages

NovaRetail standardizes on two primary backend technologies to balance team expertise, ecosystem maturity, and strategic alignment:

Language	Use Case	Rationale
.NET 10 / ASP.NET Core	Primary backend services, high-throughput APIs	Modern, performant, excellent Azure integration, strong enterprise support, growing team expertise
Java 17+ (Spring Boot)	Backend services, high-throughput APIs	Mature ecosystem, extensive team expertise, excellent performance, proven at scale
Node.js (TypeScript)	BFF, lightweight services, real-time	Fast development, async I/O, JavaScript ecosystem
Python 3.11+	Data processing, ML/AI services	Rich data science libraries, ease of prototyping
Go	Performance-critical services	Low latency, excellent concurrency

Table 8: Approved programming languages

Migration Note: Legacy PHP and .NET Framework applications will be rewritten in .NET 10 or Java Spring Boot during migration. Teams can choose based on expertise and service requirements, with .NET 10 recommended for new Azure-native services and Java Spring Boot for services requiring extensive open-source ecosystem integration.

Cloud and Infrastructure

- **Primary Cloud:** Azure (comprehensive managed services, strong .NET integration, enterprise support)

- **Secondary Cloud (Warm Standby):** AWS (disaster recovery, RTO ≤ 4 hours)
- **Tertiary Cloud (Cold Standby):** GCP (additional resilience, RTO ≤ 24 hours)
- **Container Orchestration:** Kubernetes (Azure AKS primary, AWS EKS standby, GCP GKE tertiary)
- **Service Mesh:** Istio (traffic management, security, observability)
- **Infrastructure as Code:** Terraform + Azure Bicep / AWS CDK
- **Serverless:** Azure Functions / AWS Lambda for event-driven functions
- **API Gateway:** Azure API Management (primary) / AWS API Gateway (standby)

Data and Messaging

Technology	Use Case
Azure Database for PostgreSQL / AWS RDS PostgreSQL	Transactional data (Order, Customer, Inventory)
Azure Cosmos DB / AWS DocumentDB	Document storage (Product Catalog with varying attributes)
Azure Cache for Redis / AWS ElastiCache	Caching, session management
Azure Event Hubs / AWS MSK / Apache Kafka	Event streaming backbone
Azure Cognitive Search / Elasticsearch	Search, product discovery
Azure Blob Storage / AWS S3	Object storage (images, documents)
Azure Synapse Analytics / AWS Redshift	Data warehousing, analytics

Table 9: Data and messaging technologies

API Standards

- OpenAPI 3.0 specification for all REST APIs
- JSON for request/response payloads
- JWT (OAuth2/OIDC) for authentication
- API versioning strategy: URL path versioning (/v1/, /v2/)
- Rate limiting and throttling via API Gateway
- Request/response logging (excluding sensitive data)
- API documentation auto-generated from OpenAPI specs

Event Schema Standards

- Avro or Protobuf for event serialization (schema registry)
- Event naming convention: {domain}.{action} (e.g., order.created, payment.completed)
- Event versioning: Include schema version in event metadata
- Dead letter queue (DLQ) for failed event processing

- Event replay capability for recovery scenarios

Non-Functional Requirements Checklist

Security Requirements

1. Zero-trust architecture: authenticate and authorize at every boundary
2. OAuth2/OIDC for authentication, RBAC for authorization
3. Encryption at rest (database encryption) and in transit (TLS 1.3)
4. Secrets management: Azure Key Vault / AWS Secrets Manager
5. WAF (Web Application Firewall) at API Gateway
6. Regular security scanning and penetration testing
7. PCI-DSS compliance for payment services

Resiliency Requirements

1. Circuit breakers to prevent cascading failures
2. Retry policies with exponential backoff
3. Graceful degradation (return cached data if service unavailable)
4. Multi-region deployment for high availability
5. Automated failover for critical services (RTO < 1 hour)
6. Chaos engineering practices for resilience testing

Observability Requirements

1. Structured logging (JSON format) with correlation IDs
2. Metrics: Prometheus-compatible metrics exposed at /metrics endpoint
3. Distributed tracing: OpenTelemetry with Azure Application Insights / AWS X-Ray
4. Centralized log aggregation: Azure Log Analytics / AWS CloudWatch
5. Alerting: PagerDuty / Azure Monitor alerts for critical issues
6. Dashboards: Grafana / Azure Dashboards for real-time monitoring

Performance Requirements (SLA and SLO)

Service	SLA	SLO (Target)
API Gateway	99.9% uptime	99.95%
Customer Identity	P95 latency < 200ms	P95 < 150ms
Product Catalog	P99 latency < 500ms	P99 < 300ms
Order Placement	P95 latency < 1s	P95 < 800ms
Search	P95 latency < 300ms	P95 < 200ms

Event Processing	P99 latency < 5s	P99 < 3s

Table 10: Service performance SLA and SLO targets

- Capacity Requirements:
    - Support 10x traffic spikes (Black Friday, Cyber Monday)
    - Auto-scaling: scale up within 2 minutes, scale down after 10 minutes stabilization
    - Database IOPS: Provision for 3x average load

Scalability Requirements

1. Horizontal Scaling: All services must be stateless and horizontally scalable
2. Database Scaling: Read replicas for read-heavy workloads
3. Caching: Cache hit ratio > 80% for product data
4. CDN: 90%+ of static content served from CDN edge
5. Batch Processing: Process 1M+ events per hour during peak

Data Governance Requirements

1. **Data Residency:** Customer data stored in country/region of residence
2. **Data Retention:** Comply with local laws (GDPR: right to erasure)
3. **Data Classification:** Sensitive data identified and protected
4. **Audit Trail:** All data access and modifications logged
5. **Consent Management:** Customer consent tracked and enforced
6. **Data Quality:** Validation rules, duplicate detection, data cleansing

5. Governance and Operating Model

Architecture Governance

Architecture Review Board (ARB)

Purpose: Ensure architectural consistency and quality across all development.

1. Composition:
    1.1. Principal Architect (Chair)
    1.2. Domain Architects (Customer, Order, Product)
    1.3. Platform Architect
    1.4. Security Architect
    1.5. Representative from Engineering Leadership
2. Responsibilities:
    2.1. Review and approve architecture designs for new services
    2.2. Maintain technology standards and reference architectures
    2.3. Approve exceptions to standards (with documented rationale)
    2.4. Conduct quarterly architecture health assessments
3. Review Process:
    3.1. Teams submit architecture design document (ADD) for new services
    3.2. ARB reviews within 5 business days
    3.3. Decision: Approved / Approved with conditions / Rejected with feedback
    3.4. All decisions documented with rationale

Service Ownership Model

1. Each service has a designated owning team (Amazon's "two-pizza team" model)
2. Teams have end-to-end responsibility: build, deploy, operate, support
3. On-call rotation for service owners
4. Team autonomy within architecture guardrails

Change Management

1. **Standard Changes:** Automated via CI/CD, no approval needed
2. **Normal Changes:** Peer review + automated tests, deploy to production
3. **High-Risk Changes:** ARB review + staged rollout + rollback plan

6. Success Metrics and KPIs

Technical Metrics

Metric	Current	Target (Year 3)
Deployment Frequency	Monthly	Daily (multiple per day)
Lead Time for Changes	Weeks	Hours
Mean Time to Recovery (MTTR)	4+ hours	< 1 hour
Change Failure Rate	~20%	< 5%
System Availability	95%	99.9%
API Response Time (P95)	1-2s	< 200ms

Table 11: Technical performance metrics

Business Metrics

Metric	Target Impact
Time to Market (New Features)	85% reduction (6 months → 2 weeks)
Customer Satisfaction (NPS)	+15 points
Conversion Rate	+10% (faster, more reliable experience)
Cart Abandonment Rate	-15% (better performance)
Infrastructure Cost per Transaction	-30% (cloud optimization)
Revenue per Developer	+50% (improved productivity)
New Market Launch Time	75% reduction (6 months → 6 weeks)

Table 12: Business impact metrics

Conclusion

This enterprise architecture modernization plan provides NovaRetail with a clear, executable roadmap to transform from a fragmented collection of country-specific platforms into a unified, cloud-native e-commerce powerhouse.

Key Success Factors

1. Executive Commitment: Sustained leadership support and investment over 3 years
2. Incremental Delivery: Value delivered continuously, not as big-bang at year 3
3. Team Enablement: Comprehensive training and support for technology adoption
4. Customer Focus: Migration transparent to customers; no service disruption
5. Flexibility: Ability to adapt roadmap based on business priorities and learnings

Next Steps

1. Weeks 1-2: Present to leadership, secure funding approval
2. Weeks 3-4: Assemble core architecture team, initiate training program
3. Weeks 5-8: Cloud infrastructure setup (Q1 Wave 1)
4. Week 9: Begin Customer Identity Service development (Q2 Wave 1)

This transformation will position NovaRetail for sustainable competitive advantage, enabling rapid innovation and exceptional customer experiences across all markets.

References

[1] Microservices.io. (2023). The Strangler Fig application pattern: Incremental modernization to services. https://microservices.io/post/refactoring/2023/06/21/strangler-fig-application-pattern

[2] AWS Prescriptive Guidance. (2025). Strangler fig pattern for application modernization. https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-decomposing-monoliths/strangler-fig.html

[3] Data-Nizant. (2025). 7 Enterprise Architecture Best Practices for 2025. https://datanizant.com/enterprise-architecture-best-practices/

[4] Maurer, F. (2025). The Evolving Landscape of Enterprise Architecture in 2025. LinkedIn. https://www.linkedin.com/pulse/evolving-landscape-enterprise-architecture-2025-florian-maurer-sqgje

[5] Emporix. (2023). Event Driven Architecture for E-Commerce. https://www.emporix.com/blog/event-driven-architecture-for-e-commerce

[6] Brink Commerce. (2024). Building Event-Driven Architectures in eCommerce. https://www.brinkcommerce.com/composable-commerce-blog/building-event-driven-architectures-in-ecommerce

Appendix A: Technology Stack Summary

1. Frontend:

    1.1. React/Next.js for web applications
    1.2. React Native for mobile apps
    1.3. Progressive Web App (PWA) capabilities

2. Backend:

    2.1. .NET 10 / ASP.NET Core for core services (primary)
    2.2. Java 17+ (Spring Boot) for core services (alternative)
    2.3. Node.js (TypeScript) for BFF and lightweight services
    2.4. Python for data processing and ML

3. Data:

    3.1. Azure Database for PostgreSQL / AWS RDS for transactional data
    3.2. Azure Cosmos DB / AWS DocumentDB for product catalog
    3.3. Azure Cache for Redis / AWS ElastiCache for caching
    3.4. Azure Event Hubs / AWS MSK / Kafka for event streaming
    3.5. Azure Cognitive Search / Elasticsearch for search

4. Infrastructure:

    4.1. Azure (primary), AWS (warm standby), GCP (cold standby)
    4.2. Kubernetes (AKS primary, EKS/GKE standby) for orchestration
    4.3. Istio for service mesh
    4.4. Terraform + Azure Bicep / AWS CDK for IaC

5. Observability:

    5.1. Azure Log Analytics / AWS CloudWatch for logging
    5.2. Azure Application Insights / AWS X-Ray for tracing
    5.3. Prometheus + Grafana for metrics
    5.4. PagerDuty / Azure Monitor for alerting

Appendix B: Estimated Budget Breakdown

Category	Year 1	Year 2	Year 3	Total
Cloud Infrastructure	$400K	$600K	$500K	$1.5M

Software Licenses	$150K	$200K	$200K	$550K
Professional Services	$300K	$200K	$100K	$600K
Training and Enablement	$200K	$150K	$100K	$450K
Additional Staff	$500K	$800K	$800K	$2.1M
Contingency (15%)	$232K	$292K	$255K	$779K
Total	$1.78M	$2.24M	$1.96M	$5.98M

Table 13: 3-year budget estimate

ROI Projection: Estimated 18-month payback period through:

- Infrastructure cost reduction: $800K annually
- Improved developer productivity: $1.2M annually (opportunity cost)
- Revenue increase from better conversion: $2M+ annually