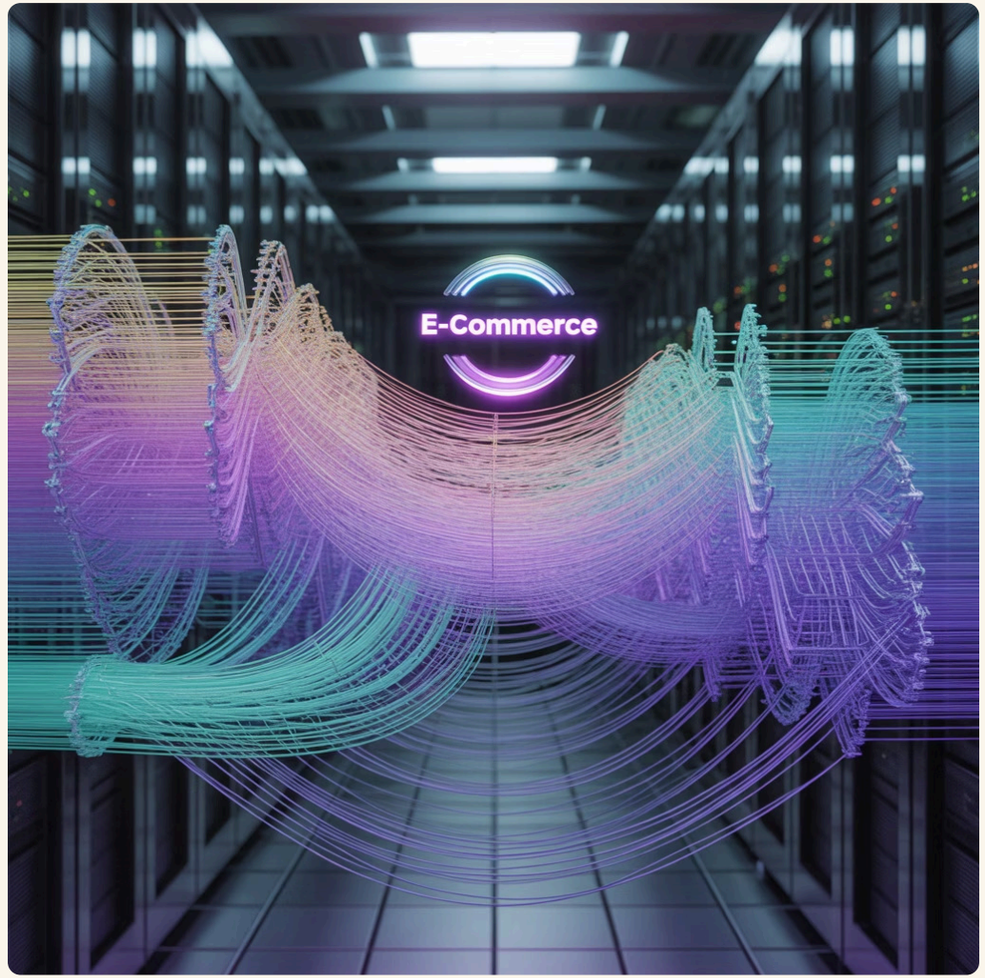
From Monolith to Microservices: E-commerce System Redesign

In this presentation, we'll explore how to transform a tightly-coupled e-commerce monolith into a flexible, scalable microservices architecture. We'll focus on redesigning the core entities—Customer, Product, Order, and Payment—into independent services that can evolve separately while working together seamlessly.

The Monolithic Challenge

Current Architecture Limitations

- All business logic and data stored in a single codebase
- Shared database creates tight coupling between entities
- Any change requires testing the entire application
- Scaling requires duplicating the entire system



A payment processing bug in a monolith can bring down the entire shopping experience, even if product browsing is unaffected.

Microservices: Modern E-commerce Architecture

Loosely Coupled

Each service represents a distinct business capability with clear boundaries and responsibilities

Independently Deployable

Teams can develop, test, and deploy services without affecting others

Failure Isolation

Issues in one service don't cascade to others; system degrades gracefully

Services communicate through well-defined APIs (REST/gRPC) for synchronous requests or message buses for asynchronous events.

Microservice 1: Customer Service

Core Identity

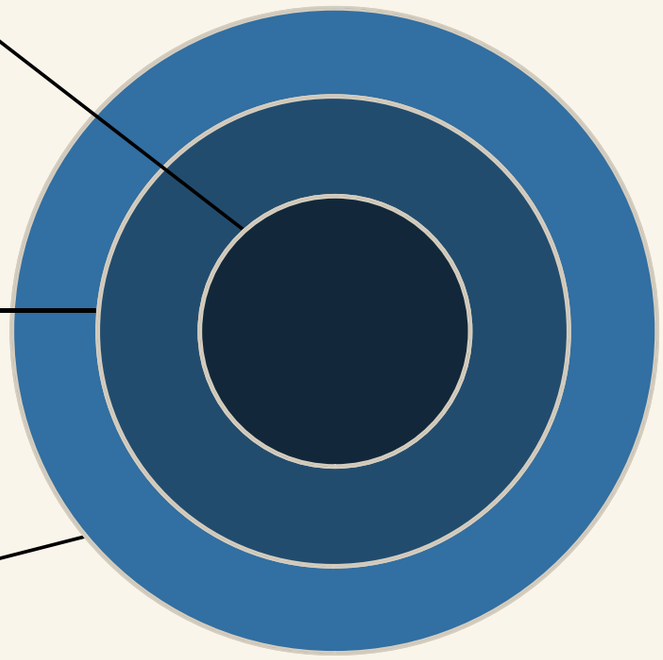
Central identity management service.

Profile Management

User profiles and data handling.

Integration Points

Authentication and user preferences.



Responsibilities

- User registration and authentication
- Profile management and preferences
- Address book and payment methods storage

Technical Aspects

- Dedicated customer database (PostgreSQL)
- JWT-based authentication service
- GDPR compliance for personal data handling

Microservice 2: Product Service

Core Functions

- Product catalog management
- Pricing strategies and discounts
- Real-time inventory tracking
- Category and search functionality

Technical Benefits

- Optimized for high-volume reads
- Caching layer for performance
- Elastically scales during sales events

Microservice 3: Order Service

Order Creation

Captures customer selections and validates product availability

Order Management

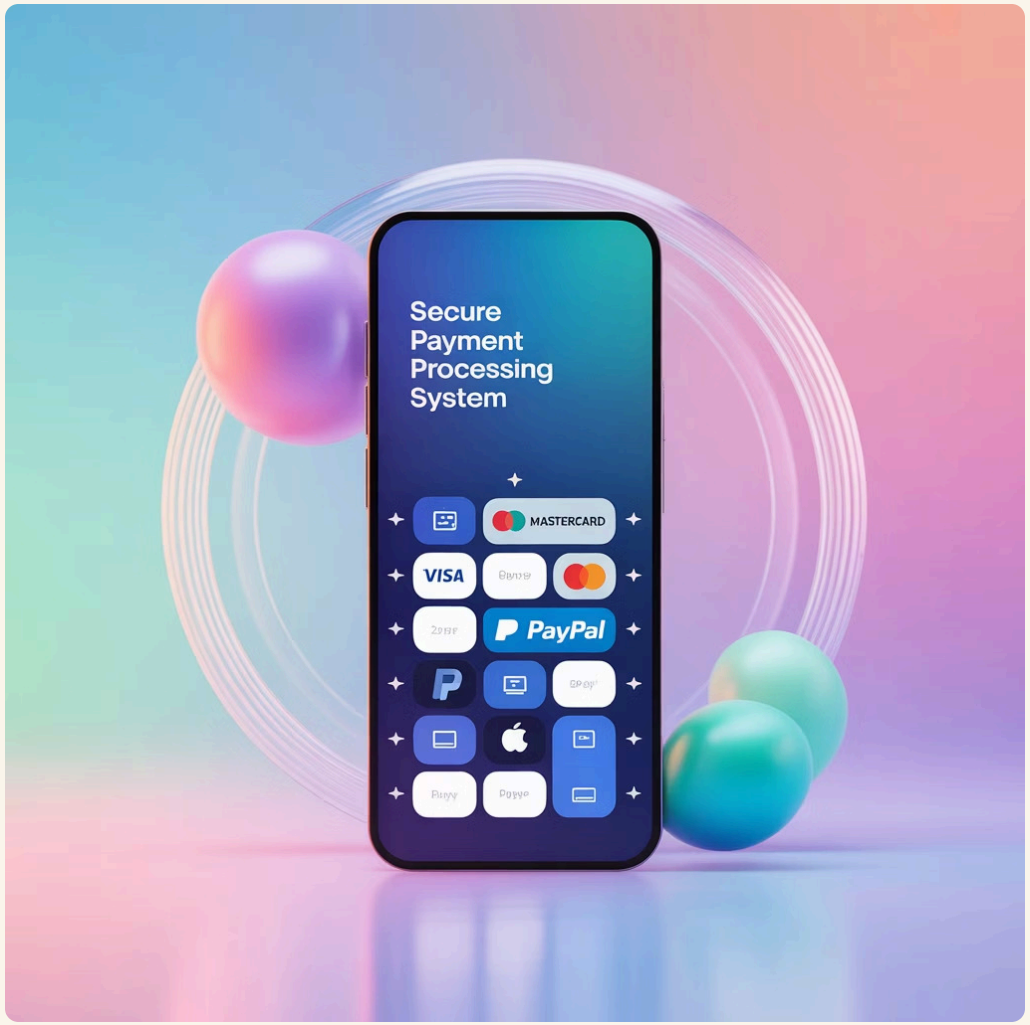
Tracks status changes and maintains order history

Fulfillment Coordination

Interfaces with shipping and inventory systems

The Order Service maintains its own database of orders but references customer and product IDs from other services.

Microservice 4: Payment Service



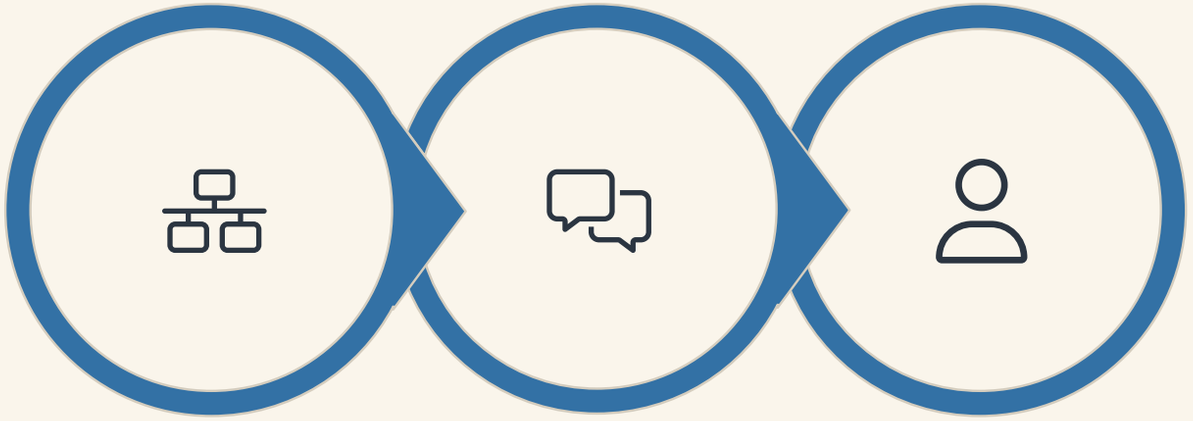
Security-First Design

- PCI DSS compliant architecture
- Tokenization of sensitive payment data
- Encryption for all transactions

Integration Capabilities

- Multiple payment gateways (Stripe, PayPal, etc.)
- Fraud detection systems
- Reconciliation with accounting systems

Inter-Service Communication Patterns



Synchronous (Request/Response)

REST or gRPC APIs for immediate operations like "Get Product Details" or "Process Payment"

Asynchronous (Event-Driven)

Kafka/RabbitMQ for events like "OrderPlaced" or "PaymentReceived" that multiple services need to know about

API Gateway

Handles authentication, routing, rate limiting, and serves as single entry point for external clients

Example Flow: Placing an Order

1

User Interaction

Customer selects products and submits order through UI/mobile app

2

Order Creation

Order Service validates product availability with Product Service and creates order with "Pending" status

3

Payment Processing

Order Service calls Payment Service to process transaction using stored payment method

4

Event Broadcasting

Payment Service emits "PaymentReceived" event to message bus

5

Service Updates

Order Service updates status; Product Service decrements inventory; Customer Service records purchase

Microservices Benefits for E-commerce

Fault Isolation

Payment processing issues won't prevent customers from browsing products or adding to cart

Independent Scaling

Product catalog can scale during Black Friday while other services maintain normal capacity

Technology Diversity

Order Service in Java for transaction handling; Product Search in Node.js for performance