Capstone Project

Case Study

Project Overview: ShopSphere E-commerce Platform

Business Context

"**ShopSphere**," a rapidly expanding e**-commerce retailer**, specializes in personalized, **high-value consumer goods** (*e.g., custom apparel, luxury electronics*).

The company **operates three geographically dispersed fulfillment centers** (*FCs*) **across different regions**.

Due to **regional data residency laws** (*e.g., GDPR, CCPA*) and the sheer volume of global traffic, ShopSphere **requires a decentralized architecture**.

The **core application logic** and **primary inventory systems** must **reside within regional data centers** (*DCs*) to **meet legal compliance** and **provide low-latency** experiences to **local customers**.

The **current legacy e-commerce system suffers from**:

- **Data Silos: Inventory data**, **user profiles**, and **order history** are **isolated** in **separate**, **inflexible databases**.
- **Scalability Bottlenecks:** The **monolithic architecture cannot handle Black Friday/holiday traffic spikes** without crashing.
- **High Downtime Risk: Deployments require full system restarts**, leading to **unacceptable downtime**.

The Challenge: The "ShopSphere Core" Platform

You are tasked with **designing** the **architecture** for "**ShopSphere Core**," a **new**, **decentralized platform** that **manages user traffic**, **product catalogs**, and **order processing**.

The **platform** must **operate primarily out of three separate Regional Data Centers** (*RDCs*), each supported by an **additional**, compliant **Disaster Recovery (*DR*) Data Center** within its **same legal jurisdiction** and a **Central Corporate Data Center** (*HQ-DC*) for **global reporting**.

The **core constraint** is achieving **near-global availability (*99.95%*)** and **low latency (*sub-100ms*)** for all **regional customers** while ensuring **data residency and localized compliance**.

Business Scenario and Constraints

1. The Company and Industry

ShopSphere is a **high-growth retailer** in the **personalized goods sector.**

- **Core Products: Custom fashion**, **premium gadgets**, and exclusive **digital subscriptions**.
- **Key Driver: Customer Experience** and **Conversion Rate**. **Every second of latency costs millions in lost sales**.
- **Business Goal: Achieve 99.95% Availability** and **increase Conversion Rate by 10%** through **high-speed**, **localized user experience**.

2. Operational Environment (The Decentralization Constraint)

ShopSphere operates **three primary Regional Pair Data Centers** (*RDC-A, RDC-B, RDC-C*) and one **centralized HQ-DC**.

Region	Primary Data Center (RDC)	Compliant DR Data Center (DR-DC)	Jurisdiction
Region 1 (Europe)	RDC-A: Frankfurt, Germany	DR-DC-A: Dublin, Ireland	European Union
Region 2 (North America)	RDC-B: Ashburn, USA	DR-DC-B: Chicago, USA	United States
Region 3 (Asia-Pacific)	RDC-C: Singapore	DR-DC-C: Sydney, Australia	APAC Region
Global Reporting	HQ-DC: New York, USA		Central Management

- **Data Residency Mandate:** Core **customer personal data**, specific **regional payment logs**, and **fulfillment data must reside within** the respective **RDC's physical boundaries** (*e.g., EU data must stay in RDC-A/DR-DC-A jurisdiction*). **Public cloud deployment** is **permissible only** for **CDN/Static Assets**, but **not for core transactional data**.
- **Active-Passive DR Mandate:** The **three DR-DCs** are intended for **Active-Passive compliant failover**. They must be **kept running** and **capable** of **taking over** the **primary RDC's transactional workload within 15 minutes** to maintain the **99.95% availability target** for that **region**.
- **RDCs (Regional Hubs): Each RDC** must **operate independently** (*active-active deployment*) to **serve** its **local customers with 100ms latency**. **Each RDC** has a **fixed resource limit** (*e.g., 8 racks, 150kW power*).
- **Global Network:** The **WAN links** between **RDCs** and the **HQ-DC** are **high-speed** but have **inherent global latency** (*e.g., 200ms*).

- **Existing Infrastructure:** The **company uses Kubernetes** for **orchestration**, **standardizes on Linux OS**, and has **existing Global DNS** and **CDN providers** that must be integrated.

3. Data Challenge

Data Source	Type	Volume/Velocity
Clickstream/Search Events	Small, structured **JSON events**	Extremely High Velocity (**500,000 events/sec total**)

4. Critical Technical Problems to Solve

Problem	Required Solution Implication
Data Residency & Low Latency	**Core transactional data** and **application services** must be **geographically dispersed** and **operate independently**.
High Availability (99.95%)	The **platform** must **withstand** a **total failure** of a **primary RDC** by rapidly **failing over** to its designated **DR-DC.**
High Traffic Spikes (Black Friday)	The **platform** must **scale rapidly** to handle **10x normal load using** the existing **Kubernetes cluster** resources.
Transactional Integrity (Payments)	**Financial records** must be **100% accurate** despite asynchronous communication between microservices.
Real-Time Personalization	Processing **user behavior** data must **deliver personalized product recommendations** within **50ms** of page load.

3. The Assignment: 12 Detailed Tasks

Section 1: Strategic Mapping

- **Assignment 1: Business Vision to Technical Vision.**
- **Assignment 2: Functional & Non-Functional Requirements.**

Section 2: Architectural Selection

- **Assignment 3: Select Paradigm.**
- **Assignment 4: Select Model.**
- **Assignment 5: Select Architecture Style.**
- **Assignment 6: Select Architecture Pattern.**

Section 3: Technical Design & Flow

- **Assignment 7: High-Level Design (HLD).**
- **Assignment 8: Low-Level Design (LLD).**
- **Assignment 9: Component & Service Selection.**
- **Assignment 10: Create 3 ADRs (Architectural Decision Records).**

Section 4: Visualizing the Flow

- **Assignment 11: System Flow of Single Clickstream Event**
- **Assignment 11: Final Architecture Picture.**

Model Answer

1. Business Vision to Technical Vision

Business Vision	Technical Vision
Increase Conversion Rate by 10% via high-speed, localized user experience.	• Implement a **Global Server Load Balancing (GSLB)** strategy **combined** with an Edge **Caching Layer (CDN)** to **route users** to the **nearest Regional Data Center** (RDC) and **serve content** with **100ms latency** and **system** must handle **500,000 reads/second** (IOPS) to manage traffic.
Achieve 99.95% Availability and recover from total RDC failure to prevent revenue loss during critical traffic spikes (e.g., Black Friday).	• Design an **Active-Passive Compliant Failover Strategy** for each RDC/DR-DC pair, services must be fully restored (**RTO**) in **<=15 minutes**. • This involves maintaining **data replication** of to the dedicated DR-DC **within the same legal jurisdiction**. • **Design** a highly **resilient, fault-tolerant deployment** across all **three RDCs**, for rapid, resource-efficient **horizontal scaling** and isolated failure domains.
Ensure Data Residency and regulatory compliance (GDPR, CCPA) for customer data.	• I**mplement** a **Geographically Partitioned Data Strategy** where **core customer** and **transactional data** is **strictly confined** to and processed within its designated **RDC** and **transactional data loss** must be minimal, defined by an **RPO of <=5 minutes**, (***due to inheritance WAN latency***) and must **guarantee 100% of sensitive records** with **no cross-region data replication** for compliance.
Central Oversight Provide centralized global business reporting and analytics.	• **Anonymized Event Streaming.** • Establish a **Central HQ-DC** that consumes only **anonymized** and **aggregated** metrics, streamed asynchronously from all regions.

2. Functional and Non-Functional Requirements (FRs & NFRs)

Core Functional Requirements (FRs)

1. **Geo-Aware Routing & Catalog Display:** The platform must **use Global DNS (GSLB) to route users** to the **nearest RDC** and **dynamically display** the **regional Product Catalog** and **localized pricing** with a **latency** of **100ms**.
2. **Idempotent Order Transaction:** The **system must process** an **order transaction** with **ACID compliance within the local RDC**, ensuring **zero double-charges** or **lost orders**, and **commit** the **final financial record before triggering downstream fulfillment processes**.
3. **Real-Time Search & Recommendations:** The **platform** must **reliably ingest** and **process** user **Clickstream and Search Events** and **use** the **resulting personalized data** to **deliver product recommendations** on **core user pages** within **50ms** of **page load**.

Critical Non-Functional Requirements (NFRs)

1. **Availability** (High Uptime & DR): The platform must maintain **99.95% Availability** across all **core e-commerce services** (*Order, Catalog, User*). In the event of a **total primary RDC failure**, the system must guarantee a **Recovery Time Objective (RTO) of 15 minutes** for that region via **automated failover** to its dedicated, **compliant DR-DC**.
2. **Data Residency** (Compliance): All customer **Order Transaction Data** and **personally identifiable information** (*PII*) **must be strictly** and **physically partitioned**, **processed**, and **stored only** within its **respective Regional Data Center** (*RDC/DR-DC*), meeting the **mandatory regulatory requirements**.
3. **Scalability** (Traffic Spikes): The platform must **achieve horizontal scaling** of **critical services** (*Order and Catalog*) by **5x within 3 minutes** using the existing **Kubernetes cluster resources** in **each RDC** to **accommodate sudden**, high-volume **traffic events**.
4. **Latency** (User Experience): The **end-user HTTP response time** for **core transactional pages** (*e.g., Checkout, Product Detail Page*) must **not exceed 100ms** for **95% of requests** (*95*), primarily **achieved through Edge** and **Service-level caching**.
5. **Data Integrity** (RPO): The **transactional data cluster** within **each RDC/DR-DC** pair must maintain a **Recovery Point Objective (***RPO***) of 5 minutes** to minimize **financial data loss** during a compliant failover to the DR-DC.

3. Select Paradigm:

Paradigm Selection	Justification
Event-Driven Programming (EDP)	EDP is the **most suitable mechanism** and programming style for the **high-volume**, **asynchronous**, and **decentralized nature** of the ShopSphere **platform**.

4. Select Model:

a. Domain Model

The **Domain Model** for the ShopSphere E-commerce Platform is a conceptual blueprint that formally defines the **relationships, behaviors, and attributes** of its core business concepts, including:

Model Component	Example
Data Entities	• **User** • **Order** • **Product** • **Inventory** • **Payment** • **Shipment**
Relationships	• **User** *places* **Order** (1: Many): • **Order** *contains* **Products** (Many: Many) • **Order** *linked to* **Payment** (1:1)
Attributes	**Order**: • **status** • **total_amount** • **shipping_address** **Product**: • **price** • **color** • **description** **User**: • **user_id** • **Email** • **residency_region**
Behavior/ Processes	• **PlaceOrder()** • **ReserveInventory()** • **ProcessPayment()** • **GenerateRecommendations()** • **CompensateOrder()**

b. Process Model

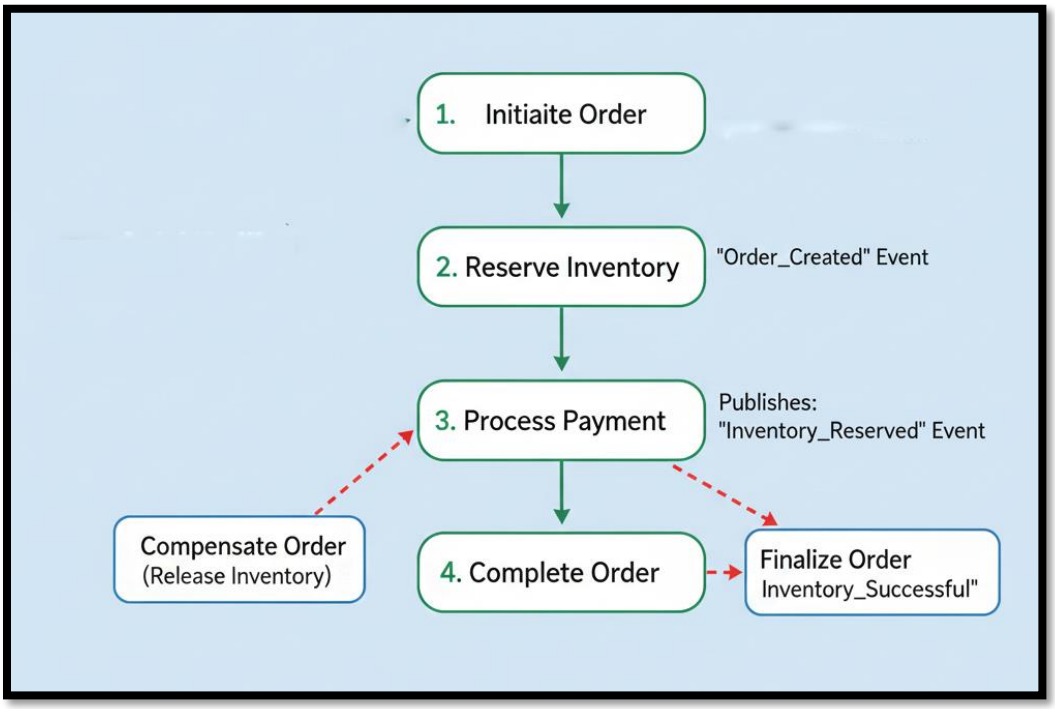
i. Order Flow

Step	Action	Path Type	Details & Business Logic
1. Initiate Order	**Initiate Order**	Synchronous	• The transaction begins. • This step typically involves the Order Service receiving the request, creating an initial record in its local database.
2. Reserve Inventory	**Reserve Inventory**	Synchronous / Event-Triggered	• This step is triggered by the "Order_Created" Event. • Catalog/Inventory Service consumes the event and attempts to logically deduct or reserve the necessary stock in its local database. • This prevents overselling while the final payment is processed.
3. Process Payment	**Process Payment**	Synchronous	• This is the most critical step. • It involves the Order Service calling an external Payment Gateway to charge the customer. • Upon successful inventory reservation, this step proceeds.
4. Complete Order	**Complete Order**	Synchronous	• This step is triggered upon successful payment processing and inventory reservation. • The Order Service updates its local record to a final status like PAID or COMPLETED.

ii. Failure and Success Paths

Path Type	Trigger Point	Action / Event Triggered	Service(s) Involved	Outcome and Rationale
Success Path	**Successful Process Payment**	Finalized Order (Inventory Successful)	Catalog/Inventory Service	**Final Commit:** • The permanent deduction of inventory is performed. • The Order Service updates the order status to COMPLETED. • Transactional integrity is preserved successfully.

Compensation Path (Rollback)	**Failure during Process Payment** (e.g., card declined)	Compensate Order (Release Inventory)	Catalog/Inventory Service	**Reversal/Consistency**: • This is the Compensating Transaction. • The service releases the reserved inventory back to the available stock. • This prevents inventory from being tied up by a failed order, maintaining data consistency.



5. Select Architectural Style:

Architecture Style Selection	Justification
Dominant Style: Microservices Architecture (MSA)	• MSA is the **mandatory** style for the platform, providing the necessary boundaries for decentralization, scaling, and failure isolation.
Complementary Styles: Event-Driven Architecture and Layered Architecture	• These styles are integrated to manage communication and provide internal structure, respectively.

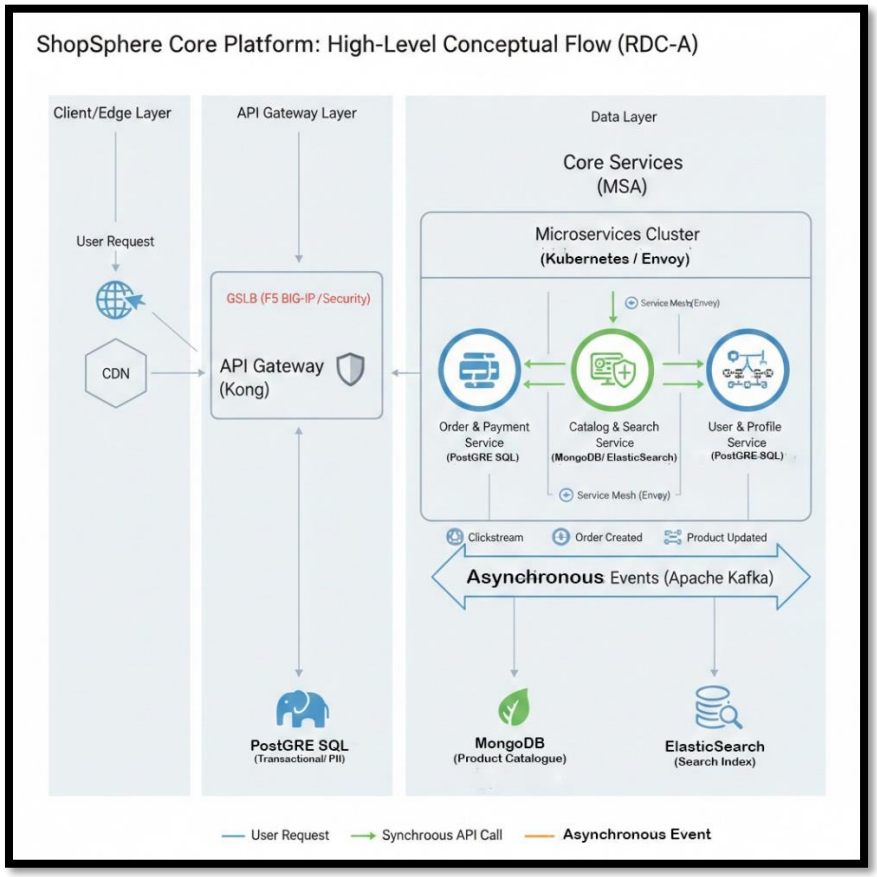
6. Select Architectural Pattern:

Pattern Selection	Justification
Saga Pattern (Choreography)	• **MANDATORY for Transactional Integrity.** • The Saga Pattern is essential to ensure **ACID-like certainty** for the **Order Transaction across** the **many decoupled microservices**.
Command Query Responsibility Segregation (CQRS)	• **MANDATORY for Performance and Scalability.** • CQRS **separates** the application's data models for **read (query)** operations from **write (command)** operations.
Circuit Breaker Pattern	• **CRITICAL for System Resilience and Availability.** • The Circuit Breaker **wraps calls to external** or **unreliable services**, **preventing cascading failures**.
Observer Pattern (via Message Broker)	• **CRITICAL for Real-Time State Notification.** • This behavioral pattern **allows services** to **automatically receive updates** when the state of an **observed entity change**.

7. High-Level Design (HLD): Conceptual Flow

The design is based on a standard **Event-Driven, Layered MSA**.

1. **Client/Edge Layer:** User requests enter via CDN → GSLB.
2. **API Gateway Layer:** Traffic is routed through the API Gateway (*Kong*) for policy enforcement (*rate limit*).
3. **Core Services Layer (MSA):** Requests are handled by the microservices (*Order, Catalog, User*) running on Kubernetes.
4. **Event/Communication Layer:** Services communicate asynchronously via Apache Kafka.
5. **Data Layer:** Polyglot Persistence (*PostgreSQL, MongoDB, Elasticsearch*) handles durability and optimized reads.



8. Low-Level Design (LLD) for Data Storage

a. PostgreSQL LLD (Order Transactions & PII)

PostgreSQL is chosen for its **ACID compliance** and mandatory **Data Residency** requirements.

i. Logical Design (Schema Details)

The **design** must **enforce data integrity** and efficiently **handle transactional workloads**.

Entity	Key Attributes	Data Type	Constraint	Rationale
Orders	order_id	UUID	Primary Key (PK)	Unique, immutable identifier
	user_id	UUID	Foreign Key (FK)	Links to the User table for compliance checks.
	status	ENUM	NOT NULL	Tracks state (CREATED, PAYMENT_PENDING, COMPLETED, COMPENSATED).
	total_amount	NUMERIC(10, 2)	NOT NULL	Financial integrity.

Users	user_id	UUID	Primary Key (PK)	Unique PII identifier.
	first_name	VARCHAR(50)	NOT NULL	Personal data for shipping/compliance.
	residency_region	VARCHAR(10)	NOT NULL	**Mandatory field** for Data Residency checks.
	created_at	TIMESTAMP	NOT NULL	Auditing.

ii. Physical Deployment (RDC-A and DR-DC-A)

The **deployment must** ensure **HA and compliant DR**.

Deployment Aspect	RDC-A (Active)	DR-DC-A (Passive)	Constraint Met
Cluster Topology	**Primary Master** + 2 **Synchronous Replicas**	**One Standby Replica** (Promotable)	for RDC failure.
Replication Mode	**Synchronous Streaming** (within RDC-A)	**Asynchronous Streaming** (RDC-A to DR-DC-A)	**Zero Data Loss** within RDC-A. **RPO** for DR, adhering to **Data Residency**.
Storage	High-performance SSD/NVMe attached volumes.	Standard SSD/EBS storage.	Low-latency I/O for high-volume transactions.
Failover	**Automated failover** managed by **Patroni** or similar cluster manager within RDC-A.	**Manual/Semi-automated promotion** of the standby replica.	**High Availability (HA)**.

9. Select Component, Feature, Service:

A. Critical Business-Logic Services (MSA Components per RDC/DR-DC)

These services are deployed via Kubernetes clusters in both the **RDC** (Active) and **DR-DC** (Passive).

Component/ Service	Primary Purpose	Key Technologies Used	Key Features
User & Profile Service	Manages core customer PII, authentication/ authorization, session state, and clickstream ingestion.	• **AuthN & AuthZ:** Keycloak. • **RDBMS:** PostgreSQL. • **Session:** Redis. • **Messaging:** Apache Kafka	**Authentication/Authorization:** • Implemented using **OAuth 2.0 / OpenID Connect (OIDC)**. • Manages **user sign-up, login**, and the **issuance** of JSON Web Tokens **(JWTs)**. **Profile Management:** • Ensures sensitive **PII resides only within the local RDC/DR-DC**

			jurisdiction using the clustered **PostgreSQL**. **Session Management:** • Application services (*e.g., User, Order, Catalog*) do not save any user-specific data (session information) locally. • All session data is stored externally in the centralized Redis cluster. • Microservices fetch the required session details from Redis for every request. **Clickstream Publishing:** • **Captures** raw **user activity** and **publishes** them directly to the local RDC's **Kafka Cluster** for high-throughput **processing**.
Catalog & Search Service	Provides low-latency product discovery and manages inventory visibility	• **Cache:** Redis. • **Search:** Elasticsearch. • **Database:** MongoDB.	**Localized Catalog Delivery:** • Serves regional product data from the **NoSQL database** and **Redis Cache** (*100ms NFR*). **Search Index Management:** • Maintains a dedicated, optimized **Elasticsearch Index** (*CQRS Read Model*) for ultra-fast query responses. **Flexible Schema for Products:** • Utilizes the flexible schema of the **NoSQL database** to manage diverse product attributes and variants.
Order & Payment Service	Manages the transaction lifecycle, ensuring financial integrity and compliance	• **RDBMS:** PostgreSQL • **Messaging:** Apache Kafka	**Idempotent Transaction Core:** • Commits the financial record to the **PostgreSQL** (*local RDC*) with **ACID compliance**. **Saga Coordination:** • **Publishes** and subscribes to **events** on the **Kafka** manage the **distributed Order Saga**.

B. Critical Infrastructure/Data Components (Foundational Support)

Component/ Service	Role in Architecture	Key Technologies Used	Key Features and Constraint Fulfillment
Global Routing & CDN Layer	Manages ingress, traffic routing, and edge performance.	• **GSLB:** Akamai Global Traffic Management (GTM) • **CDN:** Cloudflare	**Global Server Load Balancer (GSLB):** • **Directs users** to the **nearest**, healthiest **RDC** based on **geo-proximity**. **CDN:** • **Caches static assets** at the **edge**, supporting the 100ms latency NFR.
Data Persistence Cluster (RDC-DR Compliant)	Provides storage and implements the **Active-Passive DR strategy**.	• **RDBMS Replication:** PostgreSQL Streaming Replication (*support both Synchronous and Asynchronous replication*) • **NoSQL:** Redis Cluster	**Asynchronous DR Replication:** • Uses **PostgreSQL Streaming Replication** to copy transactional logs from **RDC → DR-DC** for **compliant failover** (*RPO 5min*). **RDBMS Cluster (HA):** • Uses **synchronous replication within** the **primary RDC** for instant HA.
Local Load Balancing & API Gateway	Manages internal traffic distribution, security, and external API exposure within each RDC.	• **Load Balancer:** F5 Load Balancer • **API Gateway:** Kong	**L7 Load Balancer:** • Distributes traffic to the **API Gateway** instances. **API Gateway:** • Enforce **Authentication, Rate Limiting, and Throttling** (*e.g., using Redis for counters*) to protect internal services during traffic spikes.
Message Broker Cluster	Provides the asynchronous communication backbone.	• **Messaging:** Apache Kafka	**Durable Messaging:** • Guarantees message delivery, **enabling** the **EDA/Saga pattern**. **High Throughput:** • Handles high-velocity **Clickstream data ingestion** without failure.

C. Critical Central HQ-DC Components (Global Management & Reporting)

Component/ Service	Role in Architecture	Key Technologies Used	Key Features and Constraint Fulfillment
Global Observability Hub	Centralized tooling for technical operations and management.	• **Monitoring**: Prometheus & Grafana • **Logging**: ELK Stack (Elasticsearch, Logstash, Kibana)	**Global Health Dashboard**: • Provides a **single**, **unified** view of the **system's operational health**, **latency**, and resource **utilization across all RDCs** and **DR-DCs**.

10.Architectural Decision Records (ADR):

a. ADR 001: Adopt CQRS for Catalog & Search Service

Decision	**The Catalog & Search service will utilize CQRS, separating transactional (MongoDB Write Model) from search (Elasticsearch Read Model) data stores.**
Status	Accepted
Context	Need to achieve 100ms query latency and high throughput while maintaining complex product data integrity.
Consequences	• **Pros:** Optimizes the read path for speed and complex queries; search engine failure does not impact transactional writes. • **Cons:** Introduces eventual consistency between the Write and Read models, managed via a dedicated Kafka pipeline.

b. ADR 002: Adopt PostgreSQL as the Transactional System of Record

Decision	**PostgreSQL is selected as the mandatory database for the Order & Payment Service and PII storage within the User & Profile Service.**
Status	Accepted
Context	• Financial records and PII require strict durability and transactionality (ACID compliance). • Furthermore, robust, open-source **Streaming Replication** is required for compliant DR.
Consequences	• **Pros:** Guaranteed transactional integrity and consistency. Robust, proven mechanism for compliant RPO 5min replication to the DR-DC. • **Cons:** Less flexible schema than MongoDB; requires significant operational expertise for maintenance and tuning at high scale.

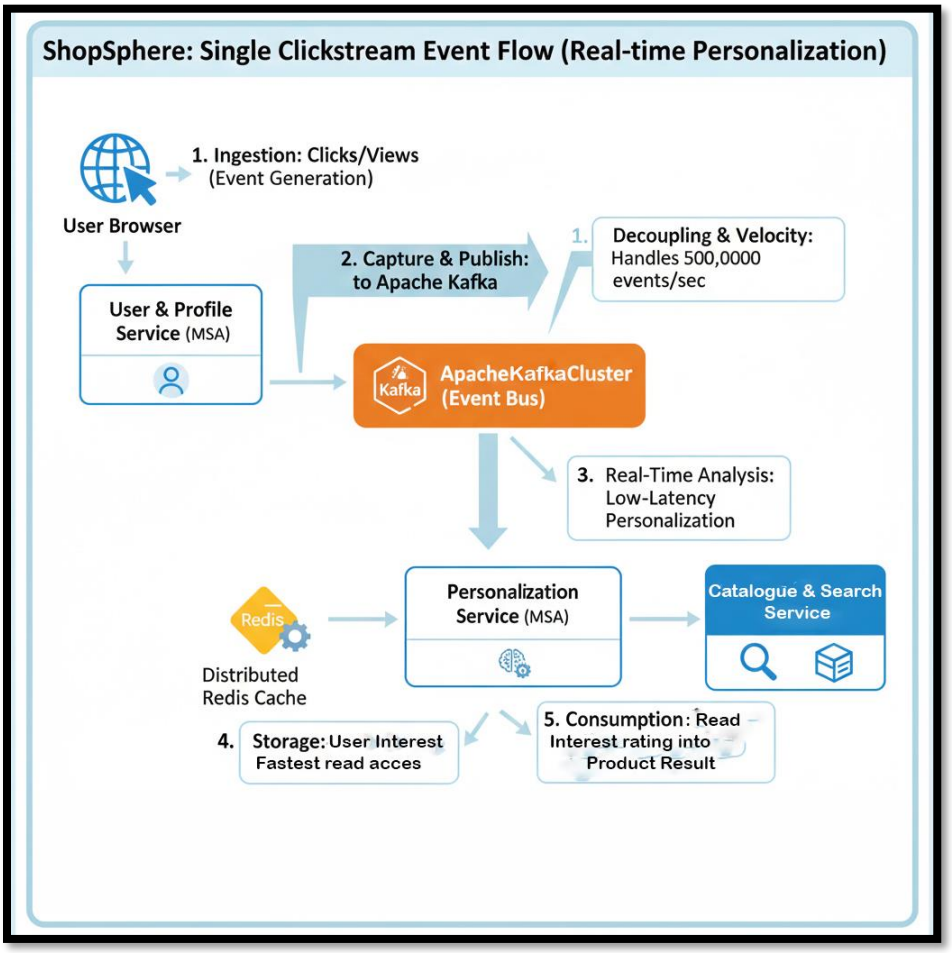
c. ADR 003: Adopt MongoDB as the Primary Catalog Write Store

Decision	**MongoDB is selected as the primary database for the Catalog Service's Write Model, leveraging the CQRS pattern.**
Status	Accepted
Context	The **Product Catalog** involves complex, highly varied, and frequently changing product attributes which do not fit a rigid relational schema.
Consequences	• **Pros:** Provides necessary **flexible schema** for complex e-commerce product variants. High horizontal scalability for the Catalog's data volume. • **Cons:** Eventual consistency between MongoDB (Write) and Elasticsearch (Read) must be carefully managed via Kafka. Cannot be used for ACID financial records.

11.System Flow: Single Clickstream Event

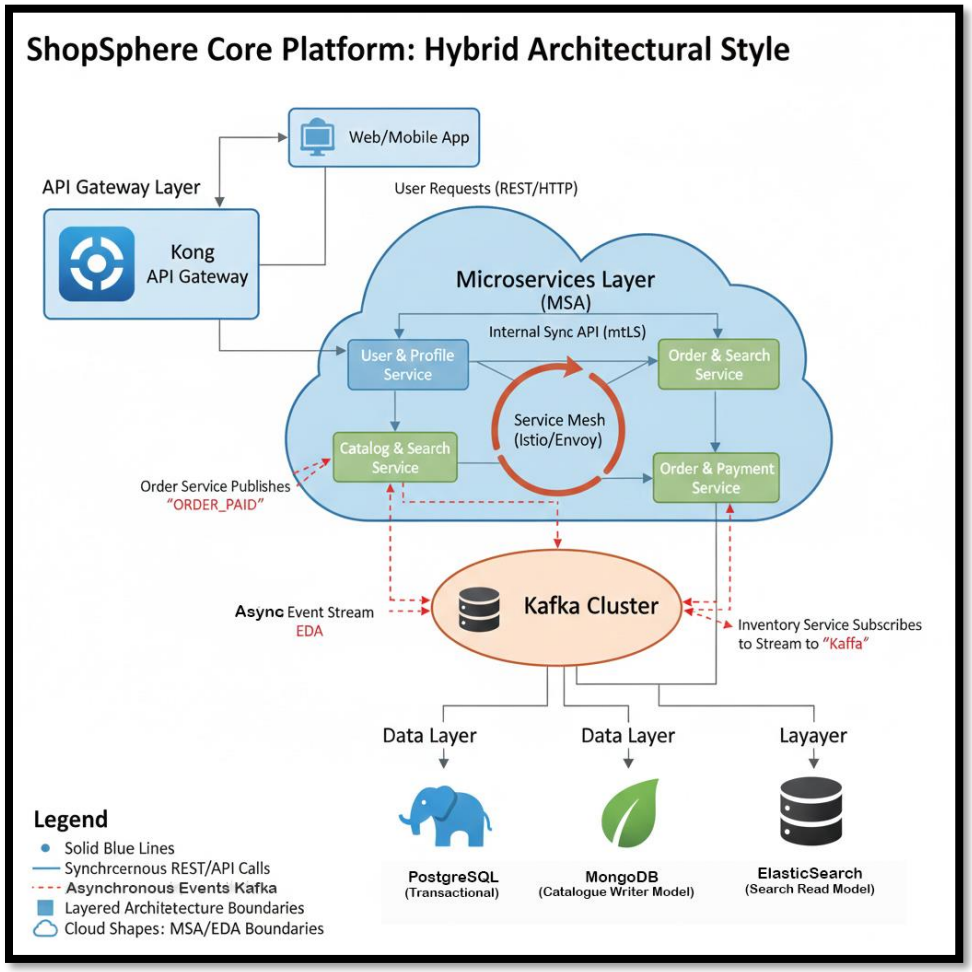
Step	Component	Action/Technology	Rationale and Constraint
1. Ingestion	**User Browser**	**Clicks/Views (*Event Generation*):** The **user interacts** with the **application** (*e.g., product view, search action*)	**Generates** the **raw event data** at the origin
2. Capture & Publish	**User & Profile Service**	**Ingests event** → Immediately publishes raw event to **Apache Kafka** via a high-throughput producer API.	**Decoupling and Velocity:** Kafka handles 500,000 events/sec velocity.
3. Processing	**Personalization Service (MSA)**	**Consumes** the **event from Kafka** → **Applies stream processing rules** (*e.g., aggregates events, calculates user interest rating*).	**Real-Time Analysis:** Uses Event Driven Programming to **process data instantly** for low-latency personalization.
4. Storage	**Personalization Service**	**Stores** the updated **user interest rating** in the **Distributed Redis Cache**.	**Speed: Redis provides** the **fastest** possible **read access** for the **recommendation engine**.
5. Consumption	**Catalog & Search Service**	**Reads** the **interest rating from** the **Redis Cache** during a search/view request.	**Low Latency (100ms):** Uses the in-memory cache to retrieve personalization data

			instantly to integrate into the product results.



12.Final Architecture Picture: Decentralized Multi-Region

The final architecture synthesizes all components and communication strategies, demonstrating **Data Residency**, **Scalability**, and **Resilience** across the three operational sites (RDC, DR-DC) and the central hub (HQ-DC).


ShopSphere Core Platform: Hybrid Architectural Style
Web/Mobile App
API Gateway Layer
User Requests (REST/HTTP)
Kong
API Gateway
Microservices Layer
(MSA)
Internal Sync API (mtLS)
User & Profile Service
Order & Search Service
Service Mesh
(Istio/Envoy)
Catalog & Search Service
Order & Payment Service
Order Service Publishes
"ORDER_PAID"
Async Event Stream
EDA
Kafka Cluster
Inventory Service Subscribes
to Stream to "Kaffa"
Data Layer
Data Layer
Layayer
PostgreSQL
(Transactional)
MongoDB
(Catalogue Writer Model)
ElasticSearch
(Search Read Model)
Legend
Solid Blue Lines
Synchrcernous REST/API Calls
Asynchronous Events Kafka
Layered Architetecture Boundaries
Cloud Shapes: MSA/EDA Boundaries