Comprehensive Design Document

Real-Time Shipment Tracking Microservices Platform

Domain-Driven Design Architecture

Version: 1.0

Date: December 2024

Classification: Technical Architecture Document

Table of Contents

1. Executive Summary

1.1 Purpose

This document presents the comprehensive technical design for a real-time shipment tracking platform built using Domain-Driven Design (DDD) principles and microservices architecture. The system provides end-to-end visibility of shipments across multiple transportation modes.

1.2 Key Business Objectives

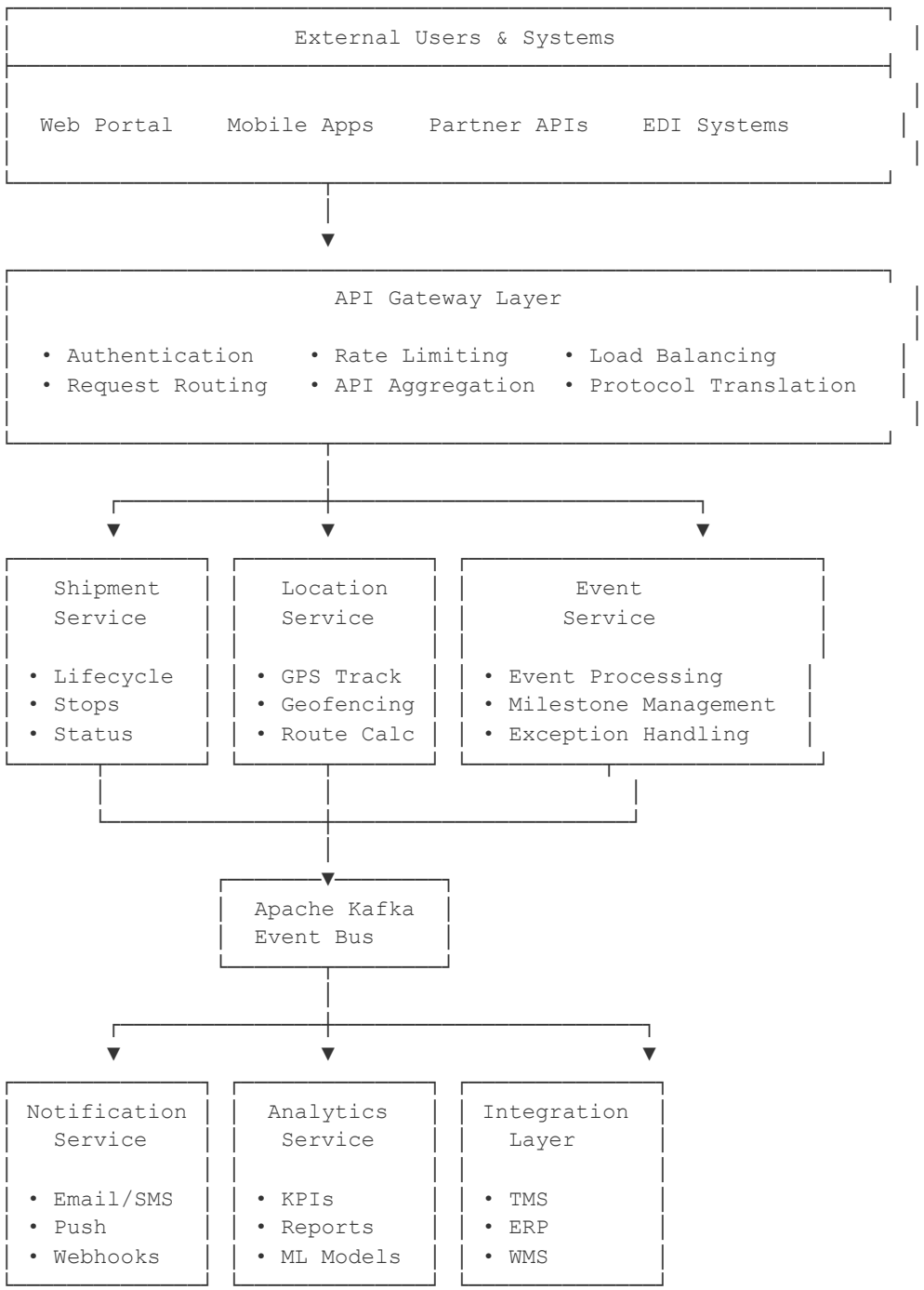
- **Real-time Visibility:** Track shipments in real-time across the supply chain
- **Event-Driven Notifications:** Proactive alerts for stakeholders
- **Analytics and Insights:** Performance metrics and predictive analytics
- **Scalability:** Support millions of concurrent shipments
- **Reliability:** 99.9% uptime SLA

1.3 Technical Approach

- **Domain-Driven Design:** Clear bounded contexts and business logic encapsulation
- **Microservices Architecture:** Independent, scalable services
- **Event Sourcing:** Complete audit trail and event replay capabilities
- **CQRS Pattern:** Optimized read and write models
- **Cloud-Native:** Containerized, orchestrated deployment

2. System Overview

2.1 High-Level Architecture



2.2 Technology Stack

Layer	Technology	Purpose
Language	Java 17	Primary development language
Framework	Spring Boot 3.2.0	Microservices framework
Messaging	Apache Kafka	Event streaming platform
Database	PostgreSQL 15	Relational data persistence
Cache	Redis 7	In-memory caching
Container	Docker	Application containerization
Orchestration	Kubernetes	Container orchestration
API Docs	OpenAPI 3.0	API documentation
Monitoring	Prometheus/Grafana	Metrics and monitoring

3. Domain-Driven Design

3.1 Domain Model Overview

> ### Core Domain: Shipment Management
>
> - Shipment Lifecycle Management
> - Stop Management and Sequencing
> - Status Transitions and State Machine

> ### Supporting Domains
>
> - **Location Tracking:** GPS Updates, Geofencing
> - **Event Management:** Event Stream, Milestones
> - **Notifications:** Multi-channel delivery, Preferences
> - **Analytics:** Metrics, Dashboards, Reports

3.2 Bounded Contexts

3.2.1 Shipment Context (Core Domain)

Purpose: Manages the complete lifecycle of shipments

Key Aggregates:

- **Shipment** (Aggregate Root)
  - Unique shipment number
  - Customer and carrier information

    - Origin and destination
    - Status management
    - Stop collection

Business Invariants:

1. Shipment numbers must be globally unique
2. Status transitions follow defined state machine
3. Delivered shipments cannot be modified
4. Stops must maintain sequential ordering

Domain Events:

- ShipmentCreatedEvent
- ShipmentStatusChangedEvent
- StopAddedEvent
- ShipmentDeliveredEvent

3.2.2 Location Context (Supporting Domain)

Purpose: Real-time GPS tracking and geospatial operations

Key Aggregates:

- **TrackingSession**
    - Device tracking
    - Breadcrumb collection
    - Movement detection
- **Geofence**
    - Boundary definition
    - Trigger configuration
    - Entry/exit detection

Domain Services:

- GeofenceCalculator

- RouteDeviationDetector
- ETACalculator

3.3 Ubiquitous Language

Term	Definition
Shipment	A consignment of goods being transported from origin to destination
Carrier	The transportation company responsible for moving the shipment
Stop	A planned location where the shipment will arrive/depart during transit
Milestone	A significant event in the shipment lifecycle
Geofence	A virtual geographic boundary that triggers events when crossed
ETA	Estimated Time of Arrival at destination
Dwell Time	Time spent at a stop beyond planned duration
Exception	An unexpected event that impacts normal shipment flow
Mode	Transportation type (FTL, LTL, Rail, Ocean, Air, etc.)
Breadcrumb	A GPS location point in the shipment's journey

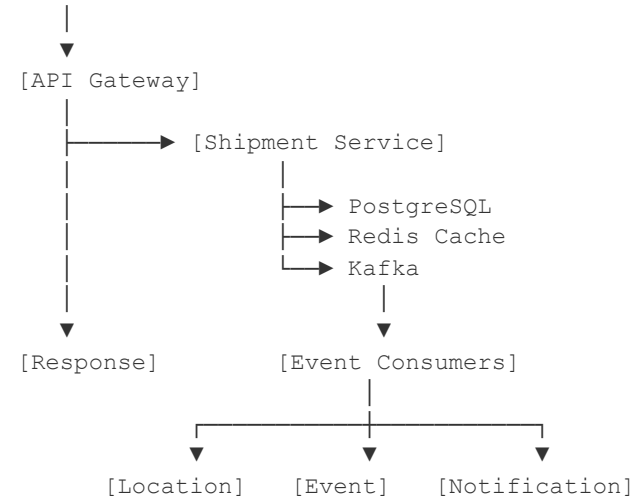
4. Microservices Architecture

4.1 Service Decomposition

Service	Port	Responsibilities
Shipment Service	8081	• Shipment CRUD operations • Status management • Stop management
Location Service	8082	• GPS tracking • Geofence management • Route calculations
Event Service	8083	• Event aggregation • Milestone tracking • Exception management
Notification Service	8084	• Multi-channel delivery • Template management • Preference handling
Analytics Service	8085	• KPI calculations • Report generation • Predictive analytics
API Gateway	8080	• Request routing • Authentication • Rate limiting

4.2 Service Communication Flow

```
User Request Flow:
═══════════════════

    [Client]
```


[API Gateway]
[Shipment Service]
PostgreSQL
Redis Cache
Kafka
[Response]
[Event Consumers]
[Location]
[Event]
[Notification]

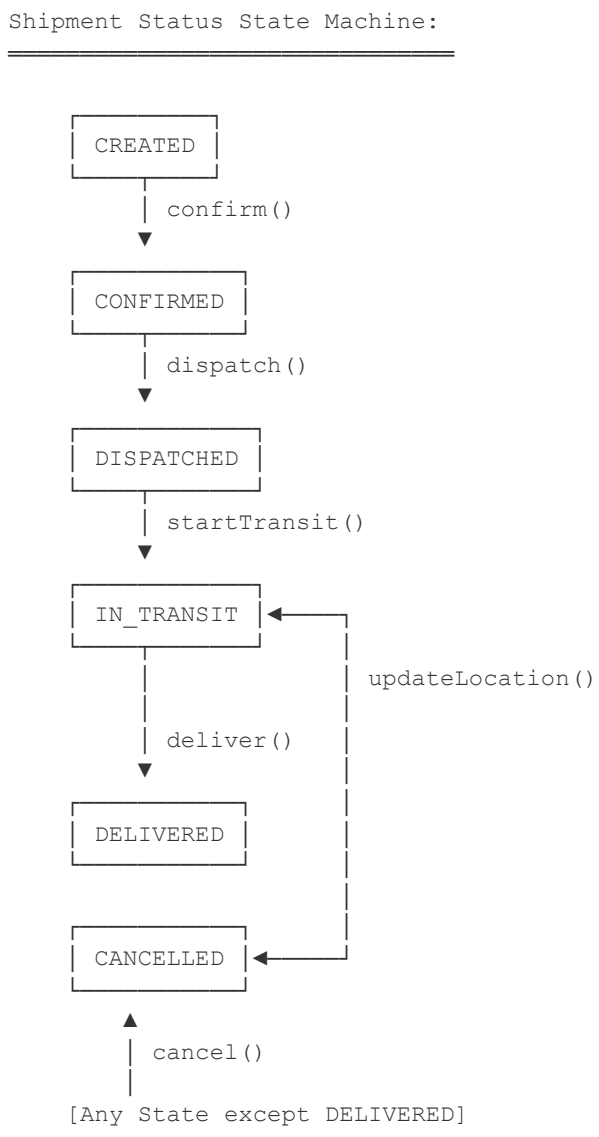
5. Technical Implementation

5.1 Aggregate Implementation Pattern

```
 // Shipment Aggregate Root
@Entity
@Table(name = "shipments")
public class Shipment extends BaseAggregateRoot {

    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;

    @Column(unique = true, nullable = false)
    private String shipmentNumber;

    @Embedded
    private ShipmentDetails details;

    @Enumerated(EnumType.STRING)
    private ShipmentStatus status;

    @OneToMany(cascade = CascadeType.ALL)
    private List<Stop> stops = new ArrayList<>();

    // Factory method for creation
    public static Shipment create(CreateShipmentCommand command) {
        validateShipmentCreation(command);

        Shipment shipment = new Shipment();
        shipment.shipmentNumber = generateShipmentNumber();
        shipment.details = ShipmentDetails.from(command);
        shipment.status = ShipmentStatus.CREATED;

        shipment.addDomainEvent(new ShipmentCreatedEvent(
            shipment.id,
            shipment.shipmentNumber
        ));

        return shipment;
    }

    // State transition with validation
    public void transitionTo(ShipmentStatus newStatus) {
```

```
        if (!isValidTransition(this.status, newStatus)) {
            throw new InvalidStateTransitionException();
        }

        ShipmentStatus oldStatus = this.status;
        this.status = newStatus;

        addDomainEvent(new ShipmentStatusChangedEvent(
            this.id, oldStatus, newStatus
        ));
    }
}
```

5.2 State Machine Implementation



5.3 Value Object Implementation

```
@Embeddable
public final class Address {

    @Column(name = "address_line_1", length = 255)
    private String addressLine1;

    @Column(name = "city", length = 100)
    private String city;

    @Column(name = "state", length = 50)
    private String state;

    @Column(name = "zip_code", length = 20)
    private String zipCode;

    @Column(name = "country", length = 50)
    private String country;

    @Column(name = "latitude", precision = 10, scale = 7)
    private BigDecimal latitude;

    @Column(name = "longitude", precision = 10, scale = 7)
    private BigDecimal longitude;

    // Factory method with validation
    public static Address of(String addressLine1, String city,
                             String state, String zipCode, String country,
                             BigDecimal latitude, BigDecimal longitude) {
        // Validate required fields
        Objects.requireNonNull(addressLine1, "Address line 1 is required");
        Objects.requireNonNull(city, "City is required");

        // Validate coordinates
        if (latitude != null && (latitude.compareTo(BigDecimal.valueOf(-90)) <
            || latitude.compareTo(BigDecimal.valueOf(90)) > 0)) {
            throw new IllegalArgumentException("Invalid latitude");
        }

        Address address = new Address();
        address.addressLine1 = addressLine1;
        address.city = city;
        address.state = state;
        address.zipCode = zipCode;
        address.country = country;
        address.latitude = latitude;
        address.longitude = longitude;
```

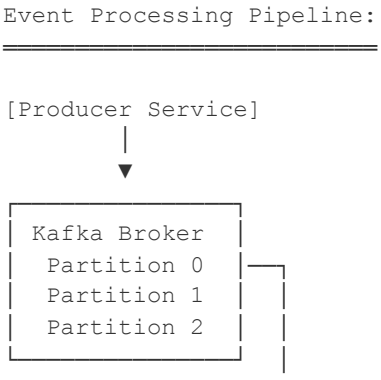
```
        return address;
    }
}
```

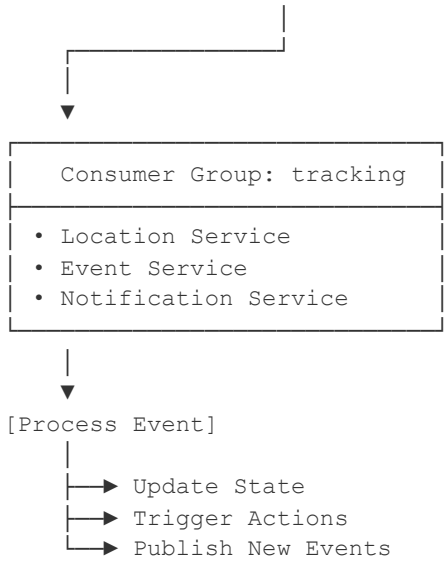
6. Event-Driven Architecture

6.1 Kafka Topic Architecture

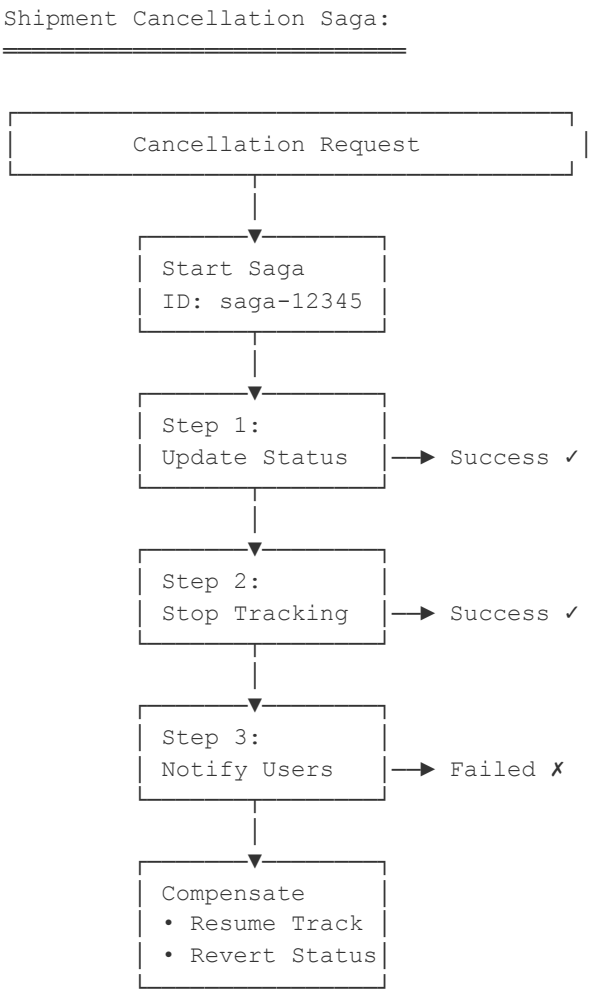
Category	Topics
Domain Events	• shipment-created • shipment-updated • shipment-cancelled • shipment-delivered • shipment-status-changed
Location Events	• location-updates • geofence-entered • geofence-exited • route-deviation
Tracking Events	• event-occurred • milestone-reached • exception-raised
System Events	• notification-sent • metrics-calculated • alert-triggered

6.2 Event Processing Flow



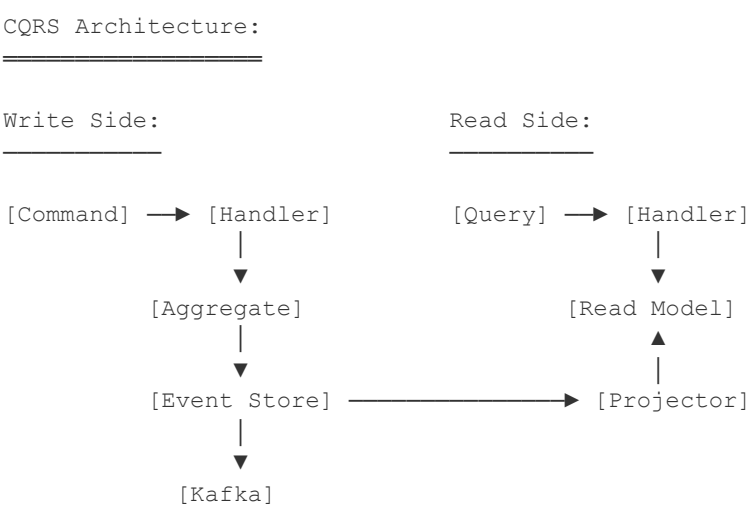


6.3 Saga Pattern Implementation



7. Data Flow and Integration

7.1 Command and Query Separation (CQRS)



7.2 Integration Architecture

Anti-Corruption Layer (ACL)

The ACL protects our domain from external system changes by:

- Protocol Translation
- Data Mapping
- Error Handling
- Retry Logic

7.3 Data Persistence Strategy

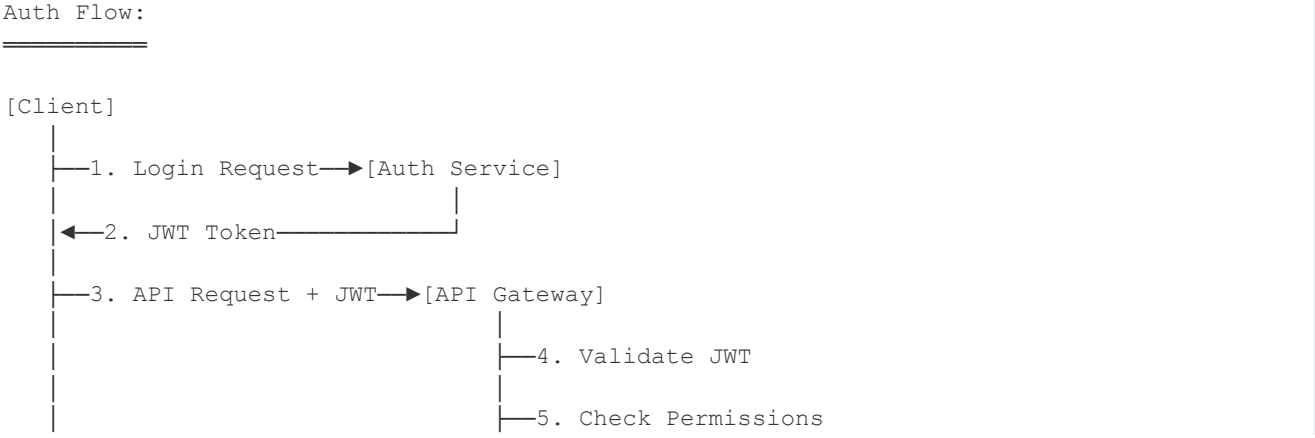
Service	Primary Storage	Secondary Storage
Shipment Service	PostgreSQL (Relational)	Redis (Cache)
Location Service	PostgreSQL + PostGIS	TimescaleDB (Time-series)
Event Service	Kafka (Event Store)	PostgreSQL (Snapshots)
Analytics Service	ClickHouse (OLAP)	Redis (Real-time)

8. Security and Compliance

8.1 Security Architecture

Layer	Security Measures
Edge Security	• WAF (Web Application Firewall) • DDoS Protection • SSL/TLS Termination
API Gateway Security	• JWT Authentication • OAuth 2.0 / OIDC • Rate Limiting • API Key Management
Service-to-Service Security	• mTLS (Mutual TLS) • Service Mesh (Istio) • Network Policies
Data Security	• Encryption at Rest • Encryption in Transit • Field-level Encryption • PII Masking

8.2 Authentication and Authorization Flow



```
│                                        │
│                                        └──6. Forward──►[Service]
│                                                            │
│◄─────────────7. Response───────────────────────────────────┘
```

9. Deployment and Operations

9.1 Kubernetes Deployment Architecture

Component	Configuration
Namespace	tracking-prod
Deployments	• shipment-service (3 pods) • location-service (3 pods) • event-service (2 pods) • notification-service (2 pods) • analytics-service (2 pods) • api-gateway (3 pods)
Services	• ClusterIP Services • LoadBalancer for Gateway
ConfigMaps & Secrets	• Application configs • Database credentials • API keys

9.2 CI/CD Pipeline

1. **Source Control:** GitHub/GitLab
2. **Build:** Maven/Gradle build
3. **Test:** Unit tests, Integration tests
4. **Code Quality:** SonarQube analysis
5. **Container:** Docker build and push
6. **Deploy Staging:** Kubernetes deployment
7. **Test Staging:** E2E tests, Performance tests
8. **Approval:** Manual approval gate
9. **Deploy Production:** Blue-Green deployment

10. **Monitor:** Health checks and monitoring

9.3 Monitoring and Observability

Observability Stack

- **Metrics:** Prometheus → Grafana
- **Logging:** Fluentd → Elasticsearch → Kibana
- **Tracing:** Jaeger Agent → Jaeger Collector → Jaeger UI

10. Appendices

Appendix A: API Endpoints

Service	Endpoint	Method	Description
Shipment	/api/v1/shipments	POST	Create shipment
Shipment	/api/v1/shipments/{id}	GET	Get shipment details
Shipment	/api/v1/shipments/{id}/status	PATCH	Update status
Location	/api/v1/locations	POST	Update location
Location	/api/v1/geofences	POST	Create geofence
Event	/api/v1/events	GET	Get events
Notification	/api/v1/notifications/preferences	PUT	Update preferences

Appendix B: Kafka Topics

Topic	Producers	Consumers	Retention
shipment-created	Shipment Service	Location, Event, Notification	7 days
location-updates	Location Service	Shipment, Event, Analytics	24 hours
event-occurred	Event Service	Notification, Analytics	30 days
milestone-reached	Event Service	Notification, Analytics	30 days

Appendix C: Performance Benchmarks

Metric	Target	Current
API Response Time (p95)	< 200ms	150ms
Throughput	10,000 req/s	12,000 req/s
Event Processing Latency	< 100ms	80ms
System Availability	99.9%	99.95%

Appendix D: Disaster Recovery

DR Strategy

- **Primary Region:** US-East (Active)
- **Secondary Region:** US-West (Standby)
- **Replication:** Continuous (Database, Kafka, Object Storage)
- **RTO:** 15 minutes
- **RPO:** 5 minutes

Document Control

Version	Date	Author	Changes
1.0	December 2024	Architecture Team	Initial version