Product Service Event Schema Documentation

Event-Driven Architecture Overview

The Product Service implements an event-driven architecture using Apache Kafka. This approach provides several benefits:

1. **Loose Coupling**: Services can evolve independently
2. **Scalability**: Event consumers can scale independently based on workload
3. **Real-time Processing**: Events enable real-time updates across the system
4. **Resilience**: Circuit breaker patterns prevent cascading failures

Kafka Topics

The Product Service publishes events to the following topics:

Topic Name	Purpose	Retention	Partitions	Replication Factor
`product-created`	New product creation events	7 days	3	3
`product-updated`	Product update events	7 days	3	3
`product-deleted`	Product deletion events	7 days	3	3
`inventory-updated`	Inventory level changes	7 days	3	3
`product-events`	Consolidated product events stream	7 days	6	3
`product-analytics`	Denormalized events for analytics	30 days	3	3

Event Schema

Base Event Schema

All events share a common base structure:

```
{
  "eventId": "uuid-string",
  "eventType": "PRODUCT_CREATED | PRODUCT_UPDATED | PRODUCT_DELETED |
INVENTORY_UPDATED",
  "entityId": "long",
  "timestamp": "ISO-8601-timestamp",
  "version": "semantic-version-string",
  "payload": { /* Event-specific data */ }
}
```

Event Payload Schemas

Product Created Event

```
{
  "eventId": "123e4567-e89b-12d3-a456-426614174000",
  "eventType": "PRODUCT_CREATED",
  "entityId": 1,
  "timestamp": "2025-05-16T10:30:45.123Z",
  "version": "1.0",
  "payload": {
    "name": "Product One",
    "description": "Product description",
    "sku": "PRD-001",
    "price": 19.99,
    "category": "electronics",
    "inventoryLevel": 100,
    "createdAt": "2025-05-16T10:30:45Z"
  }
}
```

Product Updated Event

```
{
  "eventId": "123e4567-e89b-12d3-a456-426614174001",
  "eventType": "PRODUCT_UPDATED",
  "entityId": 1,
  "timestamp": "2025-05-16T11:45:22.456Z",
  "version": "1.0",
  "payload": {
    "name": "Updated Product One",
    "description": "Updated product description",
    "sku": "PRD-001",
    "price": 24.99,
    "category": "electronics",
    "inventoryLevel": 100,
    "updatedAt": "2025-05-16T11:45:22Z",
    "changes": [
      {
        "field": "name",
        "oldValue": "Product One",
        "newValue": "Updated Product One"
      },
      {
        "field": "description",
        "oldValue": "Product description",
        "newValue": "Updated product description"
      },
      {
        "field": "price",
        "oldValue": 19.99,
        "newValue": 24.99
      }
    ]
  }
}
```

Product Deleted Event

```
{
  "eventId": "123e4567-e89b-12d3-a456-426614174002",
  "eventType": "PRODUCT_DELETED",
  "entityId": 1,
  "timestamp": "2025-05-16T14:20:15.789Z",
  "version": "1.0",
  "payload": {
    "sku": "PRD-001",
    "deletedAt": "2025-05-16T14:20:15Z",
    "reason": "DISCONTINUED"
  }
}
```

Inventory Updated Event

```
{
  "eventId": "123e4567-e89b-12d3-a456-426614174003",
  "eventType": "INVENTORY_UPDATED",
  "entityId": 1,
  "timestamp": "2025-05-16T15:10:33.012Z",
  "version": "1.0",
  "payload": {
    "sku": "PRD-001",
    "previousLevel": 100,
    "newLevel": 90,
    "changeAmount": -10,
    "changeReason": "ORDER_PLACED",
    "referenceId": "order-12345",
    "updatedAt": "2025-05-16T15:10:33Z"
  }
}
```

Enhanced Event Schema

For advanced use cases, the service also provides enhanced events with additional metadata:

```
{
  "eventId": "123e4567-e89b-12d3-a456-426614174004",
  "eventType": "PRODUCT_CREATED",
  "entityId": 1,
  "timestamp": "2025-05-16T10:30:45.123Z",
  "version": "1.0",
  "source": "product-service",
  "traceId": "trace-uuid-string",
  "correlationId": "correlation-uuid-string",
  "userId": "user-uuid-or-id",
  "tenantId": "tenant-id-for-multi-tenancy",
  "metadata": {
    "environment": "production",
    "region": "us-west",
    "datacenter": "dc1"
  },
  "payload": { /* Event-specific data */ }
}
```

Event Publishing Implementation

Java Event Classes

```java
// Base event class
public class ProductEvent {
    private String eventId;
    private ProductEventType eventType;
    private Long productId;
    private ZonedDateTime timestamp;
    private String version;
    private Map<String, Object> payload;

    // Constructors, getters, setters...
}

// Event types enum
public enum ProductEventType {
    PRODUCT_CREATED,
    PRODUCT_UPDATED,
    PRODUCT_DELETED,
    INVENTORY_UPDATED
}
```

Event Publisher with Circuit Breaker Pattern

```java
@Service
public class ProductEventPublisher {

    private final KafkaTemplate<String, ProductEvent> kafkaTemplate;
    private final String productCreatedTopic;
    private final String productUpdatedTopic;
    private final String productDeletedTopic;

    // Constructor with dependencies...

    @CircuitBreaker(name = "kafkaPublisher", fallbackMethod = "fallbackPublish")
    @Retry(name = "kafkaPublisher")
    public CompletableFuture<SendResult<String, ProductEvent>>
publishProductCreatedEvent(Product product) {
        ProductEvent event = new ProductEvent(
            UUID.randomUUID().toString(),
            ProductEventType.PRODUCT_CREATED,
            product.getId(),
            ZonedDateTime.now(),
            "1.0",
            createProductPayload(product)
        );

        return kafkaTemplate.send(productCreatedTopic, product.getId().toString(),
event)
            .completable();
    }

    // Similar methods for other event types...

    public CompletableFuture<SendResult<String, ProductEvent>>
fallbackPublish(Product product, Throwable ex) {
        // Log the failure
        log.error("Failed to publish event for product {}, falling back to
database", product.getId(), ex);

        // Store in outbox table for later processing
        eventOutboxRepository.save(new EventOutbox(product, ex.getMessage()));

        // Return a completed future to avoid blocking the caller
        return CompletableFuture.completedFuture(null);
    }
}
```

Event Consumers

Other microservices in the ecosystem consume these events:

1. **Inventory Service**: Consumes `product-created`, `product-updated`, and `product-deleted` events to maintain a product catalog

2. **Order Service**: Consumes `inventory-updated` events to validate order placement
3. **Analytics Service**: Consumes `product-events` for business intelligence
4. **Search Service**: Consumes all product events to update the search index

Event Versioning Strategy

The Product Service implements a robust event versioning strategy:

1. **Version Field**: All events include a semantic version
2. **Backward Compatibility**: New fields are added in a non-breaking manner
3. **Schema Registry**: Confluent Schema Registry ensures compatibility
4. **Consumer Resilience**: Consumers handle unknown fields gracefully

Outbox Pattern Implementation

For reliable event publishing, the service implements the Outbox pattern:

1. Domain changes and event outbox entries are saved in a single transaction
2. A scheduled job polls the outbox table and publishes events to Kafka
3. Successfully published events are marked as processed
4. Failed events are retried with exponential backoff

```
@Entity
@Table(name = "event_outbox")
public class EventOutbox {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String aggregateType;
    private Long aggregateId;
    private String eventType;
    private String payload;
    private String errorMessage;
    private int retryCount;
    private boolean processed;
    private ZonedDateTime createdAt;
    private ZonedDateTime processedAt;

    // Constructors, getters, setters...
}
```

Best Practices for Event Consumers

1. **Idempotent Processing**: Handle duplicate events gracefully
2. **Parallel Processing**: Leverage Kafka partitioning for parallel consumption
3. **Error Handling**: Implement dead-letter queues for failed processing
4. **Offset Management**: Commit offsets only after successful processing
5. **Consumer Groups**: Use consumer groups for load balancing
6. **Monitoring**: Track consumer lag and processing times