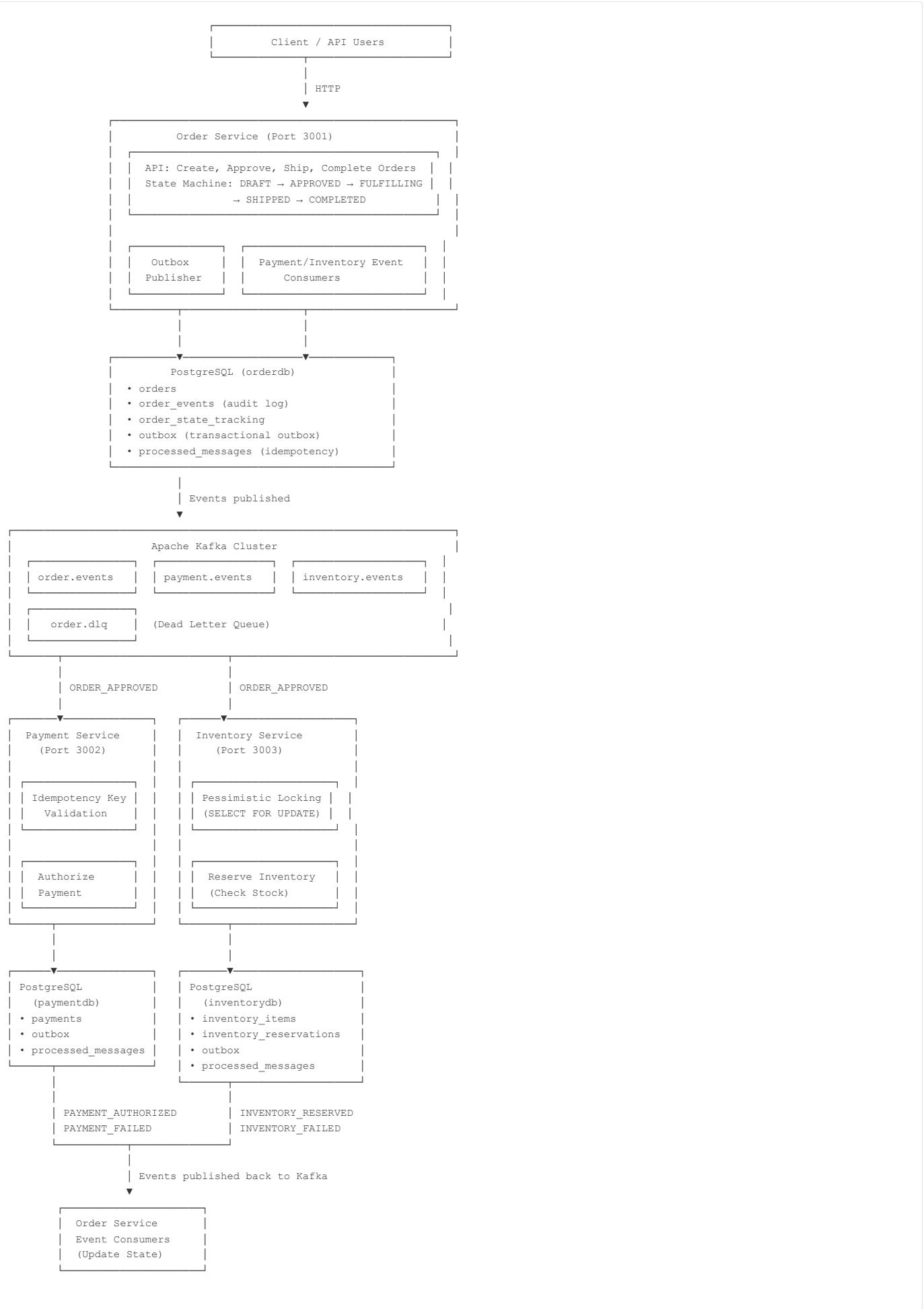


OrderFlow Architecture Diagrams

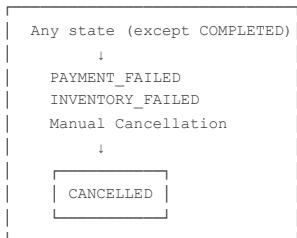
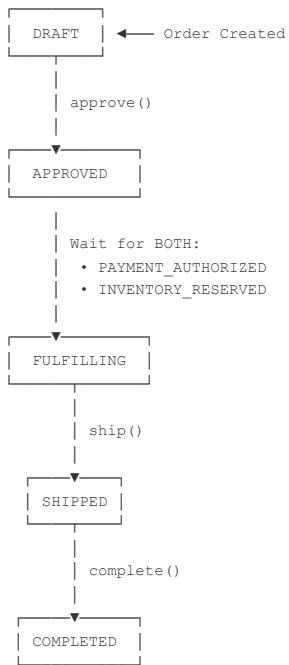
Table of Contents

- [System Architecture](#)
- [Order State Machine](#)
- [Event Flow - Happy Path](#)
- [Production Patterns](#)
- [Key Metrics & Monitoring](#)
- [Failure Scenarios](#)
- [Technology Stack](#)

System Architecture



Order State Machine



Event Flow - Happy Path

```
1. Client Request
  └─ POST /orders (create)
    └─ Order Service
      └─ Database: Insert order (state: DRAFT)
      └─ Response: Order ID

2. Order Approval
  └─ POST /orders/:id/approve
    └─ Order Service
      └─ Database Transaction:
        └─ Update order (state: APPROVED)
        └─ Insert into outbox table (ORDER_APPROVED event)
    └─ Outbox Publisher (polls every 1s)
      └─ Kafka: Publish ORDER_APPROVED to order.events

3. Parallel Processing
  └─ Payment Service
    └─ Kafka Consumer
      (reads ORDER_APPROVED)
    └─ Check Idempotency
      (processed_messages table)
    └─ Authorize Payment
      (90% success rate - demo)
    └─ Database Transaction:
      └─ Insert payment record
      └─ Insert outbox event
      └─ Mark message processed
    └─ Kafka: PAYMENT_AUTHORIZED

  └─ Inventory Service
    └─ Kafka Consumer
      (reads ORDER_APPROVED)
    └─ Check Idempotency
      (processed_messages table)
    └─ Reserve Inventory
      (SELECT FOR UPDATE)
    └─ Database Transaction:
      └─ Reserve quantity
      └─ Insert reservation
      └─ Insert outbox event
      └─ Mark message processed
    └─ Kafka: INVENTORY_RESERVED

4. State Transition
  └─ Order Service Event Consumers
    └─ Received: PAYMENT_AUTHORIZED
      └─ Update order_state_tracking (payment_authorized = true)

    └─ Received: INVENTORY_RESERVED
      └─ Update order_state_tracking (inventory_reserved = true)

    └─ Check: Both payment AND inventory ready?
      └─ YES → Transition to FULFILLING
        └─ Database Transaction:
          └─ Update order (state: FULFILLING)
          └─ Insert outbox (ORDER_STATE_CHANGED event)
        └─ Kafka: Publish ORDER_STATE_CHANGED

5. Shipping & Completion
  └─ POST /orders/:id/ship
    └─ State: FULFILLING → SHIPPED

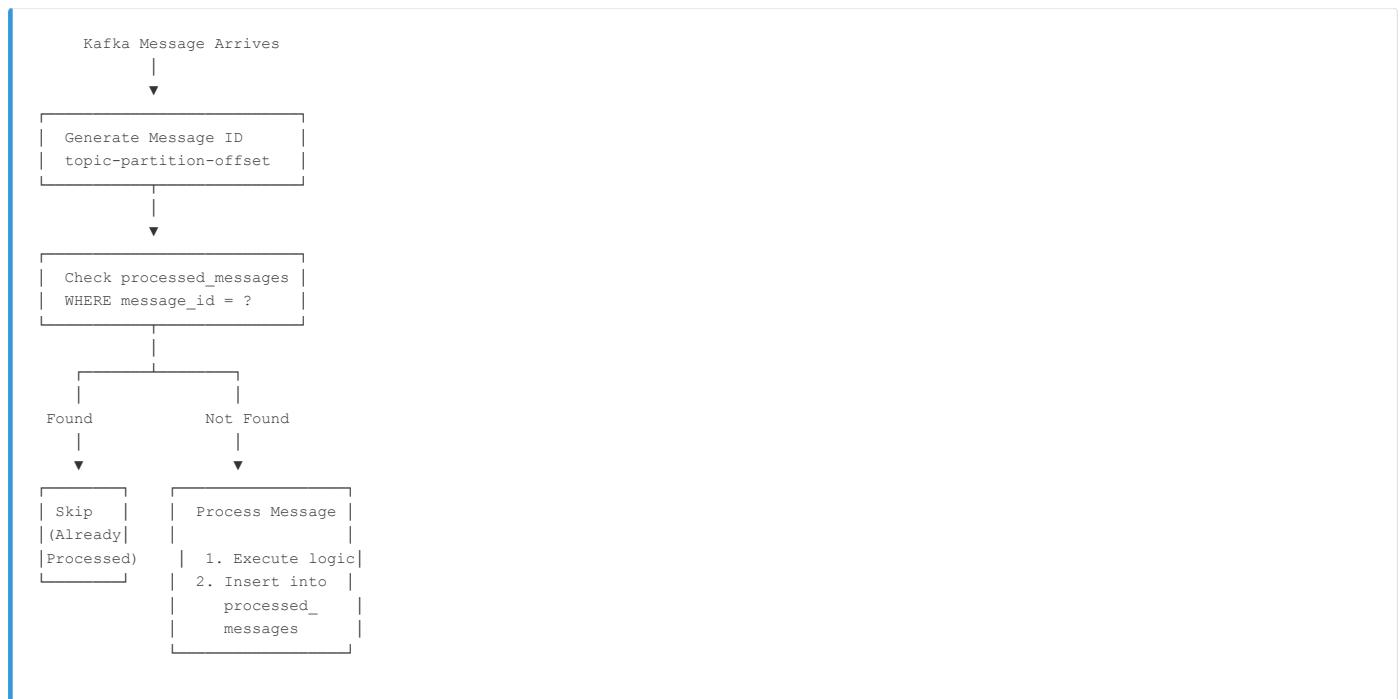
  └─ POST /orders/:id/complete
    └─ State: SHIPPED → COMPLETED
```

Production Patterns

1. Transactional Outbox Pattern



2. Idempotent Consumer



3. Out-of-Order Event Handling

Order State Tracking Table:

order_id	payment_authorized	inventory_reserved
123-abc	false	false

```

Scenario A: Payment arrives first
|→ PAYMENT_AUTHORIZED arrives
|  ↳ Update: payment_authorized = true
|  ↳ Check: inventory_reserved? NO → Wait
|
└→ INVENTORY_RESERVED arrives
  ↳ Update: inventory_reserved = true
  ↳ Check: BOTH true? YES → Transition to FULFILLING

Scenario B: Inventory arrives first
|→ INVENTORY_RESERVED arrives
|  ↳ Update: inventory_reserved = true
|  ↳ Check: payment_authorized? NO → Wait
|
└→ PAYMENT_AUTHORIZED arrives
  ↳ Update: payment_authorized = true
  ↳ Check: BOTH true? YES → Transition to FULFILLING

Result: Order progresses correctly regardless of event order!

```

Key Metrics & Monitoring

Category	Metrics
Service Metrics	<ul style="list-style-type: none">• Request Rate (orders/sec)• Request Latency (p50, p95, p99)• Error Rate (% failed requests)• Order State Distribution
Kafka Metrics	<ul style="list-style-type: none">• Consumer Lag (messages behind)• Message Throughput (msgs/sec)• Failed Messages → DLQ count
Database Metrics	<ul style="list-style-type: none">• Outbox Queue Depth (pending events)• Query Latency• Connection Pool Usage• Lock Wait Time (inventory)

Failure Scenarios

Failure	Impact	Recovery
Kafka Down	Events buffered in outbox	Outbox holds events, auto publishes when Kafka back
Payment Service Down	Orders stuck in APPROVED state	Consumer catches up from offset
Inventory Out of Stock	Order cancelled	INVENTORY_FAILED event triggers cancellation
Payment Gateway Fails	Order cancelled	PAYMENT_FAILED event triggers cancellation
Database Connection Lost	Service unavailable	Restart service, reconnect

Technology Stack

Layer	Technologies
Application Layer	<ul style="list-style-type: none">TypeScript + Node.js 20+Fastify (HTTP framework)Zod (Schema validation)
Messaging Layer	<ul style="list-style-type: none">Apache Kafka (Event streaming)kafkajs (Client library)Idempotent producers
Data Layer	<ul style="list-style-type: none">PostgreSQL 15 (ACID transactions)pg (Client library)Per-service databases
Observability Layer	<ul style="list-style-type: none">Pino (Structured logging)prom-client (Metrics)Correlation IDs
Infrastructure	<ul style="list-style-type: none">Docker + Docker ComposeGitHub Actions (CI/CD)Jest (Testing)

OrderFlow - Production-Grade Event-Driven Microservices System

GitHub: github.com/puneethkotha/Order-Flow

Generated: February 2026