# Product Requirements Document: Kafka-Based Food Processing Pipeline

**Version:** 1.0
**Date:** October 10, 2025
**Author:** Adam White
**Status:** Draft

---

## Executive Summary

### Purpose

This is a **learning and portfolio project** designed to:

1. **Master Kafka** through hands-on implementation of event-driven architecture
2. **Demonstrate systems design skills** that were gaps in the RxFood VP Engineering interview
3. **Create a working prototype** that shows architectural thinking and execution capability
4. **Potentially demo to RxFood** to address interview weaknesses and show learning agility

### What This Is NOT

- NOT a production-ready medical application
- NOT attempting to compete with RxFood's actual product
- NOT handling real PHI/PII data
- NOT pursuing commercial deployment

### Success Criteria

- ✅ Working Kafka-based event pipeline processing food images end-to-end
- ✅ Demonstrate understanding of async processing, message brokers, microservices
- ✅ Deployable demo that can be shown in interviews
- ✅ Learn enough Kafka to confidently discuss architecture decisions
- ✅ Learn enough Kubernetes to deploy and scale containerized applications
- ✅ Complete in 4-5 weekends (not a long-term project)

---

## Development Approach

**Two-Phase Strategy:**

**Phase 1-4: Docker Compose (Local Development)**

- Fast iteration cycles
- Simple setup and debugging
- Focus on application logic and Kafka patterns
- No k8s complexity while learning Kafka

**Phase 5: Kubernetes (Production-Like Deployment)**

- Deploy fully working system to k8s
- Learn container orchestration
- Demonstrate production-ready patterns
- Impressive demo for interviews

**Why This Approach:**

- Separation of concerns: Learn Kafka first, then k8s
- Faster development without k8s overhead
- More realistic (most teams develop locally, deploy to k8s)
- Demonstrates understanding of both technologies

---

# Product Vision

**Vision Statement:**
Build a simplified food image processing pipeline using Kafka that demonstrates event-driven architecture, microservices patterns, and async processing at scale.

**Inspiration:**
Based on RxFood's architecture evolution from monolith → Kafka-based agent orchestration, this project mimics their real-world technical challenges in a learning context.

---

# User Stories

## Primary User: You (The Builder)

**As a engineering leader learning Kafka:**

- I want to set up a working Kafka cluster so I understand topics, partitions, and consumer groups
- I want to build producer/consumer services so I learn the programming model
- I want to see events flow through the system so I understand observability
- I want to handle failures gracefully so I learn error handling patterns

**As a job candidate:**

- I want a working demo I can show in interviews
- I want to explain architecture decisions I actually made
- I want to demonstrate learning agility (identified gap → built solution)

## Secondary User: Hiring Managers/Interviewers

**As someone evaluating candidates:**

- I want to see working code, not just talk
- I want to understand their architectural thinking
- I want to know they can learn new technologies quickly

---

# Technical Architecture

## High-Level Flow


## Components

### 1. Infrastructure Layer

**Local Development:**

- **Docker Compose -** Fast iteration, simple setup
- **Kafka Cluster** (single broker for local)
- **Zookeeper** (Kafka dependency)
- **Kafka UI** (for visualization and debugging)
- **Postgres** (result storage)

**Production/Demo Deployment:**

- **Kubernetes Cluster** (k3s on EC2 or Minikube locally)
- **Kafka via Strimzi Operator** (k8s-native Kafka)
- **Postgres StatefulSet** (persistent storage)
- **Ingress Controller** (expose services)
- **ConfigMaps & Secrets** (configuration management)

**2. Application Services**

**API Gateway Service**

- Technology: FastAPI (Python)
- Responsibilities:
  - Accept photo uploads (multipart/form-data)
  - Generate unique request ID
  - Publish "food.submitted" event to Kafka
  - Return request ID to user for polling
  - Provide status endpoint for polling
- Port: 8000

**Image Analysis Service**

- Technology: Python + CLIP (OpenAI's open-source vision model)
- Responsibilities:
  - Consume "food.submitted" events
  - Run image through CLIP model
  - Extract food items detected
  - Publish "food.analyzed" event with detected items
- Kafka Consumer Group: "image-analysis-group"

**Food Matching Service**

- Technology: Python + USDA Food Database API
- Responsibilities:
  - Consume "food.analyzed" events
  - Look up detected foods in USDA database
  - Handle negations (without, sans, hold the mayo)
  - Find ingredient breakdowns
  - Provide approximations when exact match not found
  - Publish "food.matched" event with nutrition data
- Kafka Consumer Group: "food-matching-group"

**Result Service**

- Technology: Python + SQLAlchemy + Postgres
- Responsibilities:
  - Consume "food.matched" events
  - Store final results in Postgres
  - Update status for request ID
  - Provide data for API Gateway polling endpoint
- Kafka Consumer Group: "result-group"

**3. Frontend (Minimal)**

- Simple HTML/JavaScript upload form
- Shows request ID and polls for results
- Displays nutrition breakdown when complete

## Kafka Topics

| Topic Name | Partitions | Replication | Purpose |
|---|---|---|---|
| food.submitted | 3 | 2 | Raw upload events |
| food.analyzed | 3 | 2 | Image analysis results |
| food.matched | 3 | 2 | USDA matching results |
| food.dlq | 1 | 2 | Dead letter queue for failures |

## Data Models

### Event: food.submitted

json

```
{
 "request_id": "uuid-v4",
 "image_data": "base64-encoded",
 "timestamp": "ISO-8601",
 "user_input": "optional text description"
}
```

### Event: food.analyzed

json

```
{
 "request_id": "uuid-v4",
 "detected_items": ["apple", "banana", "chicken breast"],
 "confidence_scores": [0.95, 0.89, 0.87],
 "timestamp": "ISO-8601"
}
```

### Event: food.matched

json

```
{
 "request_id": "uuid-v4",
 "nutrition_data": [
  {
   "food": "apple",
   "usda_match": "Apples, raw, with skin",
   "calories": 95,
   "protein_g": 0.5,
   "carbs_g": 25,
   "fat_g": 0.3
  }
 ],
 "timestamp": "ISO-8601"
}
```

### Observability

**Metrics to Track:**

- Events published per topic (rate)
- Consumer lag per consumer group
- Processing time per service
- Error rate per service
- End-to-end latency (upload → result)

**Tools:**

- Kafka UI for topic/consumer monitoring
- Python logging to stdout (collected by Docker)
- Simple metrics endpoint on each service

---

# Implementation Phases

## Phase 1: Infrastructure Setup (Weekend 1)

**Goal:** Working Kafka cluster with visibility

**Tasks:**

- Set up docker-compose.yml with Kafka, Zookeeper, Kafka UI, Postgres
- Verify Kafka UI accessible at localhost:8080
- Create topics via Kafka UI or CLI
- Write simple producer/consumer test scripts
- Confirm messages flow through topics

**Success Criteria:**

- Can create topics
- Can publish test messages
- Can consume test messages
- Can see messages in Kafka UI

## Phase 2: Core Pipeline (Weekend 2)

**Goal:** End-to-end event flow working

**Tasks:**

- Build API Gateway service
  - Photo upload endpoint
  - Kafka producer for food.submitted
  - Polling endpoint for results
- Build Image Analysis service
  - CLIP model integration
  - Kafka consumer for food.submitted
  - Kafka producer for food.analyzed
- Build stub services for Food Matching and Result
  - Just pass through events for now
- Test complete event flow in Kafka UI

**Success Criteria:**

- Upload photo → see food.submitted event
- See food.analyzed event appear
- See food.matched event appear
- Can poll for status

## Phase 3: Real Processing (Weekend 3)

**Goal:** Actual ML and database integration

**Tasks:**

- Implement CLIP model in Image Analysis
  - Load pre-trained model
  - Process images
  - Return detected food items
- Implement USDA API in Food Matching
  - API client for USDA database
  - Fuzzy matching logic
  - Negation parsing (without, sans, etc.)
- Implement Postgres storage in Result Service
  - SQLAlchemy models
  - Store nutrition data
  - Update request status

**Success Criteria:**

- Upload real food photo
- Get actual detected items from CLIP
- Get real nutrition data from USDA
- Results stored in Postgres
- Can retrieve results via API

## Phase 4: Production Patterns (Weekend 4)

**Goal:** Make it demo-worthy

**Tasks:**

- Add error handling
  - Dead letter queue for failed events
  - Retry logic in consumers
  - Graceful degradation
- Add monitoring
  - Consumer lag tracking
  - Processing time metrics
  - Error rate metrics
- Add partitioning
  - 3 partitions per topic
  - Demonstrate parallel processing
- Load testing
  - Upload 100 photos concurrently
  - Measure consumer lag
  - Verify no dropped events
- Simple web UI
  - Upload form
  - Status display
  - Nutrition results table

**Success Criteria:**

- Can handle 100 concurrent uploads
- Failed events go to DLQ
- Consumer lag visible in Kafka UI
- Web UI works end-to-end
- Can explain architecture decisions

## Phase 5: Kubernetes Deployment (Weekend 5)

**Goal:** Deploy to Kubernetes cluster

**Tasks:**

- Set up local k8s cluster
  - Install Minikube or k3s
  - Verify kubectl access
- Deploy Kafka using Strimzi Operator
  - Install Strimzi operator
  - Create Kafka cluster resource
  - Create topics via Kafka CR
- Create Kubernetes manifests
  - Deployments for each service
  - Services for networking
  - ConfigMaps for configuration
  - Secrets for credentials
  - Ingress for external access
- Deploy application services
  - Build and push Docker images

    - Apply k8s manifests
    - Verify pod health
- Test end-to-end in k8s
    - Upload via ingress
    - Check pod logs
    - Verify events in Kafka
    - Confirm results in database
- (Optional) Deploy to cloud
    - Provision EC2 instance
    - Install k3s
    - Deploy application
    - Configure DNS

**Success Criteria:**

- All services running in k8s pods
- Can access API via ingress
- Events flow through Kafka
- Can scale deployments (replicas)
- Can demonstrate rolling updates
- Can show pod logs and metrics

---

# Technology Stack

## Infrastructure

- **Docker & Docker Compose** - Local development
- **Kubernetes 1.28+** - Container orchestration
- **k3s** - Lightweight k8s for single-node deployment
- **Strimzi Kafka Operator** - K8s-native Kafka management
- **Kafka 3.6+** - Message broker
- **Postgres 15** - Result storage
- **Helm 3** - K8s package manager
- **kubectl** - K8s CLI
- **Kafka UI** - Observability

## Application

- **Python 3.11+** - All services
- **FastAPI** - API Gateway
- **kafka-python** - Kafka client
- **CLIP (OpenAI)** - Image analysis model
- **SQLAlchemy** - Database ORM
- **Requests** - HTTP client for USDA API

## Deployment

- **Kubernetes (k8s)** - Production-like deployment
- **Minikube** - Local k8s for testing
- **AWS EKS** or **EC2 with k3s** - Cloud deployment
- **Helm Charts** - Package management
- **Cost:** ~$100-150/month (EC2 + storage)

---

# Out of Scope

**Explicitly NOT building:**

- Authentication/authorization
- User accounts or multi-tenancy
- PHI/PII handling or HIPAA compliance
- Mobile apps
- Advanced ML model training
- Real-time CGM integration
- Multi-region deployment
- Advanced monitoring (Datadog, Prometheus - basic k8s metrics only)
- CI/CD pipeline (manual deployments)
- Automated testing (focus on learning, not production quality)
- Service mesh (Istio, Linkerd)
- Advanced k8s features:
  - HorizontalPodAutoscaler (manual scaling only)
  - NetworkPolicies
  - Pod Security Policies
  - Multi-cluster deployments
  - Helm charts (using raw manifests)
  - GitOps (ArgoCD, Flux)

---

# Success Metrics

## Learning Objectives

- [ ] Can explain Kafka topics, partitions, consumer groups
- [ ] Can explain at-least-once vs exactly-once delivery
- [ ] Can explain consumer lag and how to monitor it
- [ ] Can explain when to use Kafka vs simpler queues
- [ ] Can explain event-driven architecture tradeoffs
- [ ] Can explain Kubernetes pods, deployments, services
- [ ] Can explain StatefulSets vs Deployments
- [ ] Can explain ConfigMaps and Secrets
- [ ] Can explain Ingress and load balancing
- [ ] Can explain how to scale applications in k8s

## Demo Objectives

- [ ] Working system that processes real food images
- [ ] Can show event flow in Kafka UI during demo
- [ ] Can explain architecture decisions made
- [ ] Can discuss how to scale this (partitions, consumers)
- [ ] Can discuss error handling patterns implemented

## Interview Objectives

- [ ] Confidently answer "Why Kafka?" questions
- [ ] Explain async processing patterns

- [ ] Discuss microservices communication
- [ ] Show learning agility (gap identified → skill acquired)

---

# Timeline

**Total Time Investment:** 4-5 weekends (32-40 hours)

- **Weekend 1:** Infrastructure setup with Docker Compose (6-8 hours)
- **Weekend 2:** Core pipeline development (8-10 hours)
- **Weekend 3:** Real processing with ML and database (8-10 hours)
- **Weekend 4:** Production patterns and error handling (6-8 hours)
- **Weekend 5:** Kubernetes deployment (6-8 hours)

**Target Completion:** Mid-November 2025

---

# Questions to Answer Through Building

## Kafka Questions

1. **When is Kafka overkill vs appropriate?**
   - For this simple pipeline, RabbitMQ might be sufficient
   - Kafka's value comes from: replay, multiple consumers, high throughput
   - Learn when added complexity is justified
2. **How do you handle schema evolution?**
   - What happens when you change event structure?
   - Backward/forward compatibility considerations
   - Schema registry (out of scope but worth understanding)
3. **How do you handle failures?**
   - Consumer crashes mid-processing
   - Broker failures
   - Downstream service unavailable
   - Dead letter queue patterns
4. **How do you monitor distributed systems?**
   - Consumer lag as leading indicator
   - Distributed tracing (conceptual understanding)
   - Alerting thresholds

## Kubernetes Questions

5. **When should you use StatefulSets vs Deployments?**
   - Postgres needs stable storage → StatefulSet
   - Stateless services → Deployment
   - Understanding persistent volumes
6. **How do you handle configuration?**
   - ConfigMaps for non-sensitive config
   - Secrets for credentials
   - Environment variables vs mounted volumes
7. **How do you scale services in k8s?**
   - Horizontal pod autoscaling
   - Resource requests and limits
   - When to scale vs when to optimize

8. **How does networking work in k8s?**
    - ClusterIP vs NodePort vs LoadBalancer
    - Ingress for external access
    - Service discovery
9. **How do you deploy updates without downtime?**
    - Rolling updates strategy
    - Readiness and liveness probes
    - Blue-green vs canary deployments

---

# Demo Script (For Interviews)

**Setup (1 minute):** "After our interview, I realized I couldn't answer your Kafka and Kubernetes questions as well as I wanted. So I built this food processing pipeline to learn event-driven architecture and container orchestration hands-on."

**Demo (4 minutes):**

1. Show architecture diagram
2. Show Kubernetes dashboard
    - `kubectl get pods` - all services running
    - `kubectl get deployments` - replica counts
3. Upload food photo via ingress URL
4. Show event flow in Kafka UI
5. Show pod logs in real-time
    - `kubectl logs -f <image-analysis-pod>`
6. Show result in database
7. Demonstrate scaling
    - `kubectl scale deployment/image-analysis --replicas=3`
    - Upload 10 photos concurrently
    - Show parallel processing across pods

**Technical Discussion (5 minutes):**

1. Why Kafka? Multi-consumer pattern, replay, scale
2. Why Kubernetes? Orchestration, scaling, resilience
3. How would you scale this further? Add partitions, increase replicas, resource limits
4. What could break? Database bottleneck, consumer lag, pod crashes
5. How is this different from your monolith? Decoupling, async, observability, container orchestration
6. Show StatefulSet for Postgres vs Deployment for stateless services
7. Explain ConfigMaps vs Secrets for configuration

**Close:** "This is how I approach gaps - I build things until I understand them. I'm confident I can architect your infrastructure for 10x scale in January because I've now worked through these patterns myself in both Kafka and Kubernetes."

---

# Resources

**Kafka Learning:**

- Kafka: The Definitive Guide (O'Reilly)
- Confluent Developer tutorials
- Apache Kafka documentation

**Kubernetes Learning:**

- Kubernetes in Action (Manning)
- Kubernetes official documentation
- Strimzi Kafka Operator docs: https://strimzi.io/
- k3s documentation: https://k3s.io/

**Tools:**

- Kafka UI: https://github.com/provectus/kafka-ui
- CLIP model: https://github.com/openai/CLIP
- USDA Food Database API: https://fdc.nal.usda.gov/api-guide.html
- Minikube: https://minikube.sigs.k8s.io/
- k3s: https://k3s.io/

**Example Code:**

- FastAPI + Kafka: https://github.com/confluentinc/confluent-kafka-python
- Strimzi examples: https://github.com/strimzi/strimzi-kafka-operator/tree/main/examples
- k8s Python apps: https://kubernetes.io/docs/tutorials/

---

# Appendix: Docker Compose Starter


yaml

```yaml
version: '3.8'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - "2181:2181"

  kafka-1:
    image: confluentinc/cp-kafka:7.5.0
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1

  kafka-ui:
    image: provectuslabs/kafka-ui:latest
    depends_on:
      - kafka-1
    ports:
      - "8080:8080"
    environment:
      KAFKA_CLUSTERS_0_NAME: local
      KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: kafka-1:9092

  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: foodpipeline
      POSTGRES_USER: pipeline
      POSTGRES_PASSWORD: pipeline123
    ports:
      - "5432:5432"
```

```
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

---

# Appendix: Kubernetes Manifests Starter

## API Gateway Deployment

yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
spec:
  replicas: 2
  selector:
    matchLabels:
      app: api-gateway
  template:
    metadata:
      labels:
        app: api-gateway
    spec:
      containers:
      - name: api-gateway
        image: your-registry/api-gateway:latest
        ports:
        - containerPort: 8000
        env:
        - name: KAFKA_BOOTSTRAP_SERVERS
          value: "kafka-cluster-kafka-bootstrap:9092"
        - name: POSTGRES_HOST
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: postgres_host
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
---
apiVersion: v1
kind: Service
metadata:
  name: api-gateway
spec:
  selector:
```

```
  app: api-gateway
ports:
- port: 80
  targetPort: 8000
type: ClusterIP
```

### Strimzi Kafka Cluster

yaml

```yaml
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: kafka-cluster
spec:
  kafka:
    version: 3.6.0
    replicas: 1
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
    config:
      offsets.topic.replication.factor: 1
      transaction.state.log.replication.factor: 1
      transaction.state.log.min.isr: 1
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 10Gi
        deleteClaim: false
  zookeeper:
    replicas: 1
    storage:
      type: persistent-claim
      size: 5Gi
      deleteClaim: false
  entityOperator:
    topicOperator: {}
    userOperator: {}
```

### Kafka Topics

yaml

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: food-submitted
  labels:
    strimzi.io/cluster: kafka-cluster
spec:
  partitions: 3
  replicas: 1
  config:
    retention.ms: 7200000
    segment.bytes: 1073741824
---
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: food-analyzed
  labels:
    strimzi.io/cluster: kafka-cluster
spec:
  partitions: 3
  replicas: 1
---
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: food-matched
  labels:
    strimzi.io/cluster: kafka-cluster
spec:
  partitions: 3
  replicas: 1
---
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: food-dlq
  labels:
    strimzi.io/cluster: kafka-cluster
spec:
```

```
  partitions: 1
  replicas: 1
```

### Postgres StatefulSet

yaml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
spec:
  serviceName: postgres
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: postgres:15
        ports:
        - containerPort: 5432
        env:
        - name: POSTGRES_DB
          value: foodpipeline
        - name: POSTGRES_USER
          valueFrom:
            secretKeyRef:
              name: postgres-secret
              key: username
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: postgres-secret
              key: password
        volumeMounts:
        - name: postgres-storage
          mountPath: /var/lib/postgresql/data
  volumeClaimTemplates:
  - metadata:
      name: postgres-storage
    spec:
      accessModes: [ "ReadWriteOnce" ]
```

```
      resources:
        requests:
          storage: 10Gi
---
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  selector:
    app: postgres
  ports:
  - port: 5432
  clusterIP: None
```

### ConfigMap

yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  kafka_bootstrap_servers: "kafka-cluster-kafka-bootstrap:9092"
  postgres_host: "postgres"
  postgres_db: "foodpipeline"
```

### Secret

yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: postgres-secret
type: Opaque
stringData:
  username: pipeline
  password: pipeline123
```

### Ingress

yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: api-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: food-pipeline.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: api-gateway
            port:
              number: 80
```

---

# Next Steps

1. **Create project structure:**

```
kafka-food-pipeline/
├── docker-compose.yml
├── k8s/
│   ├── kafka/
│   │   ├── cluster.yaml
│   │   └── topics.yaml
│   ├── postgres/
│   │   ├── statefulset.yaml
│   │   └── service.yaml
│   ├── api-gateway/
│   │   ├── deployment.yaml
│   │   └── service.yaml
│   ├── image-analysis/
│   │   └── deployment.yaml
│   ├── food-matching/
│   │   └── deployment.yaml
│   ├── result-service/
│   │   └── deployment.yaml
│   ├── configmap.yaml
│   ├── secrets.yaml
│   └── ingress.yaml
├── services/
│   ├── api-gateway/
│   │   ├── Dockerfile
│   │   ├── main.py
│   │   └── requirements.txt
│   ├── image-analysis/
│   │   ├── Dockerfile
│   │   ├── main.py
│   │   └── requirements.txt
│   ├── food-matching/
│   │   ├── Dockerfile
│   │   ├── main.py
│   │   └── requirements.txt
│   └── result-service/
│       ├── Dockerfile
│       ├── main.py
│       └── requirements.txt
├── frontend/
│   ├── index.html
│   └── app.js
```

```
└── docs/
    └── architecture.md
```

2. **Start with Phase 1 this weekend**
3. **Document learnings as you go**
4. **Prepare demo script**
5. **Send to RxFood when complete** (optional)

---

**END OF PRD**