Document Processing Platform

Architecture Summary

This event-driven, microservices architecture is capable of handling 5M documents/day requirement while providing:

- Performance: 58 docs/sec throughput
- Reliability: 99.9% up-time with automatic failover and recovery
- Scalability: Linear scaling to 50M+ docs/day without architecture changes
- Flexibility: Multiple API patterns serve different user personas effectively
- Maintainability: Modular design enables independent component updates

---

System Requirements

Volume: 5 million documents per day (~58 docs/second)

Processing: 4 ML models - Politics Detection, Language Detection, Company Mentions, Sentiment Analysis (English only)

Input: Documents received every 1 minute by internal process

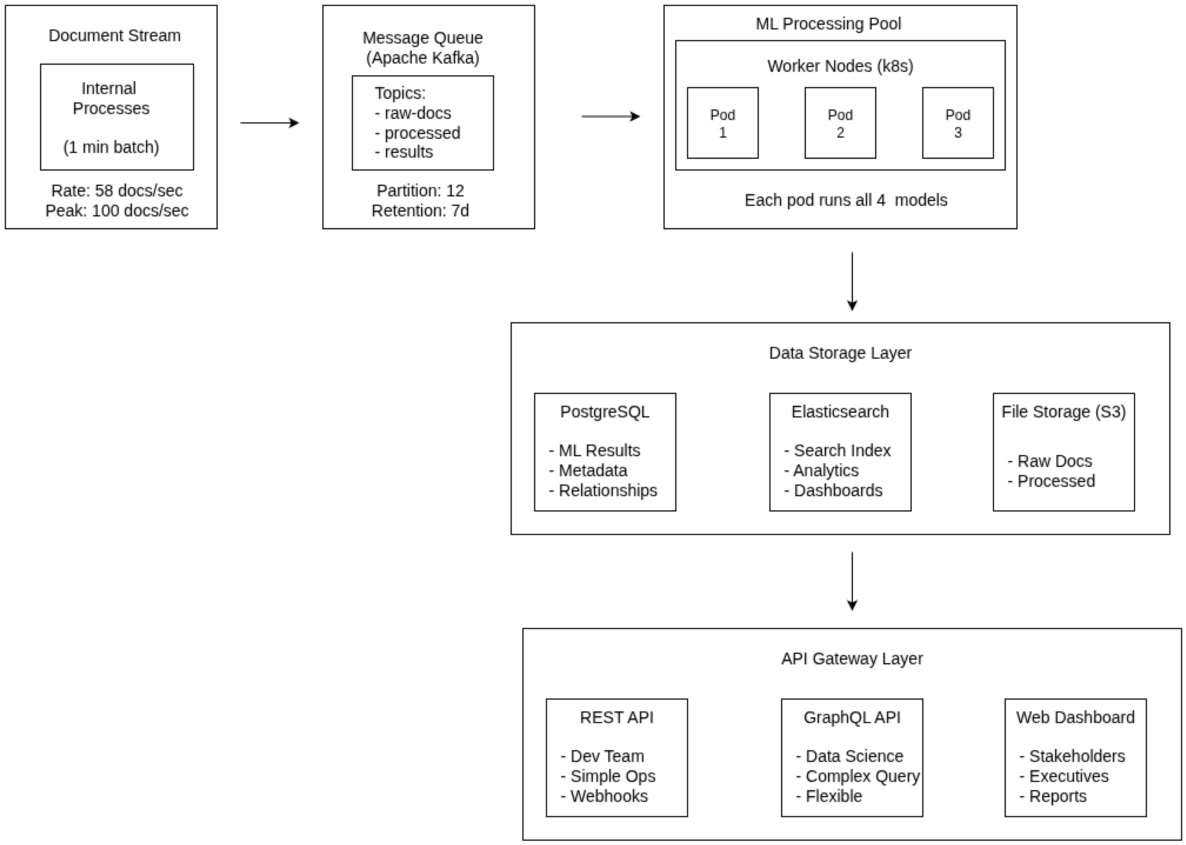
Dependencies: Sentiment analysis runs only if language is English

Access: Results accessible by development team, data scientists, and internal stakeholders

---

System Architecture

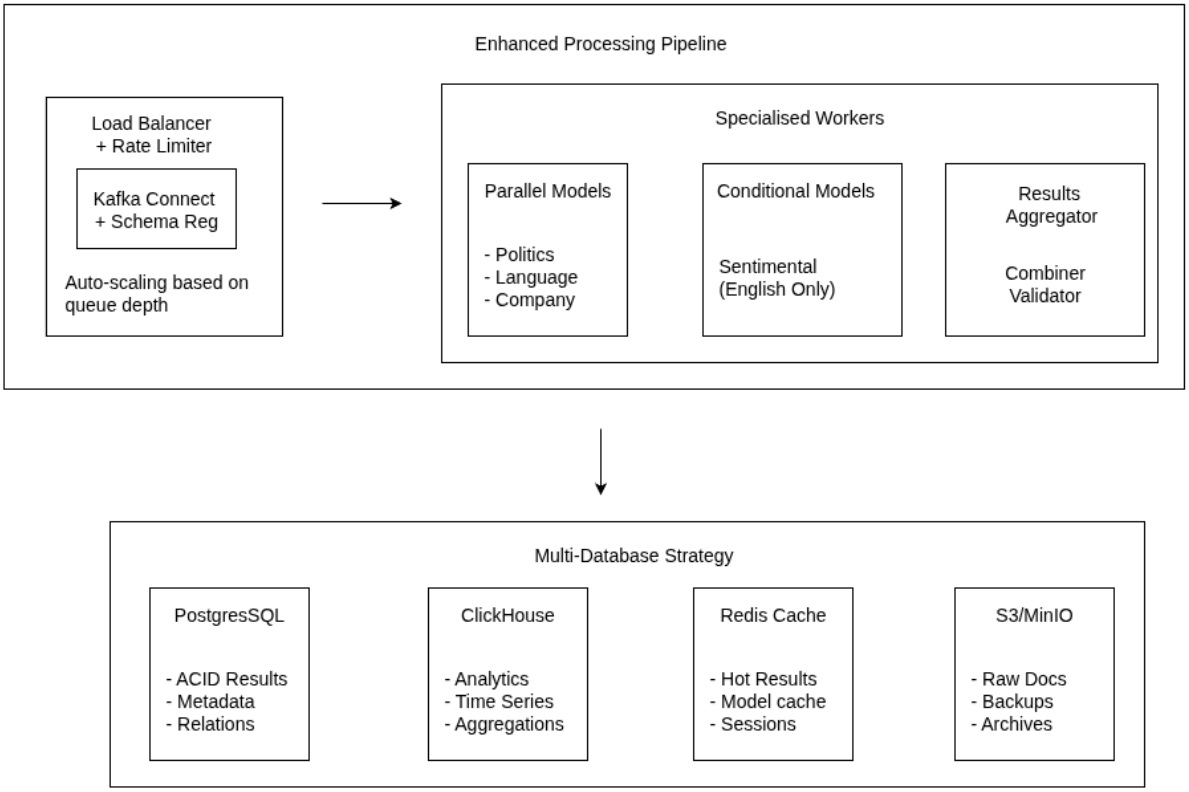
Phase 1: Initial architecture (MVP)



Processing Flow:

1. Documents arrive every minute → Kafka ingestion
2. Kubernetes workers consume from Kafka
3. Each worker runs Politics, Language, Company models in parallel
4. If language = English → Run sentiment model
5. Store results in PostgreSQL + index in Elasticsearch
6. Serve via multiple API endpoints

Phase 2: Optimised Architecture (High Scale)



Key improvements

- Specialised workers: Separate pods for parallel vs conditional models
- Smart caching: Redis for frequently accessed results
- Analytics engine: ClickHouse for fast aggregations
- Auto-scaling: Dynamic scaling based on queue depth and CPU/memory

Technology Stack

- Apache Kafka

- Kubernetes
- PostgresSQL
- ClickHouse
- Redis
- FastAPI + GraphQL

Apache Kafka

Handles 100K+ msgs/sec, fault-tolerant, replay capabilities for model retraining

Alternative: RabbitMQ (lower throughput but simpler)

Kubernetes

Auto-scaling, resource isolation, rolling deployments for ML models

Alternative: Docker swarm

PostgreSQL

ACID compliance, complex queries, JSON support for ML results

Alternative: MongoDB (document-native but eventual consistency)

ClickHouse

Significantly faster analytics than PostgresSQL, time-series optimised

Alternative: BigQuery (cloud-native but vendor lock-in)

Redis

Sub-milliseconds response, perfect for ML model caching

Alternative: Memcached (simpler but less features)

FastAPI + GraphQL

High performance, type safety, flexible querying for data scientists

Alternative: Flask + REST (simpler but less efficient)

Design Justification

- **Scalability:** Kafka + Kubernetes handle 10x current load with linear scaling
- **Fault Tolerance:** Message queuing ensures no data loss, dead letter queues for failures
- **Performance:** Parallel processing reduces latency, caching improves response time
- **Flexibility:** Multiple APIs serve different user needs without interference
- **Observability:** Each component has monitoring, tracing, and alerting built-in
- **Cost Efficiency:** Auto-scaling prevents over-provisioning, spot instances for ML processing

Limitations

- **Cold Start Latency:** ML models need 30-60 seconds to warm up
- **Memory Requirements:** 4 models per pod requires 8GB+ RAM
- **Single Point of Failure:** Language detection failure blocks sentiment analysis
- **Data Consistency:** Eventual consistency between PostgreSQL and ClickHouse
- **Model Versioning:** No A/B testing framework for model updates
- **Geographic Distribution:** Single-region deployment may not scale globally

Future Improvements

- **Model Serving:** TensorFlow Serving or Seldon for optimized inference
- **Feature Store:** Feast for consistent ML features across models
- **Model Monitoring:** Evidently AI for drift detection and model health
- **Streaming Analytics:** Apache Flink for real-time aggregations
- **Multi-Region:** Kafka MirrorMaker for global document processing
- **ML Ops:** Kubeflow for model training and deployment pipelines
- **Intelligent Routing:** Route documents based on content type
- **Batch Optimisation:** Dynamic batching based on model characteristics
- **Cost Optimisation:** Spot instances with graceful degradation
- **Real-time Alerts:** ML-based anomaly detection on processing metrics