System Design Document

Scalable Kubernetes-Based Algorithmic Trading Platform

1. Executive Summary

This document outlines the architecture and system design for a scalable, production-ready algorithmic trading platform. The platform is designed to support multiple clients executing strategies on their individual broker accounts (e.g., MT5), with centralized control and decoupled signal delivery. Built on Kubernetes, the system emphasizes modularity, fault-tolerance, and horizontal scalability.

2. Goals & Timeline

Objective:
To complete and deploy the production-ready version of this architecture within 2 months.

Milestones:
- Week 1–2: Infrastructure setup (Kubernetes, storage, Kafka)
- Week 3–4: Client pod communication, strategy broadcasting, initial testing
- Week 5–6: Monitoring, logging, scaling validations
- Week 7–8: End-to-end dry runs, QA, and production deployment

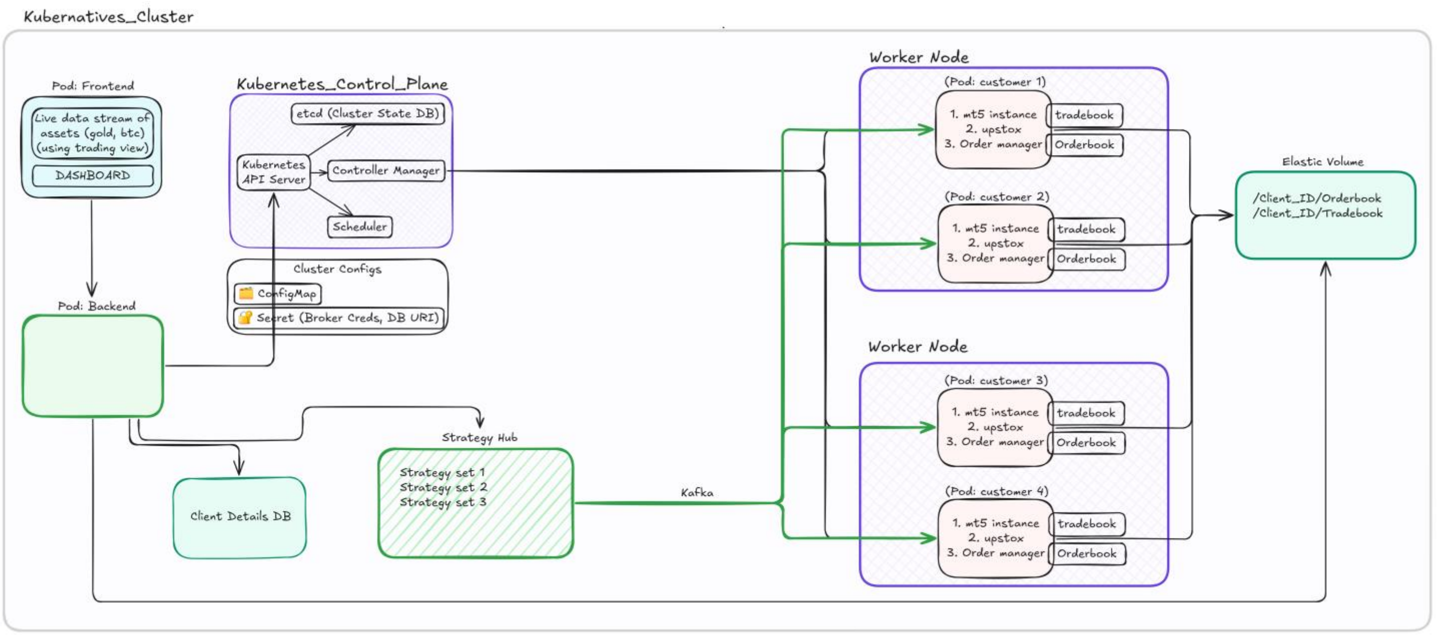
KPIs:
- <100ms signal-to-execution latency
- 99.9% system uptime
- 10,000 concurrent clients (pods) supported

3. System Overview

The architecture follows a control plane and execution pod model. Each client has an isolated execution environment. Signals are generated centrally by a Strategy Hub and sent to client pods via Kafka. Trade and order data are stored in isolated directories within a shared Elastic Volume.

Key Components:
- Strategy Hub (Strategy Service Pod with multiple sets)
- Kafka (message broker)
- Client Execution Pods (with MT5, signal listener, order manager)
- Elastic Volume (EFS/NFS)
- Monitoring Dashboard (Prometheus, Grafana, Loki)

Architecture Diagram



4. Component Breakdown

A. Strategy Hub
- Runs 3–4 sets of strategies concurrently
- Each set broadcasts trading signals via Kafka
- Clients subscribe to a designated strategy set stream (e.g., Set 1, Set 2, etc.)
- Example: 100 clients can be subscribed to Set 1, another 100 to Set 2, and so on

B. Kafka Message Broker
- Topic-based pub/sub architecture
- Each strategy set has a unique topic/channel
- Client pods only consume messages from their assigned strategy set
- Ensures real-time delivery and scalability of signals

C. Client Pods
- Each pod runs:
  - MT5/Upstox instance
  - Signal Listener (Kafka subscriber)
  - Order Manager
  - Tradebook + Orderbook writers
- Writes to isolated `/client_ID/...` directories in EFS

D. Elastic Volume (EFS)
- Shared, POSIX-compliant, supports concurrent writes
- Organized by client ID, with isolated write paths

E. Monitoring and Dashboard
- Prometheus for metrics
- Grafana for dashboards
- Loki for centralized log aggregation and querying
- Real-time dashboard may optionally include TradingView feed for price tracking

5. Tech Stack

- Orchestration: Kubernetes (GKE/EKS/self-hosted)
- Backend: FastAPI, Python
- Message Broker: Kafka
- Broker APIs: MT5 terminal (Windows runner) or Upstox APIs
- Storage: Elastic File System (AWS EFS or NFS)
- Monitoring: Prometheus, Grafana, Loki

6. Resource Requirements

Role	Headcount	Skills Required
Backend Developer	1 (can be 2)	Python, FastAPI, messaging protocols
DevOps Engineer	1 (can be 2)	K8s, Helm, CI/CD, storage (EFS/NFS), observability setup
QA Engineer	1	Staging, pod testing, signal tracking

7. Risks & Mitigations

Risk	Mitigation Strategy
Kafka topic bottlenecks	Use partitioned topics; scale horizontally
Broker disconnection	Retry logic, failover scripting in pod signal listeners
Volume write conflict	Isolated directories per pod
MT5 terminal instability	Use watchdog in pod to restart failed processes

8. Development Plan (8 Weeks)

W1–2: Infrastructure & Integration
- Deploy Kubernetes cluster (or finalize infra provider)
- Deploy Kafka cluster (Bitnami Helm chart or managed)
- Set up EFS/NFS and validate shared write access
- Integrate pre-built MT5 + OrderManager container into client pod spec
- Design Kafka topic schema and strategy-to-client assignment logic

W3–4: Strategy Hub Development
- Build Strategy Hub to run 3–4 strategy sets
- Enable each set to publish signals to a Kafka topic
- Test client subscriptions to appropriate strategy topics
- Build mapping logic between strategy sets and subscribing clients

W5–6: Signal Execution Pipeline Testing
- Launch pilot client pods (5–10) subscribing to different strategy sets
- Validate live signal ingestion, MT5 execution, and EFS writes
- Add error handling and retry logic inside signal listener

W7: Monitoring + Observability Setup
- Deploy Prometheus, Grafana, and Loki stack
- Expose metrics and logs from FastAPI, Kafka, and MT5 terminal containers
- Create alert rules for latency, failures, and Kafka consumer lag

W8: QA + Staging-to-Prod Go-Live
- Setup staging environment mirroring production
- Conduct performance/load tests simulating 100s of clients
- Validate end-to-end signal flow, storage writes, and failure recovery
- Finalize production deployment strategy and rollback mechanisms

9. Post-Finalization Discussion Points

- Final decision on infra provider (GCP/AWS/Self-Hosted)
- CI/CD setup approach (GitHub Actions, ArgoCD, etc.)
- Budgeting for Kafka deployment (self-managed vs managed)
- Rollout strategy for different asset classes (e.g., equities, crypto, forex)
- Expansion design for scaling up to 50K pods