Hedge Funder — Full Architecture System.

16.09.2025

—

Collins Dada

C.T.O (Chief Technology Officer)

PROJECT OVERVIEW

Complete architecture, diagrams, component responsibilities, data models, API sketches, deployment plan, and operational notes for the Hedge_Funder automated trading & supervision platform.

TABLE OF CONTENT

System overview (high level)

Hedge_Funder is split into four logical layers:

- **Ingress & Signals Layer**: Telegram signal parsers, social scrapers (Twitter, Reddit), webhook listeners (ngrok during development / public HTTPS in prod), and manual signal interface from admin UI.
- **AI & Orchestration Layer**: n8n or a similar orchestration agent, AI model(s) for signal completion and TP/SL optimization, and the decision engine (rules + model outputs) that produces final actionable orders.
- **Execution Layer**: Broker adapters (MT5 via Wine on EC2, Binance REST/WebSocket adapters), order manager, position tracker and reconciler.
- **Presentation & Storage Layer**: Frontend (Admin + Customer dashboards), Supabase DB (Postgres + Auth), caching layer (Redis + Dragonfly), and notification bots (Telegram/WhatsApp).

All layers are connected with secure APIs, event-driven messages, and persistent recording of every action to ensure auditability.

Components & responsibilities

Ingress

- **Telegram Listener**: Connects to Telegram groups using Bot API or MTProto client; captures messages, stores raw message + metadata, enqueues for parsing.
- **Social Scraper**: Streams Twitter (X) and Reddit posts via APIs; filters finance-related content and sentiment scoring pipeline.
- **Webhook Receiver**: Public-facing endpoint(s) for 3rd-party or manual inputs; during dev this may be ngrok.

Preprocessing & Parser

- **Signal Parser Service**: Stateless microservice (Python/Node) that extracts symbol, side, qty, tp/sl, timeframe, expiry from messages. Emits either ParsedSignal (complete) or IncompleteSignal events.

AI / Orchestration

- **AI Completion Service**: Receives IncompleteSignal, calls an LLM or model to complete fields (validate tickers, infer sizes/TP/SL), returns candidate signals with confidence score.
- **Decision Engine**: Combines parsed signals, AI output, market data (yfinance/finnhub), and risk policy to prepare OrderIntent.
- **n8n Orchestrator (Optional)**: For low-code flows and integrations (e.g., call external models, notify bots, call execution endpoints).

Market Data

- **Market Data Fetcher**: Pulls price/volume/indicators from yfinance, Finnhub, and newly added Stockpulse and optionally a paid streaming feed. Provides snapshot & historical data for analysis.

Execution

- **Order Manager**: Validates order intent, transforms to broker-specific payload, and sends to broker adapter.
- **Broker Adapters**:
    - MT5 Adapter: Communicates with MT5 terminal running on EC2 (Wine). Uses local API bridge (e.g., mt5gateway) that receives REST/GRPC and inputs into the MT5 terminal.
    - Binance Adapter: Uses Binance REST & WebSocket; handles signing, order retries, and rate limiting.
- **Position Tracker**: Subscribes to broker websockets / polling; updates position states & reconciles with database.

Persistence & Cache

- **Supabase (Postgres)**: Stores users, accounts, trade history, audit log, settings, and ML training data.
- **Redis / Dragonfly**: Cache of real-time positions, account balances, rate-limiting, and ephemeral request states.

Notifications & Observability

- **Telegram Bot**: Sends alerts to admin(s) when trades are placed, closed, or when exceptions occur.
- **WhatsApp Bot**: Mirror of Telegram notifications for admin (via Business API / 3rd party gateway).
- **Metrics & Logs**: Prometheus + Grafana (metrics), Loki or Elasticsearch (logs). #Optional

Frontend

- **Admin UI**: Manage accounts, watch live positions, view charts, force-stop/re-enable trading, create manual signals.
- **Customer UI**: OAuth login, link trading accounts, view P&L, history, and account health indicators.

Sequence Flows (core flows)

A — Signal auto-trade flow (complete signal) ⬇

```
sequenceDiagram
participant TG as Telegram
participant Parser as Parser
participant DE as DecisionEngine
participant OM as OrderManager
participant BK as BrokerAdapter
participant PT as PositionTracker
participant DB as Supabase
```

participant UI as Frontend

participant TB as TelegramBot

TG->>Parser: new message

Parser->>DE: ParsedSignal

DE->>Market: Request price/indicators

Market->>DE: Price Data

DE->>OM: OrderIntent

OM->>BK: SendOrder

BK->>OM: OrderAck

OM->>DB: SaveTrade

BK->>PT: position update

PT->>DB: UpdatePosition

OM->>TB: NotifyAdmin

DB->>UI: update websocket (realtime)

B — Incomplete signal flow -> AI completion

sequenceDiagram

participant Parser

participant AI

participant DE

participant OM

Parser->>AI: IncompleteSignal

AI->>Parser: CompletedSignal + confidence

Parser->>DE: CompletedSignal

DE->>OM: OrderIntent

OM->>Broker: Execute

C-Deployment topology

flowchart LR

subgraph AWS

EC2[EC2 (Wine + MT5 Terminal)]

AppAPI[App API (K8s / single VM)]

Redis[Redis + Dragonfly]

Supabase[(Supabase)]

Metrics[Prometheus/Grafana]

end

Browser -->|HTTPS| LoadBalancer --> AppAPI

AppAPI --> Supabase

AppAPI --> Redis

AppAPI --> EC2

AppAPI --> TB[Telegram Bot]

AppAPI --> WA[WhatsApp Gateway]

AppAPI --> n8n

n8n --> AI[LLM / Model]

n8n --> AppAPI

Can be rendered in markup language

SECURITY NOTES

- We need to consider running App API as a container (Docker) on EC2 or EKS. Keep MT5 terminal on the *same* EC2 host or in a dedicated EC2 that AppAPI can reach via secure local network.
- Use Security Groups to restrict ports; expose only required endpoints.

Data models (Supabase / Postgres highlights)

Tables (high-level)

- users (id, email, oauth_provider, oauth_id, name, role)
- accounts (id, user_id, platform, platform_account_id, credentials_ref, status)
- raw_signals (id, source, raw_message, received_at, payload_json)
- parsed_signals (id, raw_signal_id, symbol, side, size, tp, sl, timeframe, status)
- trades (id, account_id, broker_order_id, symbol, side, entry_price, qty, tp, sl, status, opened_at, closed_at, pnl)
- positions (id, account_id, symbol, qty, entry_price, current_price, pnl, updated_at)
- audit_logs (id, actor_id, action, payload, created_at)
- model_feedback (id, signal_id, predicted_values, actual_outcome, reward)

Indexes & partitions

- Partition trades by month for performance.
- Index positions(account_id, symbol) and trades(account_id, opened_at).

API contract sketches (sample endpoints)

Auth

- POST /auth/oauth/callback ⟶ handles OAuth and creates user session.

Admin

- POST /admin/accounts ⟶ add broker account (body: platform, credentials_ref)

- GET /admin/accounts ⟶ list accounts + balances
- GET /admin/accounts/:id/positions ⟶ real-time positions
- POST /admin/signal/manual ⟶ send manual signal
- POST /admin/accounts/:id/disable-trading ⟶ emergency stop

Signals

- POST /signals/telegram ⟶ webhook for captured Telegram messages
- GET /signals/:id ⟶ view parsed signal

Execution

- POST /execute ⟶ admin or orchestrator pings to execute an order intent

Realtime

- WS /realtime ⟶ authenticated websocket using JWT to stream positions and trades

Caching & realtime strategy (Redis + Dragonfly)

- Use Redis (primary) for ephemeral state: account connection statuses, rate limiting tokens, temporary order intents, and lock mechanisms.
- Use Dragonfly as a cached vector store or larger in-memory dataset if using similarity search for signals or time series snapshots.
- Use Redis pub/sub (or Redis Streams) to broadcast position updates to the App API and WebSocket server for real-time UI push.
- Cache recent price snapshots in Redis to reduce calls to Finnhub/yfinance during decision windows.

AI integration patterns

- **LLM (Prompt-based)**: For completion of incomplete signals: send message, context (symbol list, recent prices), and ask for missing fields. Return choices + confidence.
- **Model + Heuristics**: For TP/SL optimization: lightweight regression or classification model that takes volatility, ATR, historical win-rate, and returns recommended TP/SL and recommended size.
- **Feedback loop**: Save `model_feedback` with real outcomes to retrain or fine-tune models centrally.
- **Safety**: Always run a risk filter rule set (max exposure per account, max daily trades) before executing orders.

Security & secrets

- Keep broker credentials encrypted in Supabase using PG crypto or use a separate secrets manager (AWS Secrets Manager / HashiCorp Vault).
- Use JWTs for frontend auth with short TTLs and refresh tokens stored securely.
- For MT5: keep the MT5 terminal locked behind VPN or local-only interface. Expose only a secure API gateway to the Order Manager.
- Use role-based access control (RBAC) in the admin UI (super-admin, ops, viewer).
- Audit all actions into `audit_logs`. Immutable logs help when investigating bad trades.

CI/CD pipeline (GitHub Actions)

- **On PR**: Run unit tests, lint, static analysis, and container build.
- **On Merge to main**: Build Docker images, push to ECR (or DockerHub), run integration tests (staging), deploy to staging. Run smoke tests (signal parsing, execute dry run orders).
- **On Manual Promotion**: Deploy to prod after approval; migrate DB if necessary.

Sample steps:

1. checkout
2. setup python/node
3. install deps
4. run tests
5. build docker image
6. push image
7. apply terraform/ansible (if infra-as-code)
8. post-deploy smoke tests

Monitoring, logging, observability

- **Metrics**: Prometheus counters for orders sent, orders failed, latency, execution success rate, cache hit rate.
- **Tracing**: OpenTelemetry to trace a signal through Parser → AI → Decision → Execution.
- **Logging**: Structured logs saved to Loki/Elastic. Ensure correlation IDs are present.
- **Alerts**: PagerDuty