ENGINEERING RELEASE NOTES · EXECUTION ENGINE

Algotoria Executor — Release Notes

The SMA Execution Engine that turns a single strategy signal into matched positions across every client account on OKX, Binance and Bybit. This note records what has been built, how it behaves, and where it now sits on the road to live trading.

FROM	AUDIENCE	REFERENCE	STATUS
Algotoria Limited Engineering	Algotoria staff & partners	ALG-EXEC-RN-2026-06	Development complete Entering staging

Release summary. The Executor — the engine that executes trades on the exchanges — is complete and has passed local validation on demo accounts across all three venues. It replaces the earlier third-party execution component end to end. The next phase is staging and scaled testing, followed by a controlled live launch on small balances. Nothing in this document is a performance representation or an offer; it is an engineering milestone note.

Highlights of this release

Target-position execution
Accepts strategy-agnostic fractional targets and brings every subscribed account to the same exposure, sized to each account's own balance.

Three exchanges, one engine
Hand-built clients for OKX, Binance and Bybit written directly against each venue's live documentation — no third-party trading library.

Horizontally scalable
Stateless workers process accounts in parallel; capacity grows by adding workers, bounded only by each exchange's rate limits.

Maker-first, fee-aware
Posts passive limit orders to earn maker rebates, escalating to market orders only when a position would otherwise be left behind.

Built to recover
Holds no internal state: the exchange is the single source of truth, so the engine can restart, redeploy or fail over without placing duplicate trades.

Observable & controllable
Full metrics, dashboards and alerting, with day-to-day control through the Algotoria Telegram bot.

1. What we are releasing

The Executor is the execution layer of the Algotoria platform — the engine that takes a trading decision and makes it real on the exchanges, on every client account, safely and auditably.

Algotoria runs Separately Managed Accounts (SMAs): client funds stay in the client's own exchange account, and Algotoria trades them under a management mandate. A single master strategy decides what the portfolio should hold; the Executor's job is to make every subscribed account mirror that decision in proportion to its own size — placing, amending and cancelling the orders needed to get there, and keeping each account on target as the strategy moves.

1.1 The target-position model

The strategy does not send orders. It sends a **target position**: a list of instruments — currently 32 perpetual-futures markets — each with a fraction of account equity to hold, for example *"hold 10% of the account's margin balance long in BTC"*. The signal is deliberately account-agnostic; it describes a portfolio shape, not a quantity of contracts.

When a target arrives, the Executor translates that shape into concrete order quantities for each account, on each exchange, using that account's current balance at the moment the signal is received. This is what lets one strategy drive hundreds of accounts of different sizes at once, and what lets the engine work on any exchange and with any signal source, provided the signal conforms to the expected format.

> **Why targets, not trade copying.** The engine is not trying to reproduce the master account's trade history. It only cares about the destination — the position each account should hold right now. If the market moves and the target changes before earlier orders fill, the engine abandons the stale work and converges on the new target instead of completing trades that are no longer wanted. This avoids needless round-trips and the cost of executing decisions the strategy has already revised.

1.2 Two kinds of signal

- **Snapshots** — sent on a regular heartbeat (every five minutes). They state the full current target for every instrument. They keep every account anchored to the truth and double as a liveness check on the signal source.
- **Signals** — sent when the target actually changes. They carry the delta plus a snapshot to fall back on, and trigger immediate convergence.

1.3 Scope and boundaries

The Executor's remit is execution only. It places orders on linear perpetual futures, brings positions to target and polices its own operational health. It does **not** generate signals, set leverage policy or perform investment risk management — those live in other parts of the platform. Execution is maker-first: the engine posts passive limit orders by default and uses market orders only in defined fallback cases (Section 3).

2. How a signal becomes trades

Signals reach the engine over an authenticated HTTP endpoint (`POST /v1/executor/signals`). The path from arrival to execution is split into a fast intake stage and a separate work stage, so that accepting a signal is never blocked by the time it takes to trade.

2.1 Intake — fast and stateless

Two interchangeable intake services sit behind a load balancer. Whichever receives a signal validates its signature, checks it for duplicates, normalises the instrument names for each destination exchange, writes the result to the in-memory store and hands the work to a queue. It then acknowledges the sender — typically within a quarter of a second — without having touched any exchange. No single intake service holds state, so signals can land on either one, and capacity is added simply by running more of them.

2.2 Convergence — done by parallel workers

A separate pool of worker processes drains the queue. They group the accounts by exchange, split them into batches and bring each account to its target in parallel. Workers, too, are stateless and interchangeable: if one stops mid-task, another picks the work up. The engine is not constrained by its own speed — it is deliberately paced to stay within each exchange's request limits (Section 4.4).

2.3 Runtime topology

COMPONENT	ROLE	NOTES
Intake services (×2)	Receive, validate and enqueue signals	Stateless; load-balanced round-robin
Workers (×2)	Run the convergence loop against exchanges	64 concurrent worker threads in total; horizontally scalable
Scheduler (×1)	Triggers periodic convergence, heartbeats and audits	Singleton
Redis	Hot state, idempotency keys, generation counters, queue	In-memory with durable persistence
ClickHouse	Positions, orders, signal/alert history, analytics	Clustered — 3 shards, 2 replicas each
PostgreSQL	Account metadata, instrument registry, configuration	Source of account/reference data

> **The exchange is the single source of truth.** The engine deliberately does *not* remember which orders it has open. Every convergence pass re-reads live positions and open orders from the exchange before acting. This is the cornerstone of safe recovery: after any restart, redeploy or fault, the engine simply re-reads reality and converges — it never replays a stale internal queue.

3. Convergence & order execution

The convergence loop is the heart of the engine. For each account it compares the live position against the fixed target, measures the gap (the **drift**) and places the minimum orders needed to close it — then re-checks, and repeats until the account is on target or a safety condition stops it. It runs on a regular cycle (every five minutes by default) and also fires immediately when a new target arrives.

3.1 Maker-first, with a disciplined fallback

By default the engine posts **passive (post-only) limit orders**. This earns the maker rebate, avoids paying the spread, and is well suited to Algotoria's cadence, which is not high-frequency. Each instrument carries its own price adjuster that adapts independently to its liquidity, so a thinly-traded market can be quoted more aggressively than a deep one without affecting the rest.

If a passive order has not filled by the time it should have, the engine runs a **catch-up**: it cancels the stale order for that one instrument and finishes the remaining quantity with a market order, capped at a small number of attempts. A market order that comes back completely unfilled — a sign of no liquidity — stops at once and raises a warning rather than burning retries. In local testing, limit orders did almost all of the work; market fallback was the exception.

3.2 Drift, residuals and partial fills

"On target" is defined per instrument in that market's own minimum tradeable increment, not as a blanket percentage — so each symbol's tolerance respects its lot size. The engine can fill a position partially and top it up on a later cycle as liquidity appears, which is exactly what makes the target model robust across different markets and account sizes. In testing, residual drift typically settled at only a few dollars per account.

3.3 Batching and amendment

To use each exchange's request budget efficiently, orders are grouped into the largest batch the venue allows and sent together; where an exchange supports amending a live order in place, the engine amends rather than cancel-and-replace.

EXCHANGE	BATCH PLACEMENT	IN-PLACE AMEND	TRADING-LIMIT SCOPE
OKX	Up to 20 orders / batch	Yes	Per sub-account
Binance	Up to 5 orders / batch	Yes	Per sub-account and per IP
Bybit	Up to 20 orders / batch	Yes	Per sub-account and per IP

All orders are placed in one-way (net) position mode with cross-margin; the engine verifies and, if necessary, corrects the account's position mode before it begins trading, and refuses to start if positions or live orders already exist that would make the change unsafe.

4. Resilience & correctness

Because the engine handles real money across many accounts and three independent venues, most of its complexity is in failing safely. Four mechanisms carry that weight.

4.1 A four-level health model

The engine tracks health at four nested levels, so a problem stays contained at the smallest level it belongs to and never blocks everything else:

- **Signal provider** — is the strategy feed live and current?
- **Exchange** — governed by a circuit breaker that moves a venue between *healthy*, *degraded* and *unavailable* based on its recent error rate, with an automatic recovery probe.
- **Account** — isolates an account with, say, a revoked API key.
- **Instrument** — isolates one bad market so the other 31 keep converging.

4.2 No duplicate orders, ever

Every order the engine places carries a deterministic tag encoding the account, instrument and signal generation. Before placing, the engine re-reads open orders from the exchange and reconciles against these tags, so a worker that restarts mid-batch attributes any stranded order correctly instead of placing it twice. This directly addresses a real duplicate-order incident observed during earlier integration work, and is backed by idempotency keys on intake and a minimum spacing floor between repeat placements.

4.3 Graceful behaviour when the signal stops

If the strategy heartbeat goes quiet, the engine escalates in stages rather than reacting abruptly: it first warns the operations channel, then stops opening new orders, and only after a prolonged outage (24 hours) begins closing positions and halting trading. Because the engine re-reads the exchange on every cycle, a brief signal gap is usually invisible — given the trading cadence, a few missed minutes rarely matter.

4.4 Living within exchange limits

Exceeding an exchange's request limits risks throttling or an IP ban — operationally equivalent to an outage. The engine therefore budgets requests precisely. Each venue's limits are modelled by their true scope (per sub-account, per IP, and per request type), shared across all workers through the in-memory store, and the engine works to roughly 60% of each published limit, holding the rest as headroom. The published limits themselves are not static, which is why the engine also monitors the exchanges' documentation for changes (Section 6.3).

> **Recovery in one sentence.** On any restart or fault, the engine reads the live target and the live exchange state, compares them, and places only the orders needed to close the difference — so it self-heals toward the correct position instead of trusting a remembered plan.

5. Security & data protection

Execution touches client credentials and live capital, so the security posture is built in rather than bolted on.

5.1 Authenticating the signal

Every inbound signal is authenticated with an HMAC-SHA256 signature and a per-message idempotency key. The signature proves the signal genuinely came from Algotoria's strategy source; the idempotency key lets the engine discard a duplicate — for instance one resent after a network hiccup — before it does any work.

5.2 Secrets that never travel in the clear

The values used to verify and protect signals are stored only as encrypted strings, and the keys that encrypt them are held separately in a dedicated secrets manager (HashiCorp Vault), with the master key in a cloud key-management service. Even an attacker who reached a host or a log would not find the live verification values — they exist on the wire and at rest only in encrypted form. This design predates the Executor itself and was hardened on the signal source first.

5.3 Stateless by design

Because no service holds long-lived secrets or trading state in memory beyond what it needs for the task in hand, the blast radius of any single compromised component is small, and components can be replaced or restarted freely.

5.4 Disciplined logging

Logs are rich enough to explain what the engine did and why, but are written to avoid capturing personal data or full exchange responses. Identifiers that could be sensitive are partially masked, and only the fields needed for diagnosis are retained. The goal throughout is to keep operational visibility high while keeping the data that could leak as small as possible.

5.5 Demo mode

A single switch routes every exchange call to the venue's demo or testnet environment, with all other behaviour identical. This is how the engine is validated, and how new accounts can be exercised end to end without real funds.

> **Alignment with the ISMS.** These controls — authenticated signals, segregated key management, least-data logging and demo isolation — sit within Algotoria's ISO/IEC 27001 Information Security Management System. This note describes engine behaviour; the controlling policies live in the ISMS.

6. Control & observability

6.1 Day-to-day control through Telegram

The engine is operated through the Algotoria Telegram bot — there is no separate admin console. Every command runs through the same safe pipeline, can target one account or all, and is queued so that two commands never collide on the same account. Destructive actions require explicit confirmation, and the bot deliberately resets its context between commands so a stale tap cannot be misread.

COMMAND	WHAT IT DOES
`subscribe`	Associate an account with a signal source
`start` / `stop`	Begin or halt trading; `stop` can keep or close positions, gently or by force
`status`	Show equity, targets, current positions, drift, open orders and alerts
`set-risk-factor`	Scale a single account's exposure down (e.g. a lower-risk institutional mandate)
`rebalance`	Re-size targets after a deposit or withdrawal without waiting for the next signal
`set-source`	Switch an account's signal source (e.g. primary to backup, or between strategies)
`restart` / `reload-registry`	Reload configuration or the instrument map without disturbing positions

The per-account **risk factor** lets a client trade at reduced exposure — useful for an institutional account that wants, say, no more than a third of the default positioning — by scaling only that account's targets. A hard leverage cap is enforced when trading starts.

6.2 Metrics, dashboards and alerts

The engine emits detailed metrics that feed four Grafana dashboards — *execution quality*, *drift control*, *system health* and *recovery & alerts* — covering the maker/taker mix, fill and convergence quality, latency, exchange error rates and the live state of every account. Critical and warning conditions are pushed to the operations (NOC) channel on Telegram with a structured payload and a suggested action. The dashboards will be refined with the trading desk during staging; the underlying metrics are already captured.

6.3 Watching the exchanges for change

OKX, Binance and Bybit change their APIs, instruments and limits frequently — sometimes on a few weeks' notice. A separate scheduled service mirrors each venue's official documentation, compares it against what the engine relies on, and flags the specific changes that need a code update. This keeps the hand-built exchange clients accurate over time.

7. How it was built

The Executor is an AI-native build: it was developed almost entirely by AI coding agents, under close human direction, over roughly three months. The approach is itself part of the release, because it shapes how the engine will be maintained.

~3 mo	~43k	~4,800	~150k
ELAPSED BUILD	LINES OF PYTHON	AUTOMATED TESTS	LINES INCL. SPEC & TESTS

7.1 Specification-first, model-independent

Work began with a deep specification — roughly four thousand lines describing every state, edge case and reconciliation guarantee — written and cross-checked by AI before code was produced. Crucially, the specification was kept in plain, vendor-neutral form, with no lock-in to any single AI tool. When one model weakened mid-project, the team handed the same specification to another and continued without losing momentum. Every non-trivial change was independently cross-validated by a second agent rather than self-reviewed.

7.2 The build in proportion

The work was broken into roughly 46 feature plans. The result is about 43,000 lines of Python across some 147 modules, around 50,000 lines of automated tests (about 4,800 tests run on the engine alone) and roughly 35,000 lines of specification and documentation — about 150,000 lines in total. The hardest stretch, predictably, was the convergence loop with its cancellations and rate-limit accounting; that is where the cross-validation discipline earned its keep.

7.3 Why we built the exchange clients by hand

Earlier execution work relied on a third-party trading library and, before that, on an external contractor's component that mounted heavy infrastructure for every signal and could not cancel work in flight. Both were retired. The Executor talks to each exchange directly, with clients written against each venue's own live documentation and pinned to it. This avoids waiting on a third-party library to catch up with exchange changes, removes a dependency from a money-handling path, and — together with the documentation-monitoring service — keeps the engine current.

7.4 A clean-room engine

The Executor is kept structurally separate from the older platform code. It carries its own configuration, data access and models, and shares only a short, documented list of base utilities. Because it handles real money, this isolation bounds risk in both directions and keeps the engine independently maintainable.

8. Current status & what comes next

Development is complete and the engine has been validated locally, end to end, on demo accounts across all three exchanges. It is not yet trading client funds; the remaining work is deliberate, staged hardening before any live capital is involved.

8.1 What has been verified

- The full command set and the complete signal-to-execution path, exercised on demo accounts on OKX, Binance and Bybit.
- Convergence quality on OKX in particular was strong, with positions brought to target quickly and residual drift of only a few dollars.
- Deliberate fault injection — disconnecting the signal, forcing odd states, switching mid-flight — to confirm the engine fails safely and recovers.

8.2 Known limitations at this stage

- Local testing was constrained by network latency and VPN routing, which made some venues (notably Bybit) harder to exercise locally; these are testing-environment artefacts, not engine faults, and clear on staging.
- Some demo instruments lack liquidity, so a small number of thin markets could not be fully converged in the demo environment.
- Broker-rebate identifiers must be confirmed with each exchange's broker desk and configured before the first live order — a release prerequisite, not an engine change.

8.3 Roadmap to live

PHASE	GOAL
Staging deployment	Run the engine on the staging cluster under realistic network conditions
Scale testing	Drive 50, 100 and 200+ demo accounts to surface rate-limit, batching and capacity behaviour
Dashboard refinement	Tune dashboards and alerts with the trading desk
Controlled live launch	Trade small balances across all three venues, then widen gradually

> **In short.** The Executor is built, tested and ready to enter staging. It executes a single strategy across many accounts and three exchanges, maker-first and fee-aware, with strong guarantees against duplicate trades and a design that recovers by reading reality rather than trusting memory. The path from here to live trading is a measured sequence of staging, scale testing and a small-balance launch.