thinklio

Technical Overview

Architecture, execution model, security, and deployment for technical evaluators

March 2026 · Version 1.0
Prepared by Novansa OÜ

hello@thinklio.com

Contents

1. System Architecture

Thinklio is a multi-service platform written entirely in Go. The architecture separates concerns across six stateless services that communicate via Redis Streams (asynchronous events) and direct HTTP/gRPC calls (synchronous requests). All durable state lives in PostgreSQL and Redis; services themselves hold no state, enabling straightforward horizontal scaling.

Technology Stack

Component	Technology	Notes
Language	Go	All services. True concurrency, single static binary, excellent operational tooling
Database	PostgreSQL 16 + pgvector	System of record. Row-level security. Event tables partitioned by time
Cache / Events / Jobs	Redis 7 with Streams	Multi-tier caching, event bus, rate limiting, active job store
Containers	Docker	Target image size under 20 MB per service (Go static binaries on Alpine)
Orchestration	Docker Compose / Coolify	Single-host initially. Multi-host via configuration, not code changes
Infrastructure	Hetzner Cloud (EU)	Nuremberg data centre. EU data residency
Object Storage	Cloudflare R2	S3-compatible. File attachments, documents, database backups
Admin UI	React / Next.js	Agent Studio, governance config, usage monitoring
Monitoring	Prometheus + Grafana + Alertmanager	Metrics, dashboards, and alerting

2. Service Responsibilities

Each service has a single area of responsibility. Inter-service communication uses Redis Streams for event-driven patterns and direct HTTP/gRPC for request-response patterns.

Service	Responsibility
Gateway	Channel adapters, API surface routing, authentication, rate limiting, webhook delivery. The only public-facing service.
Agent	LLM reasoning, harness execution, delegation, job notification handling. Orchestrates the think/act/observe loop.
Context	Multi-layer knowledge retrieval, context assembly, token budgeting, job context injection.
Tool	Tool registry, policy enforcement, execution (internal, external, agent-as-tool), delegation governance.
Queue	Background tasks (knowledge extraction, summarisation), scheduling, job timeout monitoring.
Usage	Cost tracking per step/interaction/user/team/org, budget enforcement, delegation cost rollup.

Communication Patterns

Asynchronous (Redis Streams): Event publishing and consumption. Each service publishes events to named streams and consumes from streams relevant to its responsibilities. Consumer groups enable parallel processing with at-least-once delivery semantics within the group.

Synchronous (HTTP/gRPC): Direct service-to-service calls for request-response patterns where the caller needs an immediate result. Examples: Agent service requesting context assembly, Agent service requesting tool execution.

Services run on an internal Docker network. The Gateway is the only service exposed to the public internet.

3. Durable Execution Harness

Every agent interaction runs inside a durable execution harness that tracks each reasoning step independently. This is the core runtime that makes Thinklio's governance, cost tracking, and crash recovery possible.

The Execution Loop

Each interaction progresses through a sequence of steps. Each step has an independent state machine (created ≣ running ≣ success/failed) and is persisted before execution begins.

Context. Assemble multi-layered knowledge context. The Context service retrieves relevant facts from agent, organisation, team, and user knowledge layers, applies token budgeting, and returns a context bundle.

Think. LLM call for reasoning and decision-making. The agent receives the context bundle and the user's message, reasons about the best approach, and decides which tools to invoke.

Act. Tool execution. The chosen tool is invoked through the Tool service, which evaluates policies, enforces trust levels, and tracks costs before execution. Three modes: immediate (synchronous), deferred (dispatches a job), and interactive (multi-turn within a single interaction).

Observe. LLM call to synthesise tool results. The agent reviews what the tool returned and decides whether to respond, invoke another tool, or take further action.

Respond. Deliver the response to the user via the originating channel.

Extract. Queue background knowledge extraction. The Queue service processes the interaction asynchronously, extracting structured facts for the appropriate knowledge layer.

Step-Level Durability

Each step is persisted to the database before execution begins. If the Agent service restarts mid-interaction, execution resumes from the last completed step. If a tool call fails, the harness retries that specific step without re-executing preceding steps. Failure metadata is captured for every step: timeout, error, governance denial, budget exceeded, user cancellation, or service unavailability.

Mixed Execution Modes

A single interaction can use different execution modes for different steps. An agent might make a quick API lookup (immediate), dispatch a longer research task (deferred, returning a job reference), and respond with the lookup result plus a note that the research is in progress.

4. Data Model

Thinklio's data model is designed around a few key principles: agents are first-class entities independent of organisations, knowledge has clear ownership and privacy boundaries, and every operation is auditable.

Core Entities

Entity	Description
User	Platform-global identity. Can belong to multiple organisations and teams.
Organisation	Billing and governance entity. Owns policies, budgets, and administrative controls.
Team	Group of users within an organisation sharing context and agents.
Agent	First-class platform entity. Independent of organisations. Assigned to contexts via AgentAssignment.
AgentAssignment	Links agent to context (user, team, org) with scope config and per-assignment tool restrictions.
Interaction	Tracked unit of work triggered by a user message or job notification. Contains steps.
Step	Individual operation within an interaction. Independent state machine with cost data.
Job	Deferred work that outlives a single interaction. Contains subjobs, watched by observers.
KnowledgeFact	Structured fact with scope, semantic embedding, confidence scoring, access tracking.
Tool	Capability available to agents. Types: internal, external, agent.
Event	Immutable, append-only log entry. Partitioned by time. The system of record.

Agent Independence

Agents exist independently of any organisation or team. They are assigned to contexts via the AgentAssignment entity. This means a solo user can create a personal agent without needing an organisation, and the same agent can serve multiple contexts with isolated knowledge per context.

5. Four-Layer Knowledge Model

Every agent interaction draws from up to four knowledge layers with clear ownership, privacy, and precedence rules.

Layer	Owner	Mutability	Visibility
Agent	Agent creator	Configured at setup, updated via learning	All users of this agent
Organisation	Org admins	Curated, mostly static	All org members
Team	Team (collective)	Grows organically from interactions	Team members only
User	Individual user	Grows from user interactions	Private to that user only

Precedence and Privacy

Organisation policies override all other layers. Then agent knowledge, team knowledge, and user knowledge in descending priority. User knowledge is never visible to other users, even within the same team. When a user leaves a team, their user knowledge goes with them.

Knowledge Representation

Knowledge facts are structured as subject/predicate/value triples with scope, confidence scoring, and semantic embeddings (pgvector) for vector similarity search. The Context service performs token-budgeted context injection, prioritising by relevance, recency, and confidence.

6. Security Model

Defence in depth: least privilege, fail closed, audit everything, and structural isolation.

Authentication

Three external authentication paths, all resolving to the same platform identity and authorisation model. Standard user auth: email/password (bcrypt) or OAuth/SSO (Google, Microsoft, SAML for enterprise). JWTs with 24-hour expiry and refresh token rotation.

Four-Level Authorisation

Level	Question	Enforcement
Platform	Can this user/system access this resource?	Authentication, role-based access control
Agent	Can this agent perform this action?	Tool permissions, capability level, org policies
Assignment	Is this action further restricted here?	Per-assignment tool restrictions (narrow only)
Step	Can this step execute right now?	Trust level, budget, rate limits, delegation rules

Data Isolation

Tenant isolation is enforced at multiple layers simultaneously: database (row-level security), application (runtime context assertions), cache (tenant-scoped keys), and network (internal Docker network; Gateway is the only public-facing service). During delegation, the delegate assembles its own context independently, preventing privilege escalation through delegation chains.

Rate Limiting

Scope	Default Limit
Per IP (gateway)	100 requests/min
Per user messages	20 messages/min
Per agent tool execution	10 executions/min
Per agent LLM calls	30 calls/min
Channel/Platform API per key	60 requests/min

Scope	Default Limit
Integration API per tool	30 executions/min
Job dispatch per agent	10 jobs/min

GDPR Readiness

Data export, anonymisation, deletion, consent management, data minimisation, and configurable retention. All data stored in EU (Hetzner Cloud, Nuremberg). Immutable audit logs. User knowledge isolation is structural.

7. Three External API Surfaces

Three distinct API surfaces, each serving a fundamentally different integration pattern. These are not three versions of the same API.

Channel API

Purpose: Conversational access. An external system sends messages to an agent and receives responses.

Use cases: Mobile apps with in-app assistants. SaaS products with AI support. Internal tools with chat widgets.

Authentication: Authenticates as a user.

Platform API

Purpose: Orchestration and management. Programmatic access to creating agents, dispatching jobs, managing knowledge, querying usage.

Use cases: Service providers automating deployment. CI/CD pipelines. External workflow engines.

Authentication: Authenticates as an organisation or service account.

Integration API

Purpose: Bidirectional capability exchange. External systems register capabilities as tools and subscribe to events via webhooks.

Use cases: CRM systems registering as tools. Project management integrations. ERP and healthcare platforms.

Authentication: Authenticates as an external system.

All three surfaces resolve to the same authorisation model, governance framework, cost attribution, and audit trail. Each is versioned independently.

8. Agent Composition

Agents work together through an agent-as-tool model. From the invoking agent's perspective, delegating to another agent is identical to calling any other tool: same governance, cost tracking, and audit trail.

Delegation Governance

- Delegation can only narrow permissions, never widen them.
- Delegation depth limits are configurable at the organisation level (default: 3).
- Cycle detection runs at configuration time and at runtime.
- Knowledge isolation is maintained: the delegate assembles its own context independently.
- Costs from delegate agents roll up to the originating user and team.

Capability Levels

Level	Capabilities	Use Case
tools_only	Can use assigned tools. No workflow composition.	Q&A; agents, lookup assistants, policy checkers
workflow	Can compose multi-step workflows using tools.	Coordinators, reporting agents, research assistants
experimental	Can try novel combinations of tools and workflows.	Development and testing contexts
learning	Can codify successful workflows as reusable patterns.	Agents that improve their own processes

Capability levels are policy configuration, not code branching. The harness, governance, and accounting infrastructure is identical at every level.

9. Deployment and Scaling

Initial Deployment

Single Hetzner Cloud VPS (CPX31: 4 vCPU, 8 GB RAM) running all six services via Docker Compose. Coolify provides deployment management, environment configuration, and SSL termination. Target container image size under 20 MB per service.

Horizontal Scaling

All services are stateless. Horizontal scaling is a configuration change, not a code change.

Role	Services	Scaling Trigger
Edge	Gateway	Request volume, concurrent connections
Compute	Agent, Tool	LLM call volume, concurrent interactions
Infrastructure	Context, Queue, Usage	Knowledge retrieval load, background task depth
Data	PostgreSQL, Redis	Storage, query load, event stream throughput

Backup and Monitoring

PostgreSQL backups to local storage and Cloudflare R2. Redis RDB snapshots and AOF for event stream durability. The PostgreSQL event store is the authoritative record from which all state can be reconstructed. Prometheus metrics, Grafana dashboards, Alertmanager for notifications via Postmark.

10. Key Architectural Decisions

Recorded decisions that shaped the platform's design.

Go for all services. True concurrency, single static binary, excellent tooling. Chosen over TypeScript and hybrid approaches.

Agents as first-class entities. Independent of organisations. Assigned via AgentAssignment. Solo users need no organisation; same agent serves multiple contexts.

Redis Streams as event bus. Direct publish/consume eliminates a separate bus service. Removes a network hop and single point of failure.

Step-level state machine. Every operation is a step with independent lifecycle. Provides crash recovery, granular cost tracking, step-level retry, and complete audit trail.

Three API surfaces. Channel, Platform, and Integration serve fundamentally different patterns. Designed as first-class concerns from the start.

Per-assignment tool restrictions. Narrow only, never widen. Single agent serves different contexts with appropriately scoped capabilities.

Built from scratch. Prior TypeScript prototype informed the design but its single-user architecture was not a suitable starting point for a multi-tenant Go platform.

No calendar-bound timeline. Phases complete when success criteria are met. Quality over velocity.

Further Information

This document provides an architectural overview for technical evaluation. Detailed API specs, schema documentation, and deployment guides are available to early access partners.

Thinklio is in active pre-launch development. The architecture described represents the designed system; implementation is progressing through a phased roadmap.

Email	hello@thinklio.com
Web	thinklio.novansa.com