AAOS — Architecture Overview

AAOS — Architecture Overview

AAOS = ALAI Agent Operating System — Redesigned 2026-04-03 to collapse 10 virtual companies into 4 functional groups with pure orchestration.

The Transformation: 10 ? 4

Before AAOS, we had 10 virtual companies with overlapping responsibilities. Now we have **4 functional groups** with clear boundaries.

graph LR subgraph "Before (10 companies)" CC[CodeCraft] VZ[Vizu] DV[Datavera] FV[Finverge] SB[Skybound] PV[Proveo] SC[Securion] FF[FlowForge] AF[AgentForge] RS[Resolver] LX[Lexicon] end subgraph "After (4 groups)" BUILD["BUILD
(CodeCraft)"] REVIEW["REVIEW
(Proveo)"] OPS["OPS
(FlowForge)"] PLATFORM["PLATFORM
(AgentForge)"] end CC --> BUILD VZ --> BUILD DV --> BUILD FV --> BUILD SB --> BUILD PV --> REVIEW SC --> REVIEW FF --> OPS AF --> PLATFORM RS --> PLATFORM LX --> PLATFORM

Group Definitions

Group	Identity	Absorbs	Role
BUILD	CodeCraft	CodeCraft, Vizu, Datavera, Finverge, Skybound	Writes code — backend, frontend, data, fintech, product
REVIEW	Proveo	Proveo, Securion	READ-ONLY validation, QA, security audit
OPS	FlowForge	FlowForge	Infrastructure only — Docker, CI/CD, IaC, monitoring
PLATFORM	AgentForge	AgentForge, Resolver, Lexicon	AI system maintenance — RAG, HiveMind, docs

Key principle: John is a **pure orchestrator**. He delegates ALL execution to specialist agents.

CodeCraft Agent Board — 8 Specialists

Within the BUILD group, CodeCraft operates an **agent board** with 8 specialist agents:

graph TD A["Alem (CEO)"] --> J["John — Orchestrator"] J --> AL["architect-lead
(Opus)
decomposition, blueprint"] AL --> KA["kotlin-architect
(Sonnet)
Kotlin/Ktor backend"] AL --> NS["nextjs-specialist
(Sonnet)
Next.js 15 App Router"] AL --> DB["database-specialist
(Sonnet)
PostgreSQL, Flyway"] AL --> AA["api-architect
(Sonnet)
OpenAPI, REST, SDK"] J --> SR["security-reviewer
(Sonnet)
READ-ONLY security audit"] J --> QA["qa-specialist
(Sonnet)
Functional tests, DOM"] J --> DO["devops-specialist
(Haiku)
Docker, CI/CD, deploy"] J --> PL["PLATFORM group
(Sonnet)
RAG, HiveMind, docs"]

Agent	Domain	Model	Notes
architect-lead	Decomposition, blueprint, delegation	Opus 4.6	Can spawn other specialists
kotlin-architect	Kotlin/Ktor backend, services, DB	Sonnet	ALAI unified stack
nextjs-specialist	Next.js 15 App Router, RSC, Tailwind	Sonnet	Frontend + React
database-specialist	PostgreSQL, Flyway, migrations	Sonnet	Data layer
api-architect	OpenAPI, REST, SDK design	Sonnet	Integrations
security-reviewer	OWASP, auth, secrets	Sonnet	READ-ONLY audit
qa-specialist	Functional tests, DOM visibility	Sonnet	Quality gates
devops-specialist	Docker, CI/CD, Nginx, deploy	Haiku	Infrastructure

Model budget:

- **Opus 4.6** — Architect agents (system design, tech spec) + team leads
- **Sonnet** — All builders and validators (default for implementation)
- **Haiku** — Trivial tasks (file search, lint, git, DevOps)

John's Role: Pure Orchestrator

What John DOES:

1. Create MC task (`node ~/system/tools/mc.js add`)
2. Write GOTCHA gate (`/tmp/gotcha-task-{id}.md`)
3. Run RAG query before any action
4. Select correct specialist agent from `routing.json`
5. Read short agent reports (max 8 lines)
6. Run `qa-19.js` check after build
7. Report to Alem in **max 5 sentences**

What John NEVER does:

- Write code in `~/projects/**`
- Edit files in `~/projects/**`
- Run tests directly
- Fix bugs manually
- Deploy any service
- Call Bash commands on project source files

John is blocked from touching project source code. All implementation goes through specialist agents.

Pipeline Flow

```
SPEC → REVIEW → BUILD → REVIEW → OPS → PLATFORM
       spec check  specialist  qa-specialist   devops   learn
                               + security-reviewer
```

1. **SPEC** — John creates MC task + GOTCHA gate
2. **REVIEW** (H/CRIT only) — Spec validation before build

3. **BUILD** — Specialist agent writes code
4. **REVIEW** — qa-specialist + security-reviewer audit
5. **OPS** — devops-specialist deploys
6. **PLATFORM** — agentforge extracts learnings to HiveMind

Review cycles: Max 2 cycles before escalating to John → Alem

Task Metrics

Every MC task tracks metrics in `mission-control.db` table `task_metrics`:

Field	Type	Purpose
`task_id`	INTEGER	FK to tasks.id
`qa_score`	INTEGER	`/19` from qa-19.js
`token_cost_usd`	REAL	Anthropic API cost
`duration_seconds`	INTEGER	Wall time
`cache_hits`	INTEGER	RAG cache hits
`agents_spawned`	INTEGER	How many subagents
`rework_count`	INTEGER	How many review cycles

Purpose: Learning agent uses this to flag inefficient patterns.

Learning Agent

Tool: `~/system/tools/learning-agent.js`

Runs: Nightly at 02:00 via cron

Capabilities:

1. Analyzes task metrics (high token cost, low QA score, many rework cycles)
2. Flags patterns to HiveMind
3. Updates flywheel cache with common queries
4. Suggests agent improvements
5. Generates weekly summary report

Goal: System learns from its own execution.

RAG Flywheel

```
Question → SHA256 cache (flywheel.db) → HiveMind FTS/Qdrant → Ollama (ANVIL/FORGE) → Anthropic (fallback) → save answer back to cache
```

Databases:

- `flywheel.db` (36MB) — SHA256-keyed cache, fast hits
- `knowledge.db` (187MB) — Full RAG knowledge base
- `hivemind.db` (14K+ entries) — Structured intel + memory

Models:

- **ANVIL** — `localhost:11434` (Mac Studio M3 Ultra, 96GB)
- **FORGE** — `10.0.0.2:11434` (deepseek-r1:70b, qwen3:32b)
- **Anthropic** — Cloud fallback if local fails

Cache hit rate: 61% (as of 2026-02-24)

Config Files

File	Purpose
`~/system/agents/definitions/john-orchestrator.yaml`	John's identity + agent board definitions
`~/system/config/john-routing.json`	Domain → Group → Agent routing map
`~/system/rules/company-first-protocol.md`	Routing rules + group boundaries

Source of truth hierarchy:

1. `john-orchestrator.yaml` (agent board + groups)
2. `john-routing.json` (routing map)
3. `company-first-protocol.md` (protocol docs)

If drift is detected, `john-orchestrator.yaml` wins. Update routing → docs.

Archive

10-company model: Preserved in `~/system/archive/companies-pre-collapse-2026-04-03/`

Why collapsed: Too much routing complexity, unclear boundaries, token waste.

When collapsed: 2026-04-03 (AAOS v1.0)

Revision #2
Created 2026-04-02 23:22:56 UTC by John
Updated 2026-05-31 20:05:28 UTC by John