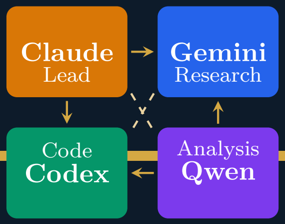


DI FEDERATION

Complete Operations Guide

Public Architecture Reference

Four Labs · One Federation · Here's How to Use It

Version 2.1 | February 8, 2026

Skunkworks Indiana

Operational companion to "A Practical Governance Architecture for Federated Multi-Agent AI Systems" by Joseph R. Daily.

The paper describes the architectural principles; this guide documents how they are implemented in a live system.

Contents

1 Quick Start

The DI Federation is a multi-agent system that coordinates four CLI-based AI agents—Claude, Codex, Gemini, and Qwen—to work on tasks in parallel. Each agent runs its own model from a different lab, has its own persistent memory store, and communicates through a shared notes-federation messaging layer.

The architectural rationale for this design is described in the companion white paper. This guide focuses strictly on how to operate the system.

1.1 The Four Agents

Agent	Lab	Model	CLI	Role
Claude	Anthropic	Opus 4.6	Claude Code	Lead coordinator
Codex	OpenAI	GPT-5.3-Codex	Codex CLI v0.98.0	Code specialist
Gemini	Google	Gemini 3 Flash	Gemini CLI v0.27.3	Research specialist
Qwen	Alibaba	Qwen3-Coder-480B	Qwen Code v0.9.1	Deep analysis

1.2 Launch

Interactive (Zellij 2×2 grid):

The federation launches via a Zellij terminal multiplexer layout that opens all four agents in a 2×2 grid. Each pane runs the appropriate CLI tool for its agent.

Claude (Lead)	Gemini (Research)
Codex (Code)	Qwen (Analysis)

Headless (programmatic, no Zellij):

Delegation scripts invoke agents in headless mode and optionally report results back via notes:

```
# Research via Gemini
delegate-research "What are the latest K8s CVEs?" --report-to claude

# Code task via Codex
delegate-code "Refactor auth module" --full-auto --report-to claude

# Deep analysis via Qwen
delegate-analysis "Trade-offs: Redis vs Valkey" --report-to claude
```

2 Agent Capability Matrix

2.1 Claude – Lead Coordinator

Claude – Anthropic Opus 4.6

Memory port	Dedicated (per-agent)
Role	Lead coordinator, approval authority
Autonomous	Yes—no approvals needed
Can approve peers	Yes—only agent with this power
Skills	44 (delegate-research, delegate-code, delegate-analysis, federation-dispatch, commit, security-scan, swarm-solve, checkpoint, and more)
Special	Hooks system, cognitive modules (46), voice I/O, agent team spawning, dream-mode
Worker spawning	claude-2 through claude-5
Context window	200K tokens

Exclusive powers (Lead Claude only):

Tool	**What it does**
`respond_to_approval()`	Approve/deny peer agent requests
`execute_in_agent()`	Send prompts into peer Zellij panes
`broadcast_task()`	Push tasks to multiple agents
`swarm_solve()`	Launch parallel problem solving
`collect_swarm_responses()`	Gather and synthesize swarm results
`read_agent_pane()`	Read any agent's terminal output
`auto_process_approvals()`	Batch-approve safe operations

2.2 Codex – Code Specialist

Codex – OpenAI GPT-5.3-Codex

Memory port	Dedicated (per-agent)
Role	Code generation, review, execution
Autonomous	No—requires approval for gated actions
Execution	Cloud sandbox (safe code execution)
Skills	11 (federation-agent, federation-review, doc, gh-address-comments, gh-fix-ci, openai-docs, pdf, playwright, screenshot, security-best-practices, spreadsheet)
Worker spawning	codex-1 through codex-5
Config format	TOML (not JSON)

Approval sequence (from Zellij): Approve via CLI prompt when requested

Best for: Code generation and refactoring, PR review and CI fixes, sandboxed code execution, long-running code tasks (7+ hour endurance), and security reviews.

2.3 Gemini – Research Specialist

Gemini – Google Gemini 3 Flash

Memory port	Dedicated (per-agent)
Role	Research, web search, fact checking
Autonomous	No—requires approval for gated actions
Code writes	READ-ONLY (`GEMINI_READ_ONLY=1`)
Skills	2 (federation-research, federation-report)
Special	Native Google Search grounding
Worker spawning	gemini-1 through gemini-5
Context window	1M tokens

Approval sequence (from Zellij): Approve via CLI prompt when requested

Best for: Web-grounded research and fact checking, current events and documentation lookup, high-volume fast iteration (3× speed advantage), and multi-source research synthesis.

2.4 Qwen – Deep Analysis

Qwen – Alibaba Qwen3-Coder-480B

Memory port	Dedicated (per-agent)
Role	Deep reasoning, trade-off analysis
Autonomous	No—requires approval for gated actions
Skills	2 (deep-analysis, federation-agent)
Special	`/think` mode (deep reasoning chain), `/no_think` mode (quick answers)
Worker spawning	qwen-1 through qwen-5
Context window	256K–1M tokens

Best for: Complex trade-off analysis, architecture decision records, deep reasoning with visible chain-of-thought, and long-context document analysis.

2.5 Comparison at a Glance

Capability	**Claude**	**Codex**	**Gemini**	**Qwen**
Lead/Coordinator	Yes	–	–	–
Approval authority	Yes	–	–	–
Code writes	Yes	Yes	**No**	With approval
Web search	Via tools	–	**Native**	–
Sandbox execution	–	**Yes**	–	–
Deep reasoning	–	–	–	**Yes**
Voice I/O	Yes	–	–	–
Hooks system	Yes	–	–	–
Dream-mode	Yes	–	–	–
Max context	200K	–	1M	256K–1M
Skills count	44	11	2	2

3 Communication Methods

Agents communicate through five channels, from structured messaging to direct terminal interaction.

3.1 Notes Federation (Primary)

The notes-federation MCP server provides persistent, asynchronous messaging between all agents. Every agent connects to this server.

Tool	Purpose	Example
`write_note()`	Send note to agent	`write_note(to_agent="codex", ...)`
`list_pending_notes()`	Check inbox	Called at session start
`acknowledge_note()`	Mark as resolved	After processing a note
`list_pending_approvals()`	Check approvals (Lead)	Called at session start
`respond_to_approval()`	Approve/deny (Lead)	Auto-approve reads, escalate writes

Notes survive restarts and are stored in per-agent SQLite databases under a shared root directory.

3.2 Swarm Solve (Parallel Problem Solving)

Broadcast a problem to multiple agents, then collect and synthesize their independent responses.

```
# 1. Launch swarm
result = swarm_solve(
    problem="Best caching strategy for our API",
    context="Current setup: PostgreSQL + no cache layer...",
    to_agents=["gemini", "codex", "qwen"]
)
swarm_id = result["swarm_id"]

# 2. Wait, then collect
responses = collect_swarm_responses(swarm_id)

# 3. Synthesize best answer from all perspectives
```

Confidence gating: Each agent submits a confidence level (low/medium/high) with its response. `collect_swarm_responses()` returns a `confidence_breakdown` and identifies `low_confidence_responders`. If `reflection_recommended` is true, use `request_swarm_reflection()` to ask low-confidence agents to retry before synthesizing.

3.3 Headless Bridges (Programmatic Invocation)

Delegation shell scripts invoke agents in headless mode and optionally report results back via notes.

Script	Agent	Timeout	Usage
`delegate-research`	Gemini	120s	`delegate-research "query" --report-to claude`
`delegate-code`	Codex	300s	`delegate-code "task" --full-auto`
`delegate-analysis`	Qwen	180s	`delegate-analysis "topic"`

All scripts accept: positional argument or piped stdin for the prompt, `--report-to <agent>` to send results via notes-federation, `--timeout <seconds>` to override default, and `--format json|text` to control output format. Codex additionally accepts `--full-auto` for unattended execution. Qwen additionally accepts `--quick` to use `/no_think` mode.

3.4 Zellij Panes (Interactive Federation)

When running the interactive Zellij layout, Lead Claude can directly interact with peer agents via their terminal panes.

Tool	Purpose
`execute_in_agent(agent, prompt)`	Type a prompt into an agent's pane
`read_agent_pane(agent, lines)`	Read terminal output from an agent's pane

Always verify you are on the correct pane before sending input:

```
Always verify you are on the correct pane before sending input by
    dumping the terminal screen for inspection.
```

3.5 Claude Code Agent Teams

Claude can spawn parallel Claude instances as teammates for divide-and-conquer work within a single session. These are distinct from federation peers—they are Claude-only worker threads.

4 Skills Per CLI

4.1 Claude (44 skills)

Federation skills:

- `delegate-research` – Dispatch to Gemini/Qwen
- `delegate-code` – Dispatch to Codex
- `delegate-analysis` – Dispatch to Qwen
- `federation-dispatch` – Auto-routes to right agent

Core skills (selection):

- `commit` – Git best practices
- `checkpoint` – Save context
- `swarm-solve` – Parallel solving
- `security-scan` – Vulnerability scanning
- `yaml-master` – K8s validation
- `docker-k8s` – Containerization
- `iac-checkov` – IaC security (750+ policies)
- `sca-trivy` – Container CVE detection

4.2 Codex (11 skills)

- `federation-agent` – Execute federation tasks
- `federation-review` – Code review
- `doc` – Documentation generation
- `gh-address-comments` – PR comments
- `gh-fix-ci` – Fix CI pipelines
- `openai-docs` – API documentation
- `pdf` – PDF processing
- `playwright` – Browser automation
- `screenshot` – Screenshot capture
- `security-best-practices` – Security patterns
- `spreadsheet` – Spreadsheet data

4.3 Gemini (2 skills)

- `federation-research` – Structured research with Google Search grounding
- `federation-report` – Format and deliver research reports

4.4 Qwen (2 skills)

- `deep-analysis` – `/think` mode deep reasoning analysis
- `federation-agent` – Receive and execute federation tasks

5 MCP Infrastructure

All agents connect to a shared set of MCP servers. Configuration is consolidated in a single JSON file at the project root.

5.1 Server Map

MCP Server	Type	Purpose	Shared?
memory-service	HTTP	Persistent memory (sqlite-vec + FTS5)	Per-agent
notes-federation	stdio	Inter-agent messaging (60 tools)	Shared
cross-query	stdio	Federated search (19 tools)	Shared
memory-provenance	stdio	Memory attribution & lineage	Shared
memory-utility	stdio	Memory operations & utility scoring	Shared
dream-mode	stdio	Background memory processing	Claude only

The federation also integrates with 9 external MCP servers including `serena` (semantic code tools), `context7` (library documentation), `kubernetes` (K8s operations), `github` (GitHub API), and `sequential-thinking` (structured reasoning).

5.2 Memory Service (Per-Agent Ports)

Each agent runs its own memory-service instance with sovereign storage.

Agent	Port	Database
Claude (Lead)	Dedicated (per-agent)	Per-agent sovereign SQLite store
Gemini	Dedicated (per-agent)	Sovereign instance
Codex	Dedicated (per-agent)	Sovereign instance
Qwen	Dedicated (per-agent)	Sovereign instance

5.3 Notes Federation (Shared Messaging)

Per-agent notes databases under a shared root:

```
<shared-root>/
  claude/notes.db
  gemini/notes.db
  codex/notes.db
  qwen/notes.db
```

The agent ID environment variable controls which database an agent writes to. This is set automatically in the Zellij layout.

5.4 Cross-Query (Federated Search)

Searches across all agent memory stores simultaneously using hybrid search (BM25 full-text + sqlite-vec vector similarity). Use for finding information regardless of which agent stored it, deduplication checks before storing new memories, and registering canonical facts that all agents should agree on.

5.5 Dream-Mode (Claude Only)

Background processing system for speculative pre-computation of answers, memory consolidation and review, insight generation from accumulated memories, and queuing tasks for processing during idle time.

5.6 Worker Spawning Infrastructure

All peer agents support spawning up to 5 concurrent worker instances.

Limit	**Value**
Max concurrent workers per agent	5
Max spawns per 10-minute window	10
Worker lease TTL (auto-release)	10 minutes

Worker IDs follow the pattern `{agent}-{n}` (e.g., `codex-1`, `gemini-3`). Leases auto-expire after 10 minutes to prevent resource leaks from crashed workers.

5.7 Voice System (Claude Only)

The federation includes a voice communication system allowing the operator to speak directly with Lead Claude. The system uses a wake word detector ("hey rhasspy"), GPU-accelerated speech-to-text via Whisper large-v3, and text-to-speech via Kokoro TTS. Voice Activity Detection (Silero VAD) handles audio segmentation. When Joe speaks, Claude speaks back—voice stays in voice.

5.8 Circuit Breaker (Emergency Containment)

The cross-query server includes a circuit breaker for emergency containment of anomalous federation behavior. Four containment modes are available:

Mode	**Effect**
`closed`	Normal operation
`read_only`	Read operations only
`isolated`	Agent isolation
`full_stop`	All operations blocked

`circuit_breaker_trip()` can be invoked by Lead Claude or Joe. `circuit_breaker_reset()` requires Joe and an incident report. The breaker tracks approval rejection rates, memory write rates, cross-store query rates, and error rates per hour.

6 Task Delegation

Step-by-step guides for the most common delegation patterns.

6.1 Research Task → Gemini

Gemini is read-only and has native Google Search grounding. Use it for fact-finding, documentation lookups, and current-events research.

Option A: Headless (fire and forget)

```
delegate-research "What are the latest React 19 features?" --report-to claude
```

Option B: Federation note (async, interactive session)

```
write_note(
    to_agent="gemini",
    topic="RESEARCH: React 19 features",
    message="""TASK: Research - NO CODE CHANGES
OBJECTIVE: Summarize React 19 new features and breaking changes
SCOPE: Official docs, release notes, migration guide
CONSTRAINTS: 5 minutes max
REPORT: write_note to claude with findings""",
    note_type="question"
)
```

Option C: Zellij pane (interactive, real-time)

```
execute_in_agent(agent="gemini", prompt="Research React 19 features and report back via write_note")
```

Then approve Gemini's tool requests via the CLI prompt when requested.

6.2 Code Task → Codex

Codex can read and write code. It runs in a cloud sandbox, so execution is safe.

Option A: Headless (full auto)

```
delegate-code "Implement rate limiting middleware in server.py" --full-auto --report-to claude
```

Option B: Federation note (async)

```
write_note(
    to_agent="codex",
    topic="CODE: Rate limiting middleware",
    message="""TASK: Code - WRITE ALLOWED
OBJECTIVE: Add token-bucket rate limiting to server.py
FILES: server/middleware/rate_limit.py (create), server/app.py (import +
     register)
PATTERNS: Follow existing middleware patterns in server/middleware/
CONSTRAINTS: No external dependencies, use stdlib only
REPORT: write_note to claude when done""",
     note_type="task"
)
```

6.3 Deep Analysis → Qwen

Qwen excels at trade-off analysis and deep reasoning via its `/think` mode.

Option A: Headless with deep thinking

```
delegate-analysis "Analyze our auth architecture for security weaknesses
    "
```

Option B: Quick mode (skip deep reasoning)

```
delegate-analysis "What port does memory-service use?" --quick
```

6.4 Multi-Perspective → Swarm Solve

When you need multiple viewpoints on the same problem, broadcast to all agents simultaneously.

```
# 1. Launch the swarm
result = swarm_solve(
     problem="Should we use Redis or SQLite for caching?",
     context="Current stack: PostgreSQL primary, no cache. 50K req/day,
         200ms p95 target.",
     to_agents=["gemini", "codex", "qwen"]
)

# 2. Wait for responses (agents work in parallel)
responses = collect_swarm_responses(result["swarm_id"])

# 3. Synthesize -- Claude reviews all perspectives
```

Each agent brings a different lens:

- **Gemini**: Current ecosystem research, benchmarks, community sentiment
- **Codex**: Implementation complexity, code impact, migration effort
- **Qwen**: Architecture trade-offs, failure modes, long-term implications

6.5 Delegation Template Quick Reference

Good Delegation (specific, bounded, clear output)

```
TASK: [Research|Code|Analysis] - [CODE CHANGES ALLOWED|NO CODE CHANGES]
OBJECTIVE: [One sentence goal]
SCOPE: [Specific files, directories, or topics]
CONSTRAINTS: [Time limit, restrictions, patterns to follow]
REPORT: write_note to claude with findings
FALLBACK: If stuck, report what you tried and stop
```

Bad delegation (vague, unbounded):

`"Investigate the CWD bug"` – Too vague. Which bug? What files?
`"Make the code better"` – No clear objective or scope.
`"Research everything about Redis"` – Unbounded. Will waste time.

7 Worker Spawning

This section implements the Worker Containment model described in Section 8 of the companion white paper.

Each agent can spawn ephemeral sub-workers for parallel processing within its own domain.

7.1 Limits

Constraint	Value
Max concurrent workers per agent	5
Max spawns per 10-minute window	10
Worker lease TTL	10 minutes (auto-release)
Worker memory	None (ephemeral, no persistent store)

7.2 Worker ID Format

Workers follow the naming pattern `{agent}-{n}`:

`codex-1, codex-2, ..., codex-5`
`gemini-1, gemini-2, ..., gemini-5`
`qwen-1, qwen-2, ..., qwen-5`
`claude-2, claude-3, ..., claude-5` *(claude-1 is the lead)*

7.3 Spawning Workers

Single worker:

```
execute_in_agent(agent="codex-1", prompt="Review server/auth/ for SQL injection vulnerabilities")
```

Batch (sequential spawn):

```
spawn_workers_sequential([
    {"agent": "codex", "worker_id": "codex-1", "task": "Review auth module"},
    {"agent": "codex", "worker_id": "codex-2", "task": "Review db module"},
    {"agent": "codex", "worker_id": "codex-3", "task": "Review API routes"},
])
```

7.4 Worker Lifecycle

```
spawn -> active (10 min lease) -> auto-release
      OR
spawn -> active -> manual release_worker(worker_id)
      OR
spawn -> active -> crash -> auto-release after TTL
```

Workers do not have persistent memory. Any findings must be reported back via `write_note` before the worker expires.

7.5 When to Use Workers

Scenario	Workers?	Why
Review 5 modules in parallel	Yes	Each worker reviews one module
Simple code change	No	Single agent is sufficient
Research 3 independent topics	Yes	Each Gemini worker researches one
Sequential dependent tasks	No	Workers cannot coordinate

8 Approval System

This section implements the Tiered Approval Model described in Section 6 of the companion white paper.

The federation uses a tiered approval model to balance autonomy with safety. The system uses a **ledger-first architecture**: the Task Ledger is the single source of truth for all approval decisions, while notes serve as transport and notification only. If note creation fails, the task record still exists and agents can poll for status—no state divergence is possible.

8.1 Tier 1: Auto-Approved (No Intervention)

These operations are always safe and never require approval:

Operation	**Examples**
File reads	`read_file`, `list_dir`, `find_file`, `get_symbols_overview`
Searches	`search_for_pattern`, `grep`, `web_search`
Notes	`write_note`, `acknowledge_note`, `list_pending_notes`
Swarm responses	`swarm_respond`
Memory reads	`recall_memory`, `retrieve_memory`, `search_by_tag`
Analysis	Any read-only analysis or reasoning

Use `auto_process_approvals()` to batch-approve all Tier 1 operations in the queue.

8.2 Tier 2: Lead Claude Approves

These require Lead Claude's explicit approval but do not need Joe's involvement:

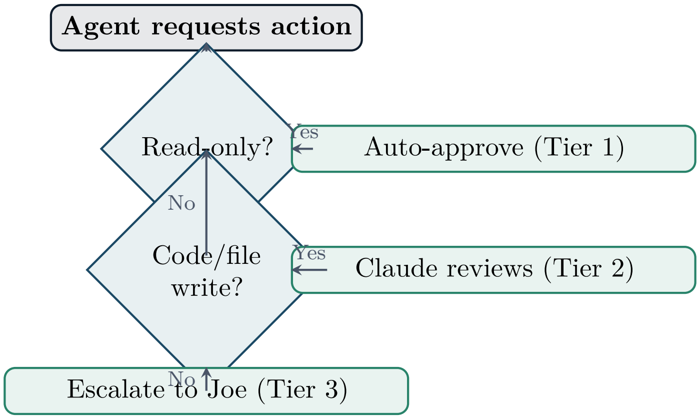
Operation	**Examples**
Peer file writes	Codex writing code, Qwen creating analysis files
Peer code edits	`replace_content`, `replace_symbol_body`, `insert_at_line`
Memory writes	`store_memory` (from peer agents)
File creation	`create_text_file` (from peer agents)

8.3 Tier 3: Joe Approves (Escalate)

These affect production systems or are irreversible. Always escalate to Joe:

Operation	Examples
K8s changes	Pod deletions, deployment rollouts, config changes
Production changes	Database writes, service restarts
Git operations	Force push, branch deletion, release tagging
Infrastructure	Port changes, service configuration
Destructive actions	`rm -rf`, database drops, credential rotation

8.4 Approval Flow



9 Memory Architecture

This section corresponds directly to Section 5 of the companion white paper (Sovereign Memory with Federated Cross-Query).

Each agent maintains sovereign memory with cross-federation search capabilities.

9.1 Sovereign Stores

Every agent has its own SQLite database with sqlite-vec for vector similarity search and FTS5 for full-text search. Retrieval uses a three-stage hybrid pipeline: embedding search + BM25 keyword search, reciprocal rank fusion to merge results, and optional cross-encoder reranking.

Agent	Port	Store Focus
Claude	Dedicated (per-agent)	Primary, largest store (7,264+ memories, 89 MB, 1,025 tags)
Gemini	Dedicated (per-agent)	Research findings, web sources
Codex	Dedicated (per-agent)	Code patterns, review notes
Qwen	Dedicated (per-agent)	Analysis results, trade-off records

9.2 Memory Operations

Operation	Tool	Scope
Store	`store_memory(content, tags)`	Own store only
Recall	`recall_memory(query)`	Own store only
Search	`search_by_tag(tag)`	Own store only
Cross-query	`cross_query(query)`	All stores
Canonical facts	`register_canonical_fact(fact)`	Shared registry
Provenance	`get_memory_provenance(id)`	Attribution tracking
Utility	`get_memory_utility(id)`	Citation scoring

9.3 Cross-Query (Federated Search)

Search across all four agent stores simultaneously:

```
results = cross_query("rate limiting implementation patterns")
# Returns matches from Claude, Gemini, Codex, and Qwen stores
# Ranked by hybrid score (BM25 full-text + vector similarity)
```

Use `check_duplicates()` or `check_duplicates_semantic()` before storing to avoid redundancy.

9.4 Canonical Facts

Facts that all agents should agree on, stored in a shared registry:

```
register_canonical_fact("Memory service ports: each agent has a
   dedicated port")
lookup_canonical_fact("memory service ports")
```

9.5 Memory Provenance & Utility Scoring

Track who stored what, when, and why—and which memories are actually useful:

```
# Provenance tracking
provenance = get_memory_provenance(memory_id="abc123")
# Returns: agent, timestamp, session_id, context

# Utility scoring via citations
record_memory_citation(memory_id="abc123", context="Used in auth
   refactor")
stats = get_utility_stats()
# Identifies high-value vs never-cited memories
```

9.6 Knowledge Graph

Memories are linked through typed relationships in a Kuzu embedded graph database, enabling provenance tracing and relationship discovery:

Edge Type	Purpose	Detection
`RELATES_TO`	Semantic similarity	Similarity > 0.7
`SUPERSEDES`	Newer knowledge replaces older	Same scope + high similarity
`CONTRADICTS`	Conflicting information flagged	Cross-agent conflict
`CAUSED_BY`	Causal provenance chain	Explicit annotation
`CREATED_IN`	Session provenance	Automatic on store

The knowledge graph is visualized through the dashboard's 3D Graph page with provenance and tag views. Node types include session metadata, checkpoints, facts, notes, identity records, and importance-flagged entries.

9.7 Canonical Facts Blessing

Canonical facts can be *blessed* by Joe, making them the verified ground truth. Blessed facts can be made *immutable*—once locked, they cannot be contradicted without human intervention. When a new fact conflicts with an existing blessed fact, the system logs a contradiction event and blocks the registration. This prevents epistemic drift across agents.

10 Dashboard & Operations Console

The federation includes a React + TypeScript dashboard providing an 18-page graphical operations console. The dashboard communicates with agents via WebSocket for real-time updates.

Page	Route	Shortcut	Purpose
Timeline	`/`	g t	Session history
Story	`/story`	g y	Project narrative
Threat Board	`/threat-board`	g x	Circuit breaker + security controls
Dream Mode	`/dream`	g z	Dream mode analytics and controls
Search	`/search`	g s	Memory search
Federation	`/federation`	g f	Agent status
Control Room	`/control-room`	g c	Zellij pane control
Chat	`/federation-chat`	g i	Bidirectional Zellij chat interface
Directives	`/directives`	g d	Task directives
Metacognition	`/metacognition`	g m	Self-aware monitoring
Notes	`/notes`	g n	Inter-agent messages
Approvals	`/approvals`	g a	Approval workflow
Tasks	`/tasks`	g k	Task tracking
Blessing	`/blessing`	g b	Canonical fact approval
Mem Reviews	`/memory-approvals`	g v	Memory review queue
Health	`/health`	g h	System health
Traces	`/traces`	g r	Execution traces
Graph	`/graph`	g g	3D knowledge graph visualization

The Graph page renders memory relationships and tag clusters in 3D space with two views: **Provenance view** (memory attribution, supersession chains, contradiction flags) and **Tag view** (semantic clustering with typed nodes). The dashboard server (`index.js`, 4,538 lines) handles REST API endpoints for federation control, memory search, agent execution, and approval management.

11 Cognitive Architecture

Lead Claude operates with 46 metacognitive modules organized into seven categories, providing self-awareness and adaptive behavior:

Category	Count	Key Modules
Instance Management	4	Instance detection, registry, heartbeat, escalation
Coordination	2	Federation coordinator, primary coordinator
Self-Awareness	4	Self-model, observability, metacognition, synthesizer
Memory Management	7	Consistency, narrative memory, collaboration patterns, preference learning, growth tracking, scar registry, theory of mind
Epistemic	5	Boundary guard, confidence tracker, context reconstruction, error taxonomy, counterfactual logging
Psychological	3	Emotional momentum, continuity tension, surprise quantification
Infrastructure	10	Feature flags, forgetting mechanisms, graceful degradation, drift monitor, trust surface controller, and more

Six modules persist state to disk for cross-session continuity: preference learning, growth tracking, counterfactual logging, emotional momentum, surprise quantification, and theory of mind. The hooks system manages session lifecycle, loading cognitive modules at startup and injecting context including worker detection, memory retrieval, and federation state.

12 Quick Reference / Cheat Sheet

Launch

```
Launch via Zellij federation layout
```

Delegate

```
# Research (Gemini)
delegate-research "topic" -report-to claude
# Code (Codex)
delegate-code "task" -full-auto
# Analysis (Qwen)
delegate-analysis "question"
# Quick (Qwen, skip /think)
delegate-analysis "question" -quick
# Multi-perspective
swarm_solve(problem="...",
to_agents=["gemini","codex","qwen"])
```

Inbox & Approvals

```
list_pending_notes()
list_pending_approvals()
auto_process_approvals()
respond_to_approval(note_id, decision,
reason)
```

Agent Interaction (Zellij)

```
execute_in_agent(agent, prompt)
read_agent_pane(agent, lines)
# Approve agents via their CLI prompts
```

Memory

```
store_memory(content, tags)
recall_memory(query)
cross_query(query)
check_duplicates_semantic(content)
register_canonical_fact(fact)
```

Workers

```
execute_in_agent(agent="codex-1", ...)
spawn_workers_sequential([...])
release_worker(worker_id)
```

Ports (Do Not Change)

```
Each agent runs on a dedicated port.
Ports are configured per-agent
and should not be changed
during operation.
```

13 Troubleshooting

13.1 Agent Not Responding

Symptom	Cause	Fix
No output in Zellij pane	Agent CLI not in PATH	Verify: `which gemini`, `which codex`, `which qwen`
Agent starts, no MCP tools	MCP config not loaded	Ensure MCP configuration is in project root, restart session
Approval prompt blocking	Waiting for approval input	Send approval sequence via CLI prompt
Agent idle after task	Finished but didn't report	Check with `read_agent_pane()`, re-prompt if needed
`execute_in_agent` no effect	Wrong pane targeted	Verify pane with screen dump, ensure Enter is sent

13.2 Notes Piling Up

Symptom	Cause	Fix
Hundreds of unread notes	Not acknowledged after processing	Run `acknowledge_note()` for each
Inbox flooded	Swarm generated many notes	`purge_inbox()` or `cleanup_federation_notes()`
Duplicate notes	Agent sent same note twice	`check_duplicates()` before writing; purge dupes
Notes from dead sessions	Previous sessions left notes	`cleanup_federation_notes(older_than="24h")`

13.3 Worker Issues

Symptom	Cause	Fix
"Worker limit reached"	5 concurrent workers active	Wait for TTL auto-release, or `release_worker()`
"Rate limit exceeded"	>10 spawns in 10-min window	Wait for rate window to reset
Worker crashed silently	Worker hit error and exited	Check `read_agent_pane()` for error output
Worker results missing	Worker expired before reporting	Increase task specificity; ensure `write_note` in prompt

13.4 Approval Issues

Symptom	Cause	Fix
Approval stuck in queue	Lead hasn't processed it	`list_pending_approvals()` then `respond_to_approval()`
Auto-approve not working	Not called	Call periodically, or add to session startup
Agent keeps re-requesting	Wrong action approved	Check specific request; approve correct one

13.5 Memory Issues

Symptom	Cause	Fix
Memory service unreachable	Port not open / service crashed	Check memory service health endpoint
Cross-query returns nothing	Embedding gateway down	Check embedding gateway health endpoint
Duplicate memories	No dedup check before store	Use `check_duplicates_semantic()`
Store growing too large	No cleanup or consolidation	Use dream-mode consolidation or manual cleanup

13.6 Common Error Messages

Error	Meaning	Resolution
`ECONNREFUSED :<port>`	Memory service not running	Start memory-service server
`ECONNREFUSED :<port>`	Embedding gateway not running	Start embedding-gateway server
`notes.db locked`	Concurrent write to notes DB	Wait and retry
`worker lease expired`	Worker exceeded 10-min TTL	Re-spawn if needed
`swarm_id not found`	Swarm timed out	Re-launch with `swarm_solve()`

End of DI Federation Operations Guide.

Four labs. One federation. Built at 12:41 AM.