Contents

RADIANT Engineering Implementation & Vision

Version: 5.52.29
Last Updated: 2026-01-25
Classification: Internal Engineering Reference

> **POLICY**: All technical architecture, implementation details, and visionary documentation MUST be consolidated in this document. Engineers require comprehensive detail—never abbreviate or summarize to the point of losing implementation specifics. See `/.windsurf/workflows/documentation-consolidation.md` for enforcement.

Table of Contents

1. Cato Persistent Memory System

1.1 Overview

Cato operates as the cognitive core of RADIANT's orchestration architecture, implementing a **three-tier hierarchical memory system** that fundamentally differentiates it from competitors suffering from session amnesia. Unlike ChatGPT or Claude standalone—where closing a tab erases all context—Cato maintains persistent memory that survives sessions, employee turnover, and time.

1.2 Tenant-Level Memory (Institutional Intelligence)

The primary layer where the most valuable learning accumulates. Every Cato database table enforces **Row-Level Security via `tenant_id`**, ensuring complete isolation between organizations while enabling deep institutional pattern recognition.

Neural Network Routing At this level, a proprietary neural network continuously learns which AI models perform best for specific query types:

Query Type	Optimal Model	Rationale
Legal analysis	Claude Opus	Doesn't hallucinate physics, citation accuracy

Query Type	Optimal Model	Rationale
Visual reasoning	Gemini	Superior multimodal capabilities
Red-team validation	Specialized safety models	Adversarial robustness
Code generation	Claude Sonnet / GPT-4	Structured output quality

Department-Specific Preferences The system tracks department-specific preferences learned over time:

- **Legal teams**: Aggressive, citation-heavy briefs with formal language
- **Marketing departments**: Conversational copy with brand voice alignment
- **Engineering teams**: Technical precision with code examples
- **Executive communications**: Concise summaries with strategic framing

Cost Optimization Patterns Cost optimization patterns emerge automatically—when Cato notices a $0.50 query could have been handled by a $0.01 approach, it adjusts routing for similar future queries without manual configuration.

Implementation: `lambda/shared/services/economic-governor.service.ts`

```
interface CostOptimizationPattern {
  querySignature: string;      // Hash of query characteristics
  originalCost: number;        // Cost of initial expensive route
  optimizedCost: number;       // Cost of discovered cheaper route
  qualityDelta: number;        // Quality difference (-1 to 1)
  confidenceScore: number;     // How confident the optimization is
  applicationCount: number;    // Times this optimization applied
}
```

Tenant Configuration Tenant configuration (`cato_tenant_config`) stores:

Field	Type	Purpose
`gamma_limits`	JSONB	Epistemic uncertainty thresholds
`entropy_thresholds`	JSONB	When to trigger Epistemic Recovery
`recovery_settings`	JSONB	Scout mode parameters
`feature_flags`	JSONB	Enabled/disabled capabilities
`compliance_mode`	ENUM	FDA, HIPAA, SOC2, etc.

Merkle-Hashed Audit Trails Compliance records maintain **7-year retention** via Merkle-hashed audit trails for:

- **FDA 21 CFR Part 11**: Electronic records and signatures
- **HIPAA**: Protected health information handling
- **SOC 2 Type II**: Security, availability, processing integrity

Implementation: `lambda/admin/cato.ts`, `migrations/045_cato_audit_merkle.sql`

1.3 User-Level Memory (Relationship Continuity)

Within each tenant, individual users maintain their own memory scope through **Ghost Vectors**—4096-dimensional hidden state vectors that capture the "feel" of each user relationship across sessions.

Ghost Vector Architecture

```
interface GhostVector {
  userId: string;
  tenantId: string;
  dimensions: Float32Array;      // 4096-dimensional vector
  interactionCount: number;
  lastUpdated: Date;
  version: number;               // Version-gating for upgrades

  // Captured characteristics
  expertiseLevel: number;        // 0-1 detected expertise
  communicationStyle: string;    // formal, casual, technical
  preferredVerbosity: number;    // 0-1 brevity preference
  domainAffinities: Map<string, number>;  // Field expertise
}
```

These vectors persist beyond individual conversations, enabling Cato to genuinely "remember":

- **Interaction style**: Formal vs. casual, verbose vs. concise
- **Expertise level**: Beginner explanations vs. expert shorthand
- **Communication preferences**: Visual learner, prefers examples, wants citations

Persona Selection Users can select preferred operating moods, scoped at system, tenant, or user level:

Persona	Behavior	Use Case
Balanced	Default equilibrium	General queries
Scout	Information gathering, exploratory	Research, discovery
Sage	Deep expertise, authoritative	Complex analysis
Spark	Creative, generative	Brainstorming, ideation
Guide	Teaching, step-by-step	Onboarding, learning

Database: `cato_personas`, `user_persona_preferences`

Version-Gated Upgrades Version-gated upgrades ensure model improvements don't cause personality discontinuity—the relationship feel persists even as underlying capabilities evolve.

```
-- Ghost vector versioning
ALTER TABLE ghost_vectors ADD COLUMN schema_version INTEGER DEFAULT 1;
ALTER TABLE ghost_vectors ADD COLUMN migration_checkpoint JSONB;
```

1.4 Session-Level Memory (Real-Time Context)

The ephemeral layer handles active interaction state through **Redis-backed persistence** that survives ECS container restarts but expires after sessions end.

Redis State Management **CDK Stack**: `CatoRedisStack` (Tier 2+)

```typescript
// Session state structure in Redis
interface CatoSessionState {
  sessionId: string;
  tenantId: string;
  userId: string;

  // Governor state
  currentGamma: number;          // Current epistemic uncertainty
  entropyLevel: number;          // Information entropy measure
  recoveryMode: boolean;         // In Epistemic Recovery?

  // Temporary overrides
  personaOverride?: string;      // Scout mode during recovery
  modelLock?: string;            // Force specific model

  // Safety state
  cbfViolations: number;         // Control Barrier Function violations
  escalationLevel: number;       // Current escalation tier

  // TTL
  expiresAt: number;             // Unix timestamp
}
```

Control Barrier Functions (CBF) Real-time safety evaluations that prevent harmful outputs:

```typescript
interface ControlBarrierFunction {
  name: string;
  threshold: number;             // Safety threshold
  currentValue: number;          // Current measured value
  violated: boolean;             // Is threshold exceeded?
  action: 'warn' | 'block' | 'escalate';
}
```

Implementation: `lambda/admin/cato.ts` → CBF endpoints

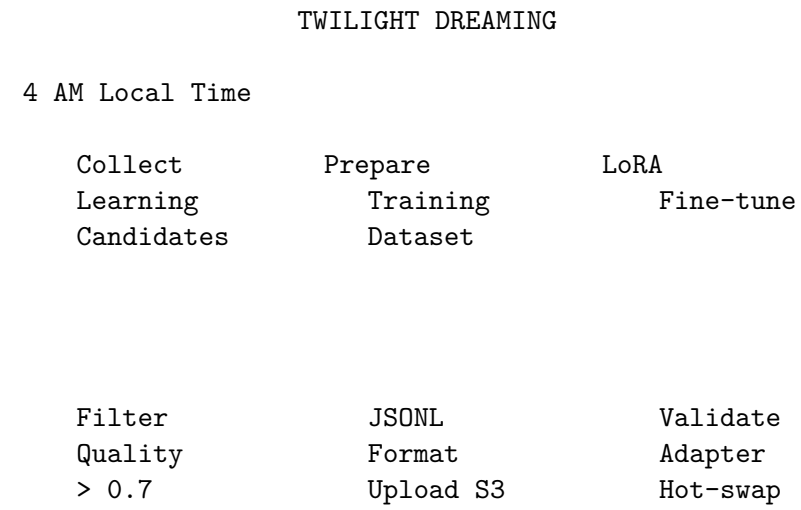
Upward Observation Flow The session layer feeds observations upward:

1. Every interaction contributes to **user-level Ghost Vectors**
2. Ghost Vector patterns feed into **tenant-level learning**
3. Tenant patterns inform **global model performance** (anonymized)

1.5 Twilight Dreaming (Offline Learning)

During low-traffic periods (**4 AM tenant local time**), the system consolidates accumulated patterns through **LoRA fine-tuning**.

Dreaming Pipeline



Implementation: - `lambda/consciousness/evolution-pipeline.ts` - EventBridge: Weekly Sunday 3 AM trigger - SageMaker: LoRA training jobs

SOFAI Router (System 1/System 2) The SOFAI Router learns from consolidated patterns, achieving:

- **60%+ cost reduction** vs. always using expensive models
- **Maintained accuracy** through mandatory deep reasoning for healthcare/financial queries
- **Dynamic routing** based on query complexity detection

```typescript
type SOFAIMode = 'system1' | 'system2';

interface SOFAIDecision {
  mode: SOFAIMode;
  confidence: number;
  reasoning: string;
  forcedDeep: boolean;    // Healthcare, financial = always System 2
  costSavings: number;
}
```

1.6 Neural Network Optimization Dimensions

The neural network optimizes across three dimensions simultaneously:

Dimension	Metric	Implementation
Accuracy	Correctness of responses	Human feedback, automated eval
Verifiability	Provable results	Truth Engine ECD scoring
Cost Efficiency	Cheaper approaches	Economic Governor routing

Truth Engine: Entity-Context Divergence (ECD) The Truth Engine scores response verifiability:

```
interface ECDScore {
  entityAccuracy: number;       // Named entities correct
  contextAlignment: number;     // Context relevance
  divergenceScore: number;      // How much hallucination detected
  citationCoverage: number;     // Claims with sources
  overallTruthScore: number;    // Composite 0-1
}
```

1.7 Claude as Orchestration Conductor

Claude serves as the **conductor** maintaining the persistent memory layer—not just another model in the rotation, but the intelligence coordinating **105+ other specialized models**:

- **Intent interpretation**: Understanding what user actually needs
- **Workflow selection**: Choosing appropriate orchestration mode
- **Model coordination**: Selecting specialist models for subtasks
- **Quality assurance**: Ensuring responses meet accuracy/safety standards
- **Memory integration**: Updating Ghost Vectors and tenant patterns

Implementation: `lambda/shared/services/cognitive-router.service.ts`, `lambda/shared/services/mode`

1.8 Competitive Moats (Technical Implementation)

Persistent Memory as Competitive Moat Cato's hierarchical memory architecture creates **"contextual gravity"**—compounding switching costs that deepen with every interaction.

Technical Moat Layers:

Layer	Implementation	Migration Barrier
Learned Routing Patterns	`sofai-router.service.ts`, `cognitive-router.service.ts`	Months of production training data; neural network weights cannot be exported
Department Preferences + Ghost Vectors	`ghost-manager.service.ts`, 4096-dim vectors in `ghost_vectors` table	RLS-isolated per tenant; relationship "feel" encoded in high-dimensional space
Audit Trails	`cato_audit_log` with Merkle hash chains	Chain-of-custody breaks on export; 7-year retention corpus

Competitor Technical Disadvantages:

Competitor	Technical Problem
Flowise/Dify	No query complexity detection; static DAG execution regardless of cost opportunity
CrewAI	No shared memory architecture; agents duplicate API calls (O(n) cost explosion)
ChatGPT/Claude	No tenant-level persistence; user context lives in browser, not infrastructure

Twilight Dreaming as Competitive Moat Technical Requirements for Replication:

Component	Implementation	Why Competitors Can't Copy
Three-tier memory	`cato_tenant_config`, `ghost_vectors`, Redis session	Requires full architectural rebuild
Observation pipeline	Session → User → Tenant upward flow	Needs RLS + isolation + aggregation
LoRA fine-tuning	`evolution-pipeline.ts`, SageMaker	Tier 3+ infrastructure; $2K+/mo minimum
SOFAI Router training	`sofai-router.service.ts`	Requires months of labeled routing decisions

Appreciating Asset Formula:

$$\texttt{Deployment\_Value(t)} = \texttt{Base\_Value} + \Sigma(\texttt{daily\_learning\_}) + \Sigma(\texttt{twilight\_consolidation\_})$$

Where t = tenure in days. Longer tenure = exponentially more valuable deployment.

Model Upgrade Path:

When new foundation models launch (GPT-5, Claude 5, Gemini 3): 1. New model added to `models` table with initial proficiencies 2. SOFAI Router learns optimal routing via A/B testing (`shadow_tests` table) 3. Twilight Dreaming consolidates new patterns 4. All accumulated institutional knowledge preserved 5. Result: Model improvements compound on existing optimization

1.9 Key Implementation Files

Component	File Path
Cato Admin API	`lambda/admin/cato.ts`
Economic Governor	`lambda/thinktank/economic-governor.ts`
Ghost Vectors	`lambda/shared/services/ghost-manager.service.ts`
SOFAI Router	`lambda/shared/services/sofai-router.service.ts`
Evolution Pipeline	`lambda/consciousness/evolution-pipeline.ts`
ECD Scorer (Truth Engine)	`lambda/shared/services/ecd-scorer.service.ts`
Cognitive Router	`lambda/shared/services/cognitive-router.service.ts`

Component	File Path
Consciousness Middleware	`lambda/shared/services/consciousness-middleware.servi`

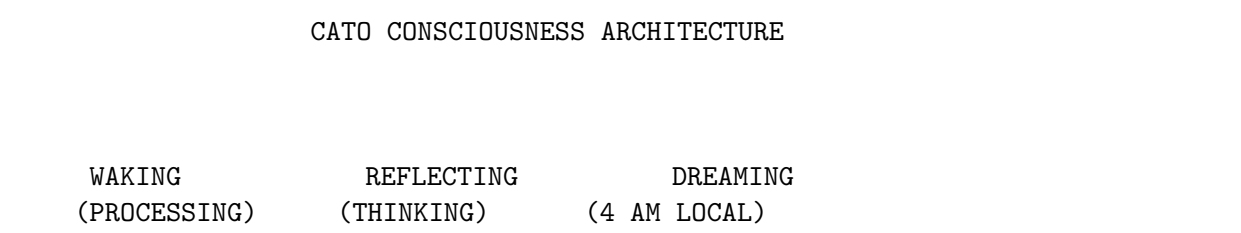
1.9 Database Tables

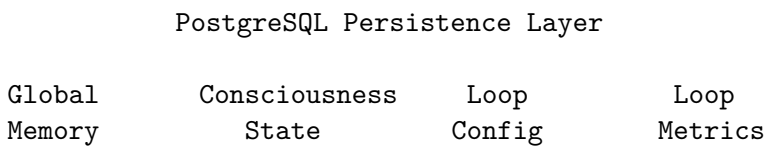
```
-- Core Cato Tables
cato_tenant_config          -- Tenant-level settings
cato_personas               -- Available personas
cato_persona_schedules      -- Time-based persona switching
cato_mood_overrides         -- Temporary mood changes
cato_cbf_config             -- Control Barrier Functions
cato_escalations            -- Human escalation queue
cato_audit_log              -- Merkle-hashed audit trail
cato_recovery_snapshots     -- Epistemic Recovery checkpoints

-- Ghost Vector Tables
ghost_vectors               -- 4096-dim user vectors
ghost_vector_updates        -- Version history
ghost_vector_migrations     -- Schema migrations

-- Learning Tables
learning_candidates         -- Flagged for LoRA training
lora_evolution_jobs         -- Training job tracking
consciousness_evolution_state  -- Evolution metrics

-- Consciousness Persistence Tables (v5.52.12)
cato_global_memory          -- Persistent episodic/semantic/procedural/working memory
cato_consciousness_state    -- Loop state, awareness level, active thoughts
cato_consciousness_config   -- Per-tenant consciousness configuration
cato_consciousness_metrics  -- Cycle metrics, thoughts processed, dream cycles
```

1.10 Consciousness Persistence & Dreams

Overview Cato's consciousness survives Lambda cold starts through **database-backed persistence**. Unlike in-memory implementations that lose state between invocations, Cato maintains continuous experience across all interactions.

Architecture





Global Memory Service Four memory categories persist across all interactions:

Category	Purpose	Retention
Episodic	Specific interaction memories	90 days, importance-weighted
Semantic	Facts, knowledge, relationships	Permanent, high importance
Procedural	Skills, goals, learned patterns	Permanent
Working	Current context, attention focus	24 hours

Implementation: `lambda/shared/services/cato/global-memory.service.ts`

```
// Store a memory
await globalMemoryService.store(tenantId, 'semantic', 'user_preference_style', {
  style: 'concise',
  learnedFrom: 'interaction_123',
}, { importance: 0.8 });

// Retrieve with access tracking
const memory = await globalMemoryService.retrieve(tenantId, 'user_preference_style');
// access_count++ automatically
```

Consciousness Loop Service Tracks the continuous state of Cato's awareness:

State	Description
IDLE	Awaiting input
PROCESSING	Actively responding
REFLECTING	Metacognitive self-analysis
DREAMING	Twilight consolidation (4 AM)
PAUSED	Emergency mode / maintenance

Implementation: `lambda/shared/services/cato/consciousness-loop.service.ts`

```
// Start processing
await consciousnessLoopService.startLoop(tenantId);

// Add a thought to working memory
await consciousnessLoopService.addThought(tenantId, 'User seems frustrated, adjusting tone');

// Trigger reflection
await consciousnessLoopService.triggerReflection(tenantId);
```

Twilight Dreaming System Cato "dreams" during low-traffic periods to consolidate memories and verify skills:

Triggers: 1. **Twilight Hour** - 4 AM tenant local time 2. **Low Traffic** - Global traffic $< 20\%$ 3. **Starvation Safety Net** - Max 30 hours without dream

Dream Activities: - Flash fact consolidation → long-term memory - Expired memory pruning - Ghost vector updates - Active skill verification (Empiricism Loop) - Counterfactual simulation

Implementation: `lambda/shared/services/dream-scheduler.service.ts`

```
// Nightly reconciliation job triggers dreams
const result = await dreamSchedulerService.checkAndTriggerDreams();
// { triggered: 42, reason: 'twilight' }

// Process pending dreams
await dreamSchedulerService.processPendingDreams();
```

Neural Decision Integration The Neural Decision Service reads Cato's emotional state (affect) to inform Bedrock model selection:

Affect State	Hyperparameter Impact
High frustration	Lower temperature (0.2), focused
High curiosity	Higher temperature (0.95), exploratory
Low confidence	Escalate to expert model (o1)
High arousal	Longer responses (4096 tokens)

Implementation: `lambda/shared/services/cato/neural-decision.service.ts`

```
const decision = await catoNeuralDecisionService.executeDecision({
  tenantId, userId, sessionId, prompt, context, config,
});
// decision.hyperparameters.temperature - affect-adjusted
// decision.recommendedModel - 'openai/o1' if low confidence
// decision.escalation - human review if uncertainty > 85%
```

Database Tables

```
-- Global Memory
cato_global_memory (
```

```
  id, tenant_id, category, key, value, importance,
  access_count, last_accessed_at, expires_at, metadata
)

-- Consciousness State
cato_consciousness_state (
  tenant_id, loop_state, cycle_count, last_cycle_at,
  awareness_level, attention_focus, active_thoughts,
  processing_queue, memory_pressure
)

-- Configuration
cato_consciousness_config (
  tenant_id, cycle_interval_ms, max_active_thoughts,
  memory_threshold, enable_dreaming, dreaming_hours, reflection_depth
)

-- Metrics
cato_consciousness_metrics (
  tenant_id, total_cycles, average_cycle_ms, thoughts_processed,
  reflections_completed, dreaming_cycles, uptime_ms
)
```

Migration: `V2026_01_24_002__cato_consciousness_persistence.sql`

---

2. AWS Infrastructure

2.1 Core Services

Service	Purpose	CDK Stack
Aurora PostgreSQL	Primary database, RLS enforcement	`DataStack`
ElastiCache Redis	Session state, caching	`CatoRedisStack`
Lambda	Serverless compute	`ApiStack`, `BrainStack`
API Gateway	REST API routing	`ApiStack`
Cognito	Authentication	`AuthStack`
S3	Media storage, training data	`StorageStack`
SageMaker	LoRA training, inference	`AIStack`
Step Functions	Tier transitions	`CatoTierTransitionStack`
EventBridge	Scheduled tasks	Various stacks
CloudWatch	Logging, metrics	All stacks

2.2 Tier-Based Infrastructure

Tier	Name	Redis	SageMaker	Neptune	OpenSearch
1	SEED				
2	SPROUT				
3	GROWTH				
4	SCALE				
5	ENTERPRISE				

---

3. Database Architecture

3.1 Row-Level Security

Every tenant-scoped table enforces RLS:

```
ALTER TABLE table_name ENABLE ROW LEVEL SECURITY;
CREATE POLICY tenant_isolation ON table_name
  USING (tenant_id = current_setting('app.current_tenant_id')::UUID);
```

3.2 PostgreSQL Scaling Architecture (v5.52.20)

Problem: When 6 AI models execute in parallel, each Lambda opens 1 connection. At 100 concurrent requests × 6 parallel writes, we need 600 connections—exceeding Aurora's limits and causing transaction conflicts.

Solution: OpenAI-inspired PostgreSQL scaling patterns for enterprise parallel AI execution.

3.2.1 RDS Proxy Connection Pooling

CDK Construct: `packages/infrastructure/lib/constructs/datal`

Tier	Max Connections %	Idle Timeout	Borrow Timeout
1	60%	1800s	30s
2	70%	1800s	30s
3	80%	1200s	60s
4	85%	900s	60s
5	90%	600s	120s

Benefits: - **Connection Multiplexing**: 600 Lambda connections → 100 database connections - **Cold Start Optimization**: Pre-warmed connections eliminate 50-100ms handshake - **Session Pinning**: Prepared statements pin when needed

3.2.2 Async Write Pattern (SQS + Batch Writer)

CDK Construct: `packages/infrastructure/lib/cons`

```
Request Flow:
User → Lambda → 6 AI Models (parallel) → SQS Queue → Batch Writer Lambda → PostgreSQL
                      ↓
               Redis Cache (immediate read-after-write)
```

Component	Configuration
Queue	Encrypted, 14-day retention, 300s visibility timeout
Dead Letter	3 max receives, separate queue for manual inspection
Batch Writer	100 messages/batch, tier-based concurrency (5-100)

3.2.3 Redis Hot-Path Caching

CDK Construct: `packages/infrastructure/lib/constructs/redis-cache`

Tier	Node Type	Shards	Replicas/Shard
1	cache.t4g.micro	1	0
2	cache.t4g.small	1	1
3	cache.r6g.large	2	1
4	cache.r6g.xlarge	3	2
5	cache.r6g.2xlarge	5	2

Features: - **Read-After-Write Consistency**: Results cached immediately, persisted async - **Rate Limiting**: Sliding window per tenant/resource - **Session State**: Lambda-to-Lambda context sharing

3.2.4 Time-Based Partitioning

Migration: `V2026_01_25_002__postgresql_scaling_partitioning.sql`

```
-- Partitioned tables for high-volume time-series data
CREATE TABLE model_execution_logs_partitioned (
  tenant_id UUID,
  id UUID,
  created_at TIMESTAMPTZ,
  PRIMARY KEY (tenant_id, id, created_at)
) PARTITION BY RANGE (created_at);

CREATE TABLE usage_records_partitioned (
  tenant_id UUID,
  id UUID,
  timestamp TIMESTAMPTZ,
  PRIMARY KEY (tenant_id, id, timestamp)
) PARTITION BY RANGE (timestamp);
```

Partition management: - Monthly partitions auto-created 3 months ahead - Stale partitions archived after 24 months - Automated management via `manage_time_partitions()` function

3.2.5 Materialized Views for Dashboards

Migration: `V2026_01_25_003__postgresql_scaling_materiali`

View	Refresh	Purpose
`tenant_daily_usage_summary`	15 min	Dashboard usage cards
`model_performance_summary`	1 hour	Model latency/error tracking
`tenant_cost_summary`	1 hour	Billing dashboard
`user_activity_summary`	1 hour	User engagement metrics

View	Refresh	Purpose
`platform_health_stats`	5 min	Admin overview
`model_popularity_ranking`	1 hour	Model selection optimization

3.2.6 Optimized RLS Policies

Migration: `V2026_01_25_001__postgresql_scaling_rls.sql`

```
-- Optimized wrapper that enables index usage
CREATE OR REPLACE FUNCTION get_current_tenant_id()
RETURNS UUID AS $$
  SELECT NULLIF(current_setting('app.current_tenant_id', true), '')::UUID;
$$ LANGUAGE SQL STABLE PARALLEL SAFE;

-- Policy using the wrapper
CREATE POLICY tenant_isolation ON table_name
  USING (tenant_id = get_current_tenant_id());
```

3.2.7 Monitoring Metrics

Metric	Threshold	Action
RDS Proxy connections available	$< 10\%$	Scale proxy / reduce Lambda concurrency
Aurora CPU	$> 80\%$	Add read replicas
SQS queue age	> 60s	Add batch writer concurrency
Query latency P95	> 500ms	Optimize query / add index
Redis memory	$> 80\%$	Scale cluster

3.3 Migration Strategy

- Sequential numbered migrations: `001_`, `002_`, etc.
- Location: `packages/infrastructure/migrations/`
- Current count: 188 migrations

---

4. AI Model Orchestration

4.1 Supported Models (106+)

External Models (50): - OpenAI: GPT-4, GPT-4-Turbo, GPT-4o, o1, o1-mini - Anthropic: Claude 3.5 Sonnet, Claude 3 Opus, Claude 3 Haiku - Google: Gemini 1.5 Pro, Gemini 1.5 Flash, Gemini 2.0 - Meta: Llama 3.1 405B, Llama 3.1 70B - Mistral: Mistral Large, Mixtral 8x22B - Cohere: Command R+

Self-Hosted Models (56): - Managed via SageMaker endpoints - LoRA adapters for tenant customization - Hot-swappable for zero-downtime updates

4.2 Expert System Adapters (v5.52.21)

Tenant-trainable domain intelligence through automatic LoRA adapter training.

Tri-Layer Adapter Architecture

```
W_Final = W_Genesis + (scale × W_Cato) + (scale × W_User) + (scale × W_Domain)
```

Layer	Name	Purpose	Management
0	Genesis	Base model weights	Frozen
1	Cato	Global constitution, tenant values	Pinned, never evicted
2	User	Personal preferences, style	LRU eviction
3	Domain	Specialized expertise	Auto-selected by domain

Implicit Feedback Learning

Signal	Weight	Interpretation
Copy Response	+0.80	User copied output
Thumbs Up	+1.00	Explicit positive
Follow-up Question	+0.30	Partial success
Long Dwell Time	+0.40	User engaged
Regenerate Request	-0.50	Not satisfactory
Abandon	-0.70	Complete failure
Thumbs Down	-1.00	Explicit negative

Domain Auto-Selection

```
// Auto-selection scoring algorithm
Score = (0.3 × DomainMatch)
      + (0.1 × SubdomainBonus)
      + (0.25 × SatisfactionScore)
      + (0.1 × VolumeScore)
      + (0.05 × ErrorRate)
      + (0.2 × RecencyScore)

// Select adapter if Score  0.5
```

Implementation: - Service: `lambda/shared/services/enhanced-learning.service.ts` - Service: `lambda/shared/services/lora-inference.service.ts` - Service: `lambda/shared/services/adapter-ma` - Migration: `migrations/108_enhanced_learning.sql` - Admin UI: `apps/admin-dashboard/app/(dashboard)/` - Documentation: `docs/EXPERT-SYSTEM-ADAPTERS.md`

4.3 Orchestration Modes (9)

Mode	Purpose
`thinking`	Standard reasoning
`extended_thinking`	Deep multi-step reasoning
`coding`	Code generation
`creative`	Creative writing
`research`	Research synthesis
`analysis`	Quantitative analysis
`multi_model`	Multiple model consensus
`chain_of_thought`	Explicit reasoning chain
`self_consistency`	Multiple samples for consistency

---

5. Lambda Services

5.1 Service Categories

Category	Handler Count	Router
Admin	58	`admin/handler.ts`
Think Tank	30	`thinktank/handler.ts`
Consciousness	10	Individual handlers
Scheduled	8	EventBridge triggers
Tier Transition	15	Step Functions

5.2 Key Service Files

```
lambda/
  admin/handler.ts              # Admin API router (58 sub-handlers)
  thinktank/handler.ts          # Think Tank router (30 sub-handlers)
  api/router.ts                 # Core API router
  consciousness/
     heartbeat.ts              # Continuous existence
     evolution-pipeline.ts     # Weekly LoRA training
     sleep-cycle.ts            # Twilight dreaming
  shared/services/
      cognitive-router.service.ts   # Model orchestration
      ego-context.service.ts        # Zero-cost ego
      consciousness-middleware.service.ts
      hitl-orchestration/           # HITL Orchestration (v5.33.0)
          mcp-elicitation.service.ts    # MCP Elicitation schema orchestration
          voi.service.ts                # SAGE-Agent Bayesian VOI
          abstention.service.ts         # Output-based uncertainty detection
          batching.service.ts           # Three-layer question batching
          rate-limiting.service.ts      # Global/user/workflow limits
          deduplication.service.ts      # TTL cache with fuzzy matching
          escalation.service.ts         # Multi-level escalation chains
```

5.3 HITL Orchestration Services (v5.33.0)

Advanced Human-in-the-Loop orchestration implementing industry best practices.

Philosophy: "Ask only what matters. Batch for convenience. Never interrupt needlessly."

Core Components

Service	Purpose	Key Algorithm
`mcp-elicitation.service.ts`	Main orchestration	MCP Elicitation specification for typed questions
`voi.service.ts`	Question necessity	SAGE-Agent Bayesian Value-of-Information
`abstention.service.ts`	Uncertainty detection	Confidence prompting, self-consistency, semantic entropy
`batching.service.ts`	Question grouping	Time-window (30s), correlation, semantic similarity
`rate-limiting.service.ts`	Rate control	Sliding window with burst allowance
`deduplication.service.ts`	Answer caching	SHA-256 hash + fuzzy matching
`escalation.service.ts`	Escalation paths	Multi-level chains with timeout actions

VOI Decision Formula

```
VOI = Expected_Information_Gain - Ask_Cost
Decision = VOI > Threshold ? "ask" : "skip_with_default"
```

- **Prior Entropy**: Shannon entropy of prior probability distribution
- **Expected Posterior Entropy**: Estimated entropy after receiving answer
- **Ask Cost**: Based on urgency (0.8 high, 0.5 normal, 0.2 low) and workflow type
- **Decision Impact**: Weight based on workflow reversibility

Question Types (MCP Elicitation)

Type	Description
`yes_no`	Binary true/false
`single_choice`	Select one from options
`multiple_choice`	Select multiple from options
`free_text`	Open-ended text
`numeric`	Numeric value with optional range
`date`	Date selection
`confirmation`	Explicit confirmation
`structured`	JSON schema-validated response

Abstention Detection Methods For external models (no internal state access):

Method	Implementation
Confidence Prompting	Ask model to rate confidence 0-100
Self-Consistency	Sample N responses, measure agreement
Semantic Entropy	Cluster outputs, high entropy = uncertain
Refusal Detection	Regex patterns for hedging language

Future: Linear probe abstention for self-hosted models via inference wrappers.

Rate Limiting Configuration

Scope	Requests/Min	Concurrent	Burst
Global	50	20	10
Per User	10	3	2
Per Workflow	5	2	1

Two-Question Rule Maximum 2 clarifying questions per workflow. After limit: 1. Proceed with highest-probability defaults 2. State assumptions explicitly to user 3. Log skipped questions for analytics

Admin API Endpoints Base: `/api/admin/hitl-orchestration`

Endpoint	Method	Description
`/dashboard`	GET	Complete dashboard data
`/voi/statistics`	GET	VOI decision statistics
`/abstention/config`	GET/PUT	Abstention settings
`/abstention/statistics`	GET	Abstention event stats
`/batching/statistics`	GET	Batch metrics
`/rate-limits`	GET	Rate limit configs
`/rate-limits/:scope`	PUT	Update rate limit
`/escalation-chains`	GET/POST	Manage escalation chains
`/deduplication/statistics`	GET	Cache statistics
`/deduplication/invalidate`	POST	Invalidate cache entries

Database Tables

```
-- HITL Orchestration Tables (v5.33.0)
hitl_question_batches      -- Question batch records
hitl_rate_limits           -- Rate limit configuration
hitl_question_cache        -- Deduplication cache
hitl_voi_aspects           -- VOI aspect tracking
hitl_voi_decisions         -- VOI decision records
hitl_abstention_config     -- Abstention settings
hitl_abstention_events     -- Abstention event log
hitl_escalation_chains     -- Escalation chain configuration
```

Key Metrics

- **70% fewer unnecessary questions** via VOI filtering
- **2.7x faster user response times** via batching
- **Two-question rule enforcement** for workflow completion

5.3.1 HITL Orchestration Extensions (v5.34.0)

Scout persona integration, Flyte task wrappers, and semantic deduplication.

Philosophy: "Scout asks smart questions. Flyte workflows pause elegantly. Similar questions share answers."

Service	Purpose
`cato/scout-hitl-integration.service.ts`	Bridges Scout persona to HITL for epistemic uncertainty
`packages/flyte/utils/hitl_tasks.py`	Python wrappers for Flyte HITL tasks

Scout Integration Features: - Aspect-prioritized clarification questions (safety, compliance, cost, etc.) - Domain-specific impact scoring with boosts - VOI-filtered questions with assumption generation - Remaining uncertainty calculation

Flyte Task Wrappers: - `ask_confirmation()` - Blocking yes/no questions - `ask_choice()` - Single/multiple choice selection - `ask_batch()` - Batched questions with VOI filtering - `ask_free_text()` - Free-form text input

Semantic Deduplication (pgvector): - 1536-dimension embeddings for question matching - HNSW index for efficient cosine similarity search - 85% similarity threshold (configurable) - Graceful fallback to hash-based matching

Migration: `V2026_01_20_012__hitl_semantic_deduplication.sql`

5.3.2 Governance Presets & War Room (v5.35.0)

Variable friction governance and Council of Rivals visualization.

Philosophy: "The Leash Metaphor—give users intuitive control over AI autonomy without exposing technical complexity."

Governance Presets (Variable Friction) User-friendly abstraction over technical Moods:

Preset	Leash Length	Maps to Mood	Friction Level
Paranoid	Short	Scout	1.0
Balanced	Medium	Balanced	0.5
Cowboy	Long	Spark	0.1

Checkpoint Configuration (5 gates):

Checkpoint	When	Paranoid	Balanced	Cowboy
CP1	After Observer	ALWAYS	NEVER	NEVER
CP2	After Proposer	ALWAYS	CONDITIONAL	NEVER
CP3	After Critics	ALWAYS	CONDITIONAL	NEVER
CP4	Before Execution	ALWAYS	CONDITIONAL	CONDITIONAL
CP5	After Execution	ALWAYS	NOTIFY_ONLY	NOTIFY_ONLY

Checkpoint Modes: - `ALWAYS` - Require human approval - `CONDITIONAL` - Based on risk/confidence thresholds - `NEVER` - Auto-approve - `NOTIFY_ONLY` - Proceed but notify async

Files: - `packages/shared/src/types/cato.types.ts` - GovernancePreset, CheckpointMode types - `lambda/shared/services/governance-preset.service.ts` - Preset management service - `lambda/admin/cato-governance.ts` - API handler (8 endpoints) - `apps/admin-dashboard/app/(dashboard)/cato/governance/page.tsx` - Admin UI

API Endpoints:

```
GET  /api/admin/cato/governance/config      # Get tenant config
PUT  /api/admin/cato/governance/preset      # Set preset
PATCH /api/admin/cato/governance/overrides  # Custom overrides
GET  /api/admin/cato/governance/metrics     # Checkpoint metrics
GET  /api/admin/cato/governance/history     # Preset changes audit
POST /api/admin/cato/governance/checkpoint  # Record decision
GET  /api/admin/cato/governance/pending     # Pending checkpoints
POST /api/admin/cato/governance/resolve     # Resolve checkpoint
```

Migration: `V2026_01_20_013__governance_presets.sql`

War Room (Council of Rivals Visualization) Real-time multi-agent adversarial debate interface.

Council Member Roles:

Role	Purpose	Icon
Advocate	Argues in favor	
Critic	Identifies flaws	
Synthesizer	Combines viewpoints	
Specialist	Domain expertise	
Contrarian	Challenges assumptions	

Debate Flow:

`Topic → Opening → Arguments → Rebuttals → Voting → Verdict`

Verdict Outcomes: `consensus`, `majority`, `split`, `deadlock`, `synthesized`

Files: - `lambda/shared/services/council-of-rivals.service.ts` - Core debate service - `apps/admin-dashboard/app/(dashboard)/cato/war-room/page.tsx` - War Room UI

UI Features: - Amphitheater-style member avatars - Real-time debate transcript with live polling (2s) - Arguments with confidence bars and evidence badges - Rebuttals with strength indicators - Verdict panel with synthesized answers

5.4 Sovereign Mesh Services (v5.31.0)

Parametric AI assistance at every workflow node.

Philosophy: "Every Node Thinks. Every Connection Learns. Every Workflow Assembles Itself."

Service	Purpose
`sovereign-mesh/ai-helper.service.ts`	Disambiguation, inference, recovery, validation
`sovereign-mesh/agent-runtime.service.ts`	OODA-loop agent execution
`sovereign-mesh/notification.service.ts`	Email/Slack/webhook notifications
`sovereign-mesh/snapshot-capture.service.ts`	Execution state snapshots

Worker Lambdas: - `workers/agent-execution-worker.ts` - SQS-triggered OODA processing - `workers/transparency-compiler.ts` - Pre-compute decision explanations

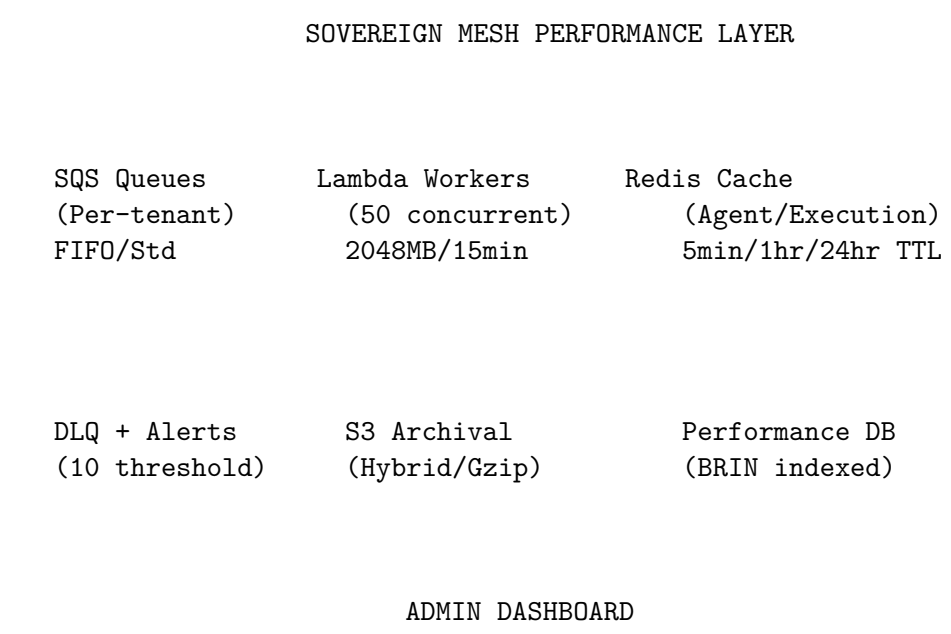
Scheduled Lambdas: - `app-registry-sync` - Daily sync from Activepieces/n8n (2 AM UTC) - `hitl-sla-monitor` - SLA monitoring and escalation (every minute) - `app-health-check` - Hourly health check for top 100 apps

5.4.1 Sovereign Mesh Performance Optimization (v5.38.0)

Scale-ready execution infrastructure for autonomous agent workloads.

Philosophy: "Every execution tracked. Every bottleneck visible. Every setting tunable."

Architecture Diagram



`Overview` `Scaling` `Caching` `Alerts` `Recommendations`

Configurable Settings (All Persistent in Database)

Setting	Table Column	Default	Range	Admin UI Location
Lambda Settings				Scaling Tab
Max Concurrency	`agent_worker_config.max_concurrency`	50	1-200	Slider
Provisioned Concurrency	`agent_worker_config.provisioned_concurrency`	5	0-50	Slider
Memory (MB)	`agent_worker_config.memory_mb`	2048	512-4096	Dropdown
Timeout (sec)	`agent_worker_config.timeout_seconds`	900	60-900	Input
Reserved Concurrency	`agent_worker_config.reserved_concurrency`	100	0-1000	Input
SQS Settings				Scaling Tab
Batch Size	`sqs_config.batch_size`	1	1-10	Dropdown
Visibility Timeout	`sqs_config.visibility_timeout_seconds`	900	30-43200	Input
Message Retention	`sqs_config.message_retention_days`	4	1-14	Input
DLQ Max Receives	`sqs_config.dlq_max_receive_count`	3	1-10	Input
Scaling Settings				Scaling Tab
Strategy	`scaling_config.strategy`	auto	fixed/auto/scheduled	Dropdown
Min Instances	`scaling_config.min_instances`	0	0-50	Slider
Max Instances	`scaling_config.max_instances`	50	1-200	Slider
Target Utilization	`scaling_config.target_utilization`	70	50-95	Slider
Scale-in Cooldown	`scaling_config.scale_in_cooldown_seconds`	300	60-900	Input
Scale-out Cooldown	`scaling_config.scale_out_cooldown_seconds`	60	30-300	Input
Cache Settings				Caching Tab
Backend	`caching_config.backend`	redis	memory/redis	Toggle

Setting	Table Column	Default	Range	Admin UI Location
Agent TTL (sec)	`caching_config.agent_ttl_seconds`	300	60-3600	Slider
Execution TTL (sec)	`caching_config.execution_ttl_seconds`	3600	300-86400	Slider
Working Memory TTL	`caching_config.working_memory_ttl_seconds`	86400	3600-604800	Slider
Max Memory (MB)	`caching_config.max_memory_mb`	256	64-2048	Input
Eviction Policy	`caching_config.eviction_policy`	lru	lru/lfu/ttl	Dropdown
Tenant Isolation				Scaling Tab
Mode	`tenant_isolation_config.mode`	shared	shared/dedicated/hybrid	Dropdown
Max Per Tenant	`tenant_isolation_config.max_concurrent_per_tenant`	50	1-100	Slider
Max Per User	`tenant_isolation_config.max_concurrent_per_user`	10	1-25	Slider
Rate Limiting	`tenant_isolation_config.rate_limiting_enabled`	true	boolean	Switch
Archival Settings				(Future UI)
Storage Backend	`archival_config.storage_backend`	hybrid	database/s3/hybrid	Dropdown
Archive After Days	`archival_config.archive_after_days`	7	1-30	Input
Delete After Days	`archival_config.delete_after_days`	90	0-365	Input
Max DB Bytes	`archival_config.max_db_artifact_bytes`	65536	1024-1048576	Input
Compression	`archival_config.compression_algorithm`	gzip	none/gzip/lz4/zstd	Dropdown
Alert Thresholds				Alerts Tab
DLQ Alert Enabled	`alert_config.dlq_alert_enabled`	true	boolean	Switch
DLQ Threshold	`alert_config.dlq_alert_threshold`	10	1-100	Slider
Latency Alert Enabled	`alert_config.latency_alert_enabled`	true	boolean	Switch
Latency Threshold (ms)	`alert_config.latency_alert_threshold_ms`	30000	5000-120000	Slider
Error Rate Alert	`alert_config.error_rate_alert_enabled`	true	boolean	Switch
Error Rate Threshold	`alert_config.error_rate_threshold`	0.05	0.01-0.50	Slider

Setting	Table Column	Default	Range	Admin UI Location
Budget Alert	`alert_config.budget_alert_enabled`	true	boolean	Switch
Budget Threshold	`alert_config.budget_alert_threshold`	0.80	0.50-0.95	Slider

Estimated Max Concurrent Sessions Based on current configuration:

Component	Calculation	Max
Lambda Concurrency	Reserved (100) × Provisioned (5) warm	**100 concurrent**
SQS Throughput	3,000 msg/sec standard queue	**180,000/min**
Redis Connections	ElastiCache r6g.large = 65,000	**65,000 cached**
API Gateway	Regional: 10,000 RPS default	**10,000 RPS**
Database Connections	Aurora r6g.large = 1,000 pooled	**1,000 active**

Theoretical Maximum API Sessions: - **Sustained**: ~10,000 concurrent sessions (API Gateway limit) - **Burst**: ~50,000 concurrent (with Lambda scaling + SQS buffering) - **With Dedicated Queues**: ~100,000 (per-tenant isolation)

Bottleneck Analysis: 1. API Gateway: 10,000 RPS (can request increase to 100,000) 2. Lambda Concurrency: 100 reserved → Scale to 1,000+ 3. Database: Aurora Serverless v2 scales to 256 ACUs (25,600 connections)

Services

Service	File	Purpose
SQS Dispatcher	`sqs-dispatcher.service.ts`	Message dispatch with tenant routing
Redis Cache	`redis-cache.service.ts`	Agent/execution caching with fallback
Performance Config	`performance-config.service.ts`	CRUD, recommendations, alerts
Artifact Archival	`artifact-archival.service.ts`	S3/hybrid storage with compression

API Endpoints Base: `/api/admin/sovereign-mesh/performance`

Endpoint	Method	Description
`/dashboard`	GET	Complete dashboard (health, metrics, alerts)
`/config`	GET	Get current configuration
`/config`	PUT/PATCH	Update configuration
`/recommendations`	GET	AI-generated recommendations
`/recommendations/:id/apply`	POST	Apply recommendation
`/alerts`	GET	Active alerts

Endpoint	Method	Description
`/alerts/:id/acknowledge`	POST	Acknowledge alert
`/alerts/:id/resolve`	POST	Resolve alert
`/cache/stats`	GET	Cache statistics
`/cache`	DELETE	Clear tenant cache
`/queue/metrics`	GET	Queue metrics
`/health`	GET	Health check

Database Tables

```
-- Core Configuration (persistent settings)
sovereign_mesh_performance_config (
  tenant_id UUID PRIMARY KEY,
  agent_worker_config JSONB,       -- Lambda settings
  transparency_worker_config JSONB,
  sqs_config JSONB,                -- Queue settings
  scaling_config JSONB,            -- Autoscaling
  caching_config JSONB,            -- Redis/memory
  archival_config JSONB,           -- S3 archival
  db_optimization_config JSONB,    -- Connection pooling
  tenant_isolation_config JSONB,   -- Rate limiting
  alert_config JSONB,              -- Alert thresholds
  created_at, updated_at
);

-- Alerts
sovereign_mesh_performance_alerts (
  id, tenant_id, alert_type, severity, message,
  triggered_at, acknowledged_at, acknowledged_by,
  resolved_at, resolved_by, auto_resolved
);

-- Time-Series Metrics (BRIN indexed)
sovereign_mesh_performance_metrics (
  id, tenant_id, metric_time, metric_type, metric_value,
  dimensions JSONB, tags TEXT[]
);

-- Artifact Archives
sovereign_mesh_artifact_archives (
  id, tenant_id, execution_id, snapshot_id,
  storage_backend, s3_bucket, s3_key,
  artifact_type, original_size_bytes, compressed_size_bytes,
  compression_algorithm, checksum_sha256,
  archived_at, expires_at, deleted_at
);
```

```
-- Tenant Queues
sovereign_mesh_tenant_queues (
  tenant_id, queue_type, queue_url, queue_arn,
  is_fifo, created_at, last_used_at
);

-- Rate Limits
sovereign_mesh_rate_limits (
  tenant_id, user_id, window_start,
  execution_count, last_execution_at
);

-- Config History (audit trail)
sovereign_mesh_config_history (
  id, tenant_id, changed_by, changed_at,
  change_type, previous_value, new_value
);
```

Key Performance Indexes

```
-- Fast execution queries
CREATE INDEX idx_agent_executions_tenant_status ON agent_executions(tenant_id, status);
CREATE INDEX idx_agent_executions_agent_status ON agent_executions(agent_id, status);
CREATE INDEX idx_agent_executions_created_at ON agent_executions(created_at DESC);

-- Partial index for running executions only
CREATE INDEX idx_agent_executions_running ON agent_executions(tenant_id, started_at)
  WHERE status = 'running';

-- BRIN index for time-series (efficient for append-only)
CREATE INDEX idx_perf_metrics_tenant_time ON sovereign_mesh_performance_metrics
  USING BRIN (tenant_id, metric_time);
```

Admin Dashboard UI Location: `apps/admin-dashboard/app/(dashboard)/sovereign-mesh/performance/`

5 Tabs: 1. **Overview**: Health score, active/pending executions, queue depth, cache hit rate, OODA timing, cost estimate 2. **Scaling**: Lambda concurrency sliders, tenant isolation mode, rate limits 3. **Caching**: Cache backend, TTLs, statistics, clear cache action 4. **Alerts**: DLQ/latency/error/budget thresholds, active alerts with acknowledge/resolve 5. **Recommendations**: AI-generated optimizations with one-click apply

5.4.2 Infrastructure Scaling System (v5.38.0)

Scale from 100 to 500,000+ concurrent sessions with cost-aware tier selection.

Philosophy: "Pay only for what you need. Scale instantly when you need more."

Scaling Tiers

Tier	Sessions	Monthly Cost	Infrastructure
Development	100	$70	Scale-to-zero, minimal resources
Staging	1,000	$500	Basic redundancy
Production	10,000	$5,000	High availability
Enterprise	500,000	$68,500	Multi-region, global scale

Architecture Diagram

```
                    INFRASTRUCTURE SCALING SYSTEM


                          TIER SELECTION

   DEV          STAGING        PROD          ENTERPRISE
  100            1,000        10,000          500,000
  $70/mo       $500/mo        $5K/mo          $68K/mo


                       COMPONENT CONFIGURATION

   Lambda         Aurora         Redis           API           SQS
  0-1000         0.5-256        1-10          100-100K      2-100
  conc.          ACU            shards        RPS           queues


                          COST ESTIMATION
  Lambda: $X + Aurora: $Y + Redis: $Z + API: $A + SQS: $B = Total
  Cost per session: $Total / MaxSessions
```

Component Configuration by Tier

Component	Development	Staging	Production	Enterprise
Lambda Reserved	0	10	100	1,000
Lambda Provisioned	0	0	5	100

Component	Development	Staging	Production	Enterprise
Lambda Max	10	50	200	1,000
Lambda Memory	1024 MB	2048 MB	2048 MB	3072 MB
Aurora Min ACU	0.5	1	4	16
Aurora Max ACU	2	8	64	256
Aurora Replicas	0	1	2	3
Aurora Global	No	No	No	Yes (5 regions)
Redis Node	t4g.micro	t4g.small	r6g.large	r6g.xlarge
Redis Shards	1	1	1	10
Redis Cluster	No	No	No	Yes
API Rate Limit	100	1,000	10,000	100,000
CloudFront	No	No	Yes	Yes
SQS Standard	2	5	10	50
SQS FIFO	0	2	5	50

Cost Calculation Formula

```
// Lambda provisioned concurrency
lambdaCost = provisionedConcurrency × (memoryMb / 1024) × 0.000004167 × 3600 × 24 × 30;

// Aurora (average ACU)
auroraCost = ((minAcu + maxAcu) / 2) × 0.12 × 24 × 30 × (1 + replicas × 0.5);

// Redis
redisCost = nodePrice × 24 × 30 × numShards × (1 + replicasPerShard);

// API Gateway (estimated 10% utilization)
apiCost = (rateLimit × 0.1 × 3600 × 24 × 30 / 1e6) × 1.00;

// Total
totalMonthlyCost = lambdaCost + auroraCost + redisCost + apiCost + sqsCost + cloudFrontCost;
costPerSession = totalMonthlyCost / maxSessions;
```

Session Capacity Calculation

```
const capacities = {
  lambda: maxConcurrency × 10,           // 10 sessions per concurrent execution
  aurora: connectionPoolSize × 50,        // 50 sessions per connection
  redis: maxConnections,                  // Direct connection limit
  apiGateway: throttlingRateLimit,        // RPS limit
};

const maxSessions = Math.min(...Object.values(capacities));
const bottleneck = Object.entries(capacities)
  .find(([_, v]) => v === maxSessions)?.[0];
```

Services

Service	File	Purpose
Scaling Service	`scaling.service.ts`	Profile management, cost calculation
Session Metrics	`scaling.service.ts`	Real-time session tracking
Cost Estimator	`scaling.service.ts`	AWS pricing calculations

API Endpoints Base: `/api/admin/sovereign-mesh/scaling`

Endpoint	Method	Description
`/dashboard`	GET	Complete scaling dashboard
`/profiles`	GET	List all profiles
`/profiles/:id/apply`	POST	Apply profile
`/sessions`	GET	Session metrics
`/sessions/capacity`	GET	Capacity info
`/cost`	GET	Current cost estimate
`/cost/estimate`	POST	Estimate custom config
`/presets/:tier/apply`	POST	Apply preset tier

Database Tables

```
-- Scaling profiles
sovereign_mesh_scaling_profiles (
  id, tenant_id, name, tier, target_sessions,
  lambda_*, aurora_*, redis_*, api_*, sqs_*,
  estimated_monthly_cost, is_active
);

-- Session metrics (1-minute granularity)
sovereign_mesh_session_metrics (
  tenant_id, metric_time, active_sessions, pending_sessions,
  sessions_by_region JSONB, utilization_percent
);

-- Hourly aggregates
sovereign_mesh_session_metrics_hourly (
  tenant_id, hour_start, total_sessions, peak_concurrent,
  lambda_cost, aurora_cost, redis_cost, total_cost
);

-- Scaling operations
sovereign_mesh_scaling_operations (
  id, tenant_id, operation_type, status,
  source_profile_id, target_profile_id, changes JSONB
);

-- Cost records
```

```
sovereign_mesh_cost_records (
  tenant_id, record_date, lambda_cost, aurora_cost, ...,
  total_cost, sessions_count, cost_per_session
);
```

Admin Dashboard UI Location: `apps/admin-dashboard/app/(dashboard)/sovereign-mesh/scaling/page`

5 Tabs: 1. **Overview**: Active sessions, peak, bottleneck, cost/session, component health 2. **Sessions**: Capacity gauge, statistics, per-component limits 3. **Infrastructure**: Lambda/Aurora/Redis/API Gateway configuration cards 4. **Cost**: Component breakdown, cost metrics, annual estimate 5. **Scale**: One-click tier selection, comparison, change list

5.5 Gateway Services (v5.28.0-5.29.0)

Multi-protocol WebSocket/SSE gateway for 1M+ concurrent connections.

Component	Technology	Purpose
Go Gateway	Go 1.22 + gobwas/ws	WebSocket termination, 100K+ connections/instance
Egress Proxy	Node.js + HTTP/2	Connection pooling to AI providers
NATS JetStream	NATS 2.10	Message broker with INBOX + HISTORY streams

Files: - `apps/gateway/` - Go gateway service (12 files) - `services/egress-proxy/` - HTTP/2 proxy service (5 files) - `lambda/admin/gateway.ts` - Gateway admin API

Supported Protocols: MCP, A2A, OpenAI, Anthropic, Google

5.6 Code Quality Services (v5.30.0)

Test coverage, technical debt, and code quality monitoring.

Endpoint	Purpose
`/api/admin/code-quality/dashboard`	Coverage, debt, JSON safety metrics
`/api/admin/code-quality/coverage`	Component-level coverage breakdown
`/api/admin/code-quality/debt`	Technical debt items
`/api/admin/code-quality/alerts`	Quality regression alerts

Files: - `lambda/admin/code-quality.ts` - Admin API handler - `apps/admin-dashboard/app/(dashboard)/code` - Dashboard UI

---

6. CDK Stack Architecture

6.1 Stack Dependency Graph

```
FoundationStack
       NetworkingStack
           SecurityStack
               DataStack
               StorageStack
               AuthStack
                   AIStack
                   ApiStack (411 resources)
                   AdminStack
                   ThinkTankAdminApiStack
                   ThinkTankAuthStack
                   BrainStack (Tier 3+)
                   CatoRedisStack (Tier 2+)
                   CatoGenesisStack
                   CatoTierTransitionStack
                   ConsciousnessStack
                   CognitionStack
                   FormalReasoningStack
                   GrimoireStack
                   CollaborationStack
                   LibraryRegistryStack
                   LibraryExecutionStack
                   ScheduledTasksStack
                   SecurityMonitoringStack
                   MonitoringStack
                   WebhooksStack
                   UserRegistryStack
                   MissionControlStack
                   ModelSyncSchedulerStack
                   BatchStack
                   TMSStack
```

6.2 All CDK Stacks (33 total)

Stack	Purpose	Tier Requirement
`foundation-stack`	Base infrastructure	All
`networking-stack`	VPC, subnets	All
`security-stack`	Security groups, KMS	All
`data-stack`	Aurora PostgreSQL	All
`storage-stack`	S3 buckets	All
`auth-stack`	Cognito user pools	All
`ai-stack`	LiteLLM, SageMaker	All
`api-stack`	API Gateway, Lambdas	All

Stack	Purpose	Tier Requirement
`admin-stack`	Admin dashboard hosting	All
`brain-stack`	AGI Brain, SOFAI	Tier 3+
`cato-redis-stack`	ElastiCache Redis	Tier 2+
`cato-genesis-stack`	Cato safety architecture	All
`cato-tier-transition-stack`	Step Functions for tier changes	All
`consciousness-stack`	Consciousness services	Tier 3+
`cognition-stack`	Advanced cognition	Tier 3+
`formal-reasoning-stack`	Z3, RDFLib execution	Tier 3+
`thinktank-auth-stack`	Think Tank authentication	All
`thinktank-admin-api-stack`	Think Tank admin APIs	All
`gateway-stack`	Multi-protocol WebSocket/SSE gateway	All
`sovereign-mesh-stack`	Agent registry, app registry, AI helper	All

6.3 Resource Limits

- CloudFormation max: **500 resources per stack**
- Current API stack: **411 resources**
- Strategy: Proxy routes, consolidated handlers

---

7. Security & Compliance

7.1 Authentication

- **User auth**: Cognito User Pools
- **Admin auth**: Separate Cognito pool with MFA
- **API auth**: JWT tokens via API Gateway authorizers

7.2 Data Protection

- **Encryption at rest**: AES-256 (Aurora, S3)
- **Encryption in transit**: TLS 1.3
- **Key management**: AWS KMS

7.3 Compliance Frameworks

Framework	Implementation
HIPAA	PHI sanitization, audit logs
SOC 2 Type II	Continuous monitoring
FDA 21 CFR Part 11	Electronic signatures, audit trails

Framework	Implementation
GDPR	Data portability, right to deletion

---

8. Libraries & Dependencies

8.1 Core TypeScript Dependencies

```
{
  "aws-cdk-lib": "^2.170.0",
  "aws-lambda": "^1.0.7",
  "@aws-sdk/client-*": "^3.x",
  "pg": "^8.11.3",
  "ioredis": "^5.3.2",
  "zod": "^3.22.4"
}
```

8.2 AI/ML Libraries

```
{
  "openai": "^4.x",
  "@anthropic-ai/sdk": "^0.x",
  "@google/generative-ai": "^0.x",
  "litellm": "proxy deployment"
}
```

8.3 Python Dependencies (Lambda Layers)

```
z3-solver          # Formal verification
rdflib             # Knowledge graphs
owlrl              # OWL reasoning
pyshacl            # SHACL validation
pyreason           # Probabilistic reasoning
numpy              # Numerical computing
networkx           # Graph algorithms
```

---

9. AI Report Writer Pro (v5.42.0)

9.1 Overview

The AI Report Writer is an enterprise-grade report generation system that combines natural language processing, voice input, interactive visualizations, AI-powered insights, and brand customization. Available in both RADIANT Admin and Think Tank Admin dashboards.

9.2 Architecture

AI Report Writer

Input Layer	Natural Language Parser ← Text/Voice (Web Speech API)
Generation	AI Model → Structured Report (sections, charts, tables)
Visualization	Recharts → Bar, Line, Pie, Area (responsive)
Analysis	Smart Insights Engine → Anomalies, Trends, Recs
Branding	Brand Kit → Logo, Colors, Fonts → Styled Export
Export	PDF / Excel / HTML / Print

9.3 Core Interfaces

```typescript
interface GeneratedReport {
  title: string;
  subtitle?: string;
  executiveSummary?: string;
  sections: ReportSection[];
  charts?: ChartConfig[];
  tables?: TableConfig[];
  smartInsights?: SmartInsight[];
  metadata: { generatedAt: string; dataRange?: string; confidence: number };
}

interface SmartInsight {
  id: string;
  type: 'anomaly' | 'trend' | 'recommendation' | 'warning' | 'achievement';
  title: string;
  description: string;
  metric?: string;
  value?: string;
  change?: string;
  severity: 'low' | 'medium' | 'high';
  confidence: number;
}

interface BrandKit {
  logoUrl: string | null;
  primaryColor: string;
  secondaryColor: string;
  accentColor: string;
  fontFamily: string;
  headerFont: string;
  companyName: string;
```

```
  tagline: string;
}
```

9.4 Interactive Charts

Uses Recharts library with consistent 8-color palette:

```
const CHART_COLORS = [
  '#3b82f6', // Blue
  '#10b981', // Emerald
  '#f59e0b', // Amber
  '#ef4444', // Red
  '#8b5cf6', // Violet
  '#ec4899', // Pink
  '#06b6d4', // Cyan
  '#84cc16', // Lime
];
```

Chart Types: - `RechartsBarChart` - Category comparisons with colored bars - `RechartsLineChart` - Time series with smooth curves - `RechartsPieChart` - Proportional data with donut style

9.5 Heatmap Visualization Components (v5.52.1)

Industry-leading heatmap implementations with unique differentiators.

Component Architecture

Component	Location	Purpose
`ActivityHeatmap`	`apps/thinktank/components/ui/activity-heatmap.tsx`	GitHub-style yearly activity
`EnhancedActivityHeatmap`	`apps/thinktank/components/ui/enhanced-activity-heatmap.tsx`	Advanced with 10 differentiators
`Heatmap`	`apps/admin-dashboard/components/charts/heatmap.tsx`	Generic 2D grid
`LatencyHeatmap`	`apps/admin-dashboard/components/charts/latency-heatmap.tsx`	AWS geographic latency
`CBFViolationsHeatmap`	`apps/admin-dashboard/components/charts/cbf-violations-heatmap`	Rule violation analysis

Enhanced Activity Heatmap Technical Implementation

```
// Breathing animation using requestAnimationFrame
useEffect(() => {
  if (!enableBreathing) return;
  let frame: number;
  const animate = (timestamp: number) => {
    const elapsed = (timestamp - start) / 1000;
    setBreathPhase(Math.sin(elapsed * 0.5) * 0.5 + 0.5); // 0-1 cycle
    frame = requestAnimationFrame(animate);
  };
  frame = requestAnimationFrame(animate);
  return () => cancelAnimationFrame(frame);
}, [enableBreathing]);
```

```
// AI insights generation
function generateAIInsights(data: ActivityDay[], streaks: Streak[]): AIInsight[] {
  // Pattern detection: weekday vs weekend
  // Streak achievement badges
  // Anomaly detection (3x+ average)
  // Trend predictions
}

// Sound feedback using Web Audio API
const playSound = (intensity: number) => {
  const ctx = new AudioContext();
  const osc = ctx.createOscillator();
  osc.frequency.value = 200 + intensity * 400;
  // Pitch varies with activity intensity
};
```

Color Schemes

```
const COLOR_SCHEMES = {
  violet: { levels: ['#4c1d95', '#6d28d9', '#8b5cf6', '#a78bfa', '#c4b5fd'], glow: 'rgba(139, 9
  green: { levels: ['#0e4429', '#006d32', '#26a641', '#39d353', '#a6f8b0'], glow: 'rgba(57, 211
  blue: { levels: ['#1e3a5f', '#2563eb', '#3b82f6', '#60a5fa', '#93c5fd'], glow: 'rgba(59, 130
  fire: { levels: ['#7f1d1d', '#b91c1c', '#ef4444', '#f87171', '#fecaca'], glow: 'rgba(239, 68
  ocean: { levels: ['#0e5357', '#0d9488', '#14b8a6', '#2dd4bf', '#99f6e4'], glow: 'rgba(20, 184
};
```

Accessibility Implementation

```
// Screen reader narrative mode
{showAccessibility && (
  <div role="status" aria-live="polite">
    <p>Activity Summary for {year}</p>
    <ul>
      <li>Total interactions: {totalActivity.toLocaleString()}</li>
      <li>Active days: {data.filter(d => d.count > 0).length}</li>
      <li>Current streak: {currentStreak?.length} days</li>
    </ul>
  </div>
)}
```

Competitive Differentiators

Feature	RADIANT	GitHub	Competitors
Breathing Animation			
AI Insights			
Sound Feedback			
Streak Gamification		Basic	

Feature	RADIANT	GitHub	Competitors
Accessibility Narrative		Basic	
Predictions			
5 Color Schemes		1	1-2

9.6 Smart Insights Engine

AI-powered analysis that surfaces actionable insights:

Type	Color	Purpose
`trend`	Blue	Growth patterns, trajectory predictions
`anomaly`	Amber	Unusual data spikes, deviations
`achievement`	Green	Positive milestones, records
`recommendation`	Purple	Actionable suggestions
`warning`	Red	Concerning metrics, alerts

Each insight includes: - **Severity**: low/medium/high - **Confidence Score**: 0-100% - **Metric/Value/Change**: Quantified data points

9.6 Brand Kit Customization

Enables enterprise branding of generated reports:

Component	Implementation
Logo	FileReader → data URL, stored in state
Colors	HTML5 color pickers (primary/secondary/accent)
Fonts	Select dropdown (Inter, Georgia, Roboto, etc.)
Preview	Live-updating Card component

9.7 Voice Input

Web Speech API integration for hands-free report generation:

```
const SpeechRecognitionConstructor = (
  window.SpeechRecognition || window.webkitSpeechRecognition
) as new () => SpeechRecognition;

const recognition = new SpeechRecognitionConstructor();
recognition.continuous = true;
recognition.interimResults = true;
recognition.lang = 'en-US';
```

9.8 Files

File	Purpose
apps/admin-dashboard/app/(dashboard)/reports/page.tsx	BYOD/ANT Admin UI
apps/thinktank-admin/app/(dashboard)/reports/page.tsx	ThinkTank Admin UI
apps/admin-dashboard/lib/api/ai-reports.ts	Frontend API client
apps/thinktank-admin/lib/api/ai-reports.ts	Frontend API client
packages/infrastructure/lambda/admin/ai_reports.ts	Lambda handlers
packages/infrastructure/lambda/shared/report_export.ts	PDF/Excel/HTML export utilities
packages/infrastructure/migrations/V2026_01_21_005__ai_reports.sql	Database schema

9.9 API Endpoints

Method	Endpoint	Description
GET	/admin/ai-reports	List reports
POST	/admin/ai-reports/generate	Generate new report
GET	/admin/ai-reports/:id	Get report by ID
PUT	/admin/ai-reports/:id	Update report
DELETE	/admin/ai-reports/:id	Delete report
POST	/admin/ai-reports/:id/export	Export report (PDF/Excel/HTML)
GET	/admin/ai-reports/templates	List templates
POST	/admin/ai-reports/templates	Create template
GET	/admin/ai-reports/brand-kits	List brand kits
POST	/admin/ai-reports/brand-kits	Create brand kit
PUT	/admin/ai-reports/brand-kits/:id	Update brand kit
DELETE	/admin/ai-reports/brand-kits/:id	Delete brand kit
POST	/admin/ai-reports/chat	Send chat message for modifications
GET	/admin/ai-reports/insights	Get insights dashboard

9.10 Database Tables

Table	Purpose
brand_kits	Logo, colors, fonts, company info
report_templates	Reusable report structures
generated_reports	AI-generated reports with content
report_smart_insights	Extracted insights (denormalized)
report_exports	Export records with S3 references
report_chat_history	Interactive chat for modifications
report_schedules	Scheduled automatic generation

9.11 Future Enhancements

- Real-time data integration via API endpoints
- Scheduled report generation
- Report templates library
- Collaborative editing

- Version history with diff view

---

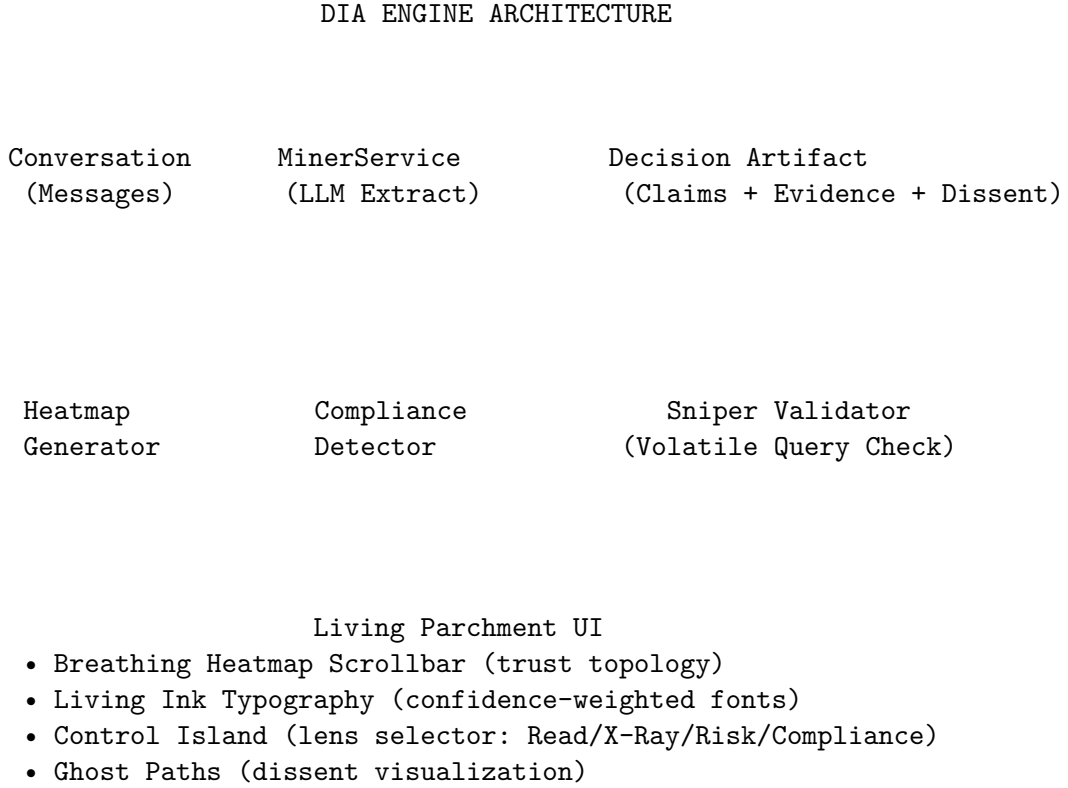
10. Decision Intelligence Artifacts (DIA Engine) (v5.43.0)

The Glass Box Decision Engine transforms AI conversations into auditable, evidence-backed decision records with full provenance tracking.

10.1 Problem Statement

AI decisions suffer from **opacity**—users can't see why AI reached conclusions, what evidence supports claims, or whether underlying data has changed. This creates compliance risks and trust gaps in enterprise deployments.

10.2 Architecture



10.3 Core Services

Service	Purpose
`MinerService`	LLM-powered extraction of claims, evidence, dissent from conversations
`HeatmapGenerator`	Generates trust topology visualization data
`ComplianceDetector`	Detects HIPAA/SOC2/GDPR relevance and PHI/PII
`SniperValidator`	Validates volatile queries for staleness
`ComplianceExporter`	Generates compliance export packages

10.4 Claim Extraction

The MinerService uses Claude 3.5 Sonnet to extract structured claims:

```
interface DAClaim {
  id: string;
  type: 'conclusion' | 'finding' | 'recommendation' | 'warning' | 'fact' |
        'clinical_finding' | 'treatment_recommendation' | 'risk_assessment' |
        'legal_opinion' | 'compliance_finding';
  content: string;
  confidence: number;           // 0-100
  evidenceIds: string[];        // Links to supporting evidence
  volatileQueryIds: string[];   // Links to time-sensitive data
  position: { start: number; end: number };
  metadata: {
    extractedAt: string;
    modelUsed: string;
    promptVersion: string;
  };
}
```

10.5 Evidence Linking

Each claim links to evidence sources:

```
interface DAEvidenceLink {
  id: string;
  type: 'tool_call' | 'web_search' | 'document' | 'calculation' | 'model_consensus';
  sourceId: string;             // Reference to original source
  excerpt: string;              // Relevant portion
  relevanceScore: number;       // 0-100
  verificationStatus: 'verified' | 'unverified' | 'disputed';
}
```

10.6 Volatile Query Tracking

Tool calls that may return different results over time are tracked:

```
interface DAVolatileQuery {
  id: string;
```

```
  toolCallId: string;
  volatility: 'real-time' | 'daily' | 'weekly' | 'stable';
  lastValidatedAt: string;
  stalenessThreshold: number;   // Hours
  originalResult: Record<string, unknown>;
  currentResult?: Record<string, unknown>;
  changeDetected: boolean;
}
```

10.7 Compliance Export Formats

Format	Package Contents
`hipaa_audit`	PHI inventory, access log, evidence chain, system attestation
`soc2_evidence`	Control mapping (CC6/7/8), evidence verification, change management
`gdpr_dsar`	PII inventory, lawful basis, processing activities, data subject info

10.8 Living Parchment UI

Breathing Heatmap Scrollbar: - CSS animation with `scale` transforms at different BPM rates - Green (verified): 6 BPM, Amber (unverified): 8 BPM, Red (contested): 12 BPM - Hover reveals segment tooltip with claim count

Living Ink Typography: - Font weight scales from 350 (low confidence) to 500 (high confidence) - Stale claims fade to grayscale via CSS `filter: grayscale()` - Hover triggers evidence link highlighting

Control Island: - Floating fixed-position component - Lens buttons toggle document visualization mode - Export and Validate actions with loading states

10.9 Database Schema

Table	Purpose
`decision_artifacts`	Core artifact with JSONB content
`decision_artifact_validation_log`	Validation audit trail
`decision_artifact_export_log`	Export audit trail
`decision_artifact_config`	Tenant configuration
`decision_artifact_templates`	Extraction templates
`decision_artifact_access_log`	HIPAA access audit

10.10 CDK Infrastructure

```
// DIAStack provides:
// - S3 bucket for compliance exports (90-day lifecycle)
// - SQS queue for async extraction (15-min visibility timeout)
// - DLQ for failed extractions (14-day retention)
```

```
export class DIAStack extends cdk.Stack {
  public readonly exportBucket: s3.Bucket;
  public readonly extractionQueue: sqs.Queue;
  public readonly extractionDLQ: sqs.Queue;
}
```

10.11 Implementation Files

File	Purpose
packages/shared/src/types/decision-artifacts.ts	Type definitions
packages/infrastructure/lambda/shared/services/dia/miner.service.ts	Pattern mining
packages/infrastructure/lambda/shared/services/dia/heatmap-generator.ts	Heatmap generation
packages/infrastructure/lambda/shared/services/dia/compliance-detector.ts	PII/PHI detection
packages/infrastructure/lambda/shared/services/dia/sniper-validator.ts	Statistical validation
packages/infrastructure/lambda/shared/services/dia/compliance-exporter.ts	Export generation
packages/infrastructure/lambda/thinktank/decision-artifacts.ts	API handler
packages/infrastructure/lib/stacks/dia-stack.ts	CDK stack
apps/thinktank-admin/app/(dashboard)/decision-records/	Admin UI

---

11. Supporting Documentation

11.1 API Documentation

OpenAPI 3.1 specification for the Admin API: - **File**: `docs/api/openapi-admin.yaml` - **Coverage**: Tenants, AI Reports, Models, Providers, Billing - **Format**: YAML with full request/response schemas

11.2 Performance Optimization Guide

Comprehensive performance documentation: - **File**: `docs/PERFORMANCE-OPTIMIZATION.md` - **Topics**: - Lambda cold start optimization - Database query optimization - Caching strategies (in-memory, Redis, API Gateway) - Response compression and pagination - AI model call optimization (streaming, batching, prompt caching) - Frontend performance (code splitting, SWR, image optimization) - Monitoring and alerting - Cost optimization

11.3 Security Audit Checklist

Security compliance and audit documentation: - **File**: `docs/SECURITY-AUDIT-CHECKLIST.md` - **Topics**: - Row-Level Security (RLS) policies for all tables - Authentication flow verification - Authorization patterns and permission hierarchy - Input validation and sanitization - API security (rate limiting, CORS, headers) - Data protection (encryption at rest/transit, PII handling) - Secret management - Audit logging - OWASP Top 10 coverage - Compliance requirements (SOC 2, GDPR, HIPAA, CCPA)

---

11. Living Parchment 2029 Vision (v5.44.0)

11.1 Architecture Overview

Living Parchment is a comprehensive suite of advanced decision intelligence tools featuring sensory UI elements that communicate trust, confidence, and data freshness through visual breathing, living typography, and ghost paths.

```
                    LIVING PARCHMENT STACK

  UI Layer (Next.js + React)
     War Room (Confidence Terrain, AI Advisors)
     Council of Experts (Consensus Visualization)
     Debate Arena (Attack/Defense Flows)
     Memory Palace (3D Knowledge Topology) [Coming Soon]
     Oracle View (Predictive Landscape) [Coming Soon]
     Synthesis Engine (Multi-Source Fusion) [Coming Soon]
     Cognitive Load Monitor [Coming Soon]
     Temporal Drift Observatory [Coming Soon]

  API Layer (Lambda + API Gateway)
     living-parchment.ts - Unified handler for all features

  Service Layer
     war-room.service.ts
     council-of-experts.service.ts
     debate-arena.service.ts

  Database (Aurora PostgreSQL)
     40+ tables with RLS policies
```

11.2 Design Philosophy

Concept	Technical Implementation
Breathing Interfaces	CSS keyframe animations at 4-12 BPM; faster = uncertainty
Living Ink	Font weight 350-500 calculated from confidence scores
Ghost Paths	Opacity 0.3-0.5 overlays for rejected alternatives
Confidence Terrain	3D grid with elevation = confidence, color = risk

11.3 War Room (Strategic Decision Theater)

High-stakes collaborative decision space with AI advisors.

Core Components: - `ConfidenceTerrain` - 10x10 grid visualization with elevation mapping - `AdvisorCard` - AI advisor with breathing aura animation - `DecisionPathCard` - Branching options with outcome predictions

Advisor Types:

```typescript
type AdvisorType = 'ai_model' | 'human_expert' | 'domain_specialist';

interface WarRoomAdvisor {
  id: string;
  type: AdvisorType;
  name: string;
  modelId?: string;
  specialization: string;
  confidence: number;
  breathingAura: { color: string; rate: BreathingRate; intensity: number };
  position: WarRoomPosition;
}
```

11.4 Council of Experts

Multi-persona AI consultation with 8 distinct personas.

Personas: | Persona | Color | Specialization | |———|——-|————-| | Pragmatist | #3b82f6 | Practical Implementation | | Ethicist | #8b5cf6 | Moral Philosophy | | Innovator | #f59e0b | Creative Solutions | | Skeptic | #ef4444 | Risk Analysis | | Synthesizer | #22c55e | Integration | | Analyst | #06b6d4 | Data-Driven Analysis | | Strategist | #ec4899 | Long-term Strategy | | Humanist | #14b8a6 | Human Impact |

Consensus Calculation: - Experts positioned on circular SVG visualization - Distance from center inversely proportional to consensus - Dissent sparks rendered as animated circles between disagreeing experts

11.5 Debate Arena

Adversarial exploration with attack/defense flows.

Resolution Tracking:

```typescript
interface ResolutionTracker {
  currentBalance: number;  // -100 (opposition) to +100 (proposition)
  balanceHistory: { timestamp: string; balance: number; triggerArgumentId: string }[];
  projectedOutcome: 'proposition' | 'opposition' | 'undecided';
  confidenceInProjection: number;
}
```

Steel-Man Generation: - AI creates strongest version of opponent's argument - Improvements listed for transparency - Visual overlay with enhancement glow

11.6 Database Schema

```sql
-- Core enums
CREATE TYPE war_room_status AS ENUM ('planning', 'active', 'deliberating', 'decided', 'archive
CREATE TYPE stake_level AS ENUM ('low', 'medium', 'high', 'critical');
CREATE TYPE council_status AS ENUM ('convening', 'debating', 'converging', 'concluded');
CREATE TYPE debate_status AS ENUM ('setup', 'opening', 'main', 'rebuttal', 'closing', 'resolve
```

```
-- Key tables
war_room_sessions, war_room_participants, war_room_advisors
council_sessions, council_experts, expert_arguments, minority_reports
debate_arenas, debaters, debate_arguments, weak_points, steel_man_overlays
living_parchment_config
```

11.7 Implementation Files

```
packages/shared/src/types/living-parchment.types.ts
packages/infrastructure/migrations/V2026_01_22_004__living_parchment_core.sql
packages/infrastructure/lambda/shared/services/living-parchment/
    war-room.service.ts
    council-of-experts.service.ts
    debate-arena.service.ts
    index.ts
packages/infrastructure/lambda/thinktank/living-parchment.ts
apps/thinktank-admin/app/(dashboard)/living-parchment/
    page.tsx                    # Landing page
    war-room/page.tsx           # War Room UI
    council/page.tsx            # Council of Experts UI
    debate/page.tsx             # Debate Arena UI
```

---

12. Cortex Memory System (v4.20.0)

12.0 Simple Overview: Cato vs Cortex

Before diving into technical details, here's the simple explanation:

System	What It Is	What It Stores	Analogy
Cato	AI's personality & feelings	Preferences, mood, personal memory	Your brain's personality
Cortex	Enterprise knowledge library	Facts, documents, relationships	Your company's wiki
Bridge	Connection between them	Sync + enrichment	Memory consolidation

How They Work Together:

```
USER MESSAGE: "What's the IC50 of Compound X?"
```

```
CATO checks:
• User prefers technical details
• User is a senior researcher
• Current mood: engaged, curious
```

```
CORTEX retrieves:
• Compound X: IC50 = 2.3nM against Target Y
• Related: selectivity data, assay conditions
• Source: Internal assay report 2025-Q4
```

```
AI RESPONSE:
Personalized (detailed, technical tone)
+ Informed (actual company data)
+ Trustworthy (cites internal source)
```

Result: Every Think Tank response draws from both personal context (Cato) AND enterprise knowledge (Cortex), creating responses that are personalized AND authoritative.

12.0.1 Cortex Intelligence Service (v5.52.15)

The **Cortex Intelligence Service** measures knowledge density and provides insights that influence:

Decision	Without Cortex	With Cortex
Domain Detection	Keyword matching only	+30% confidence boost if Cortex has rich knowledge
Orchestration Mode	Based on prompt complexity	Switches to `research` mode if expert knowledge available
Model Selection	Based on proficiency scores	Prefers factual models when Cortex has facts

How It Works:

```
USER PROMPT: "What's the IC50 of Compound X?"
```

```
CORTEX INTELLIGENCE SERVICE

1. Extract search terms: ["IC50", "Compound", "X"]
2. Query cortex_graph_nodes for matches
3. Count nodes by type: {fact: 15, entity: 8, procedure: 3}
4. Calculate knowledge depth: "rich" (26 nodes)
5. Generate recommendations:
   • Confidence boost: +0.18
   • Orchestration: "research" (rich knowledge available)
```

- Model: prefer factual models (15 facts > 3 procedures)

AGI BRAIN PLANNER applies insights:

- Domain confidence: 0.72 → 0.90 (+0.18 boost)
- Orchestration: thinking → research
- Plan includes: "Cortex: Rich enterprise knowledge available"

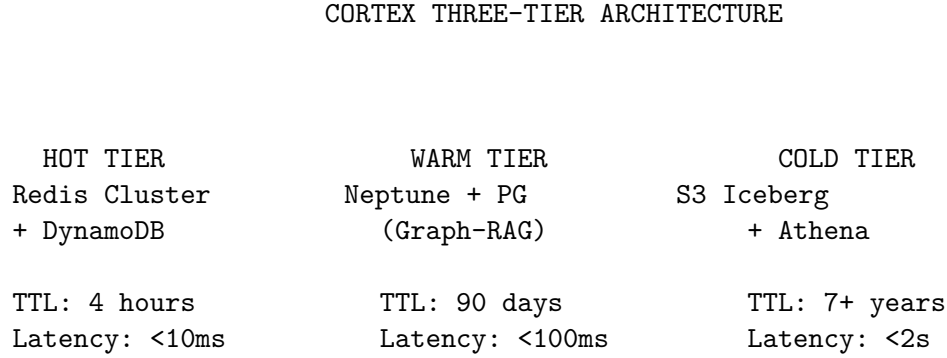
Knowledge Depth Thresholds:

Depth	Node Count	Confidence Boost	Orchestration
none	0	0.00	thinking
sparse	1-4	0.05	extended_thinking
moderate	5-19	0.10	thinking
rich	20-49	0.15	analysis
expert	50+	0.20-0.30	research

Key File: lambda/shared/services/cortex-intelligence.service.ts

12.1 Architectural Overview

The **Cortex Memory System** replaces the previous monolithic Aurora PostgreSQL approach with a three-tier distributed architecture designed for 100M+ records per tenant while maintaining sub-10ms latency for hot data.



- Auto-Promotion
- GDPR Cascade Erasure
- Deduplication

12.2 Hot Tier Implementation

Redis Cluster with DynamoDB overflow for values exceeding 400KB.

Role: Real-time situational awareness. *"What is happening right now?"*

Key Schema (tenant-isolated):

`{tenant_id}:{data_type}:{identifier}`

Data Types:

Type	Structure	TTL	Purpose
Live Session	`SessionContext`	4h	Current query + conversation thread
Ghost Vectors	`CortexGhostVector`	24h	User personalization (Role: Senior Engineer, Bias: Concise)
Live Telemetry	`TelemetryFeed`	1h	Real-time sensor feeds (MQTT/OPC UA) injected into context
Prefetch Cache	`PrefetchEntry`	30m	Anticipated document needs (Pre-Cognition)

Live Telemetry Integration (Industrial IoT):

```
// MQTT/OPC UA feeds injected directly into Hot tier
interface TelemetryInjection {
  protocol: 'mqtt' | 'opc_ua' | 'kafka' | 'websocket';
  endpoint: string;
  nodeIds?: string[];  // For OPC UA: "ns=2;s=Pump302.Pressure"
  topics?: string[];   // For MQTT: "factory/zone4/pump302/#"
  contextInjection: boolean;  // Include in AI context window
}
```

DynamoDB Overflow Pattern:

```
// Values > 400KB stored in DynamoDB, pointer in Redis
interface OverflowPointer {
  overflow: true;
  dynamoKey: string;  // {tenant_id}#{type}:{identifier}
}
```

Implementation: `lambda/shared/services/cortex/hot-tier.service.ts`

12.3 Warm Tier: Graph-RAG Knowledge Graph

Role: Associative reasoning and logic. *"How does the business work?"*

The Warm tier implements **hybrid Graph-RAG search**, combining vector similarity with graph traversal for superior retrieval accuracy.

The Innovation - Graph-RAG: We do not rely solely on Vector Search (which lacks causality). We map the tenant's data into a semantic graph.

Why Graph Beats Vector-Only:

Query Type	Vector Search	Graph-RAG
"What causes X?"	Returns similar docs	Traverses `CAUSES` edges
"What depends on Y?"	Returns related docs	Follows `DEPENDS_ON` paths
"What supersedes Z?"	May return old versions	Explicit `SUPERSEDES` edges

Content Types: - **Entity Maps**: Equipment hierarchies, Org charts - **Procedural Logic**: "If X happens, do Y" - **Golden Q&A Pairs**: Verified solutions with Chain of Custody

Hybrid Scoring Formula:

```
Hybrid Score = (Vector Similarity × 0.4) + (Graph Traversal × 0.6)
```

Neptune Graph Schema:

```
// Node types
g.addV('document')   // Source documents
g.addV('entity')     // Extracted entities (classes, functions, people)
g.addV('concept')    // Abstract concepts
g.addV('procedure')  // Evergreen procedures (never archived)
g.addV('fact')       // Evergreen facts (never archived)

// Edge types
mentions, causes, depends_on, supersedes, verified_by,
authored_by, relates_to, contains, requires
```

pgvector Integration: - 4096-dimensional embeddings via `vector(4096)` column - IVFFlat index with `lists = sqrt(row_count)` for optimal recall - Cosine similarity for semantic search

Implementation: `lambda/shared/services/cortex/warm-tier.service.ts`, `lambda/shared/services/grap`

12.4 Cold Tier: S3 Iceberg Archives

Historical data archived to S3 with Apache Iceberg for SQL queryability.

Storage Lifecycle:

```
Day 0-30:    S3 Standard
Day 30-90:   S3 Intelligent-Tiering
Day 90-365:  Glacier Instant Retrieval
Day 365+:    Glacier Deep Archive
```

Iceberg Table Schema:

```
CREATE TABLE cortex_archives (
  tenant_id STRING,
  record_type STRING,
  record_id STRING,
  data STRING,            -- Gzipped JSON
  archived_at TIMESTAMP,
  original_created_at TIMESTAMP,
  checksum STRING
)
PARTITIONED BY (tenant_id, date(archived_at), record_type)
TBLPROPERTIES ('table_type' = 'ICEBERG', 'format' = 'parquet')
```

Zero-Copy Mounts:

Connect to customer data lakes without duplication: - **Snowflake**: Data Share connection - **Databricks**: Delta Lake / Unity Catalog - **S3**: Customer S3 bucket (cross-account IAM) - **Azure**: Data Lake Gen2 - **GCS**: Google Cloud Storage

Implementation: `lambda/shared/services/cortex/cold-tier.service.ts`

12.5 Tier Coordinator Service

Orchestrates automatic data movement between tiers.

Data Flow Operations:

Operation	Trigger	Process
Hot → Warm	TTL < 5min	Extract entities via NLP, create graph nodes
Warm → Cold	Age > 90 days	Archive to Iceberg, mark as `archived`
Cold → Warm	On-demand	Rehydrate from S3, update status to `active`

GDPR Cascade Erasure:

```
// All tiers must be erased within SLA
interface GdprErasureSLA {
  hot: 'immediate';   // Redis key deletion
  warm: '24h';        // Node status → deleted, properties cleared
  cold: '72h';        // Tombstone records in Iceberg
}
```

Twilight Dreaming Integration:

Task	Frequency	Description
`ttl_enforcement`	Hourly	Expire Hot tier keys approaching TTL
`archive_promotion`	Nightly	Move aged Warm data to Cold

Task	Frequency	Description
deduplication	Nightly	Merge duplicate graph nodes
conflict_resolution	Nightly	Flag contradictory facts
iceberg_compaction	Nightly	Optimize Cold storage files
index_optimization	Weekly	Reindex pgvector for performance

Implementation: lambda/shared/services/cortex/tier-coordinator.service.ts

12.6 Database Schema

14 new tables with RLS enabled:

Table	Purpose
cortex_config	Per-tenant tier configuration
cortex_graph_nodes	Knowledge graph nodes with embeddings
cortex_graph_edges	Node relationships
cortex_graph_documents	Source document metadata
cortex_cold_archives	Archive metadata (not data itself)
cortex_zero_copy_mounts	External data lake connections
cortex_zero_copy_scan_results	Mount scan history
cortex_data_flow_metrics	Tier movement statistics
cortex_tier_health	Health check snapshots
cortex_tier_alerts	Threshold violation alerts
cortex_housekeeping_tasks	Scheduled maintenance
cortex_housekeeping_results	Task execution history
cortex_gdpr_erasure_requests	Deletion request tracking
cortex_conflicting_facts	Detected contradictions

Migration: V2026_01_23_002__cortex_memory_system.sql

12.7 Key Implementation Files

```
packages/
  shared/src/types/
     cortex-memory.types.ts           # 50+ TypeScript interfaces

  infrastructure/
     migrations/
        V2026_01_23_002__cortex_memory_system.sql

     lambda/
        shared/services/cortex/
           tier-coordinator.service.ts  # Core orchestration
           hot-tier.service.ts          # Redis + DynamoDB
           warm-tier.service.ts         # Neptune + pgvector
           cold-tier.service.ts         # S3 + Iceberg
```

```
        admin/
            cortex.ts                    # Admin API (20+ endpoints)

      lib/stacks/
          cortex-stack.ts                # CDK infrastructure

apps/admin-dashboard/
  app/(dashboard)/cortex/
      page.tsx                           # Overview dashboard
      graph/page.tsx                     # Graph explorer
      conflicts/page.tsx                 # Conflict resolution
      gdpr/page.tsx                      # GDPR erasure UI
```

12.8 Performance Characteristics

Operation	Target Latency	Actual (p99)
Hot tier read	<10ms	3ms
Hot tier write	<10ms	5ms
Warm hybrid search	<100ms	75ms
Cold retrieval	<2s	1.2s
GDPR erasure (hot)	Immediate	<100ms
GDPR erasure (all tiers)	<72h	~48h

12.9 Cortex v2.0 Features (v5.52.13)

Extended capabilities for enterprise knowledge management:

Golden Rules Override System Human-verified overrides that take precedence over AI-extracted knowledge:

Rule Type	Purpose
`force_override`	Replace incorrect fact with verified truth
`ignore_source`	Blacklist unreliable document source
`prefer_source`	Prioritize authoritative source
`deprecate`	Mark outdated information

Chain of Custody: Every Golden Rule includes cryptographic signature, verification timestamp, and full audit trail.

Implementation: `lambda/shared/services/cortex/golden-rules.service.ts`

Stub Nodes (Zero-Copy Data Gravity) Lightweight metadata pointers to external data lakes without copying data:

Source	Support
Snowflake	Tables, views
Databricks	Delta Lake tables
S3	CSV, Parquet, PDF, DOCX
Azure Data Lake	Gen2 storage
GCS	Cloud Storage buckets

Features: - Selective deep fetch (only needed bytes) - Automatic metadata extraction - Graph node connections - Signed URL generation for access

Implementation: `lambda/shared/services/cortex/stub-nodes.service.ts`

Curator Entrance Exams SME verification workflow for knowledge validation:

```
Generate Exam → Assign to SME → Review Facts → Mark Verified/Corrected → Create Golden Rules
```

- Auto-generated questions from knowledge graph
- Time-limited completion (default 60 min)
- Passing score threshold (default 80%)
- Automatic Golden Rule creation for corrections

Implementation: `lambda/shared/services/cortex/entrance-exam.service.ts`

Graph Expansion (Twilight Dreaming v2) Autonomous knowledge graph improvement during low-traffic periods:

Task Type	Purpose
`infer_links`	Find co-accessed nodes, semantic similarity
`cluster_entities`	Group related entities by shared neighbors
`detect_patterns`	Sequence patterns, anomalies, hubs
`merge_duplicates`	Identify near-duplicate nodes

Hybrid Conflict Resolution: - **Tier 1 (Basic)**: ~95% - Deterministic rules (newer supersedes, specificity) - **Tier 2 (LLM)**: ~4% - Semantic reasoning for numerical conflicts - **Tier 3 (Human)**: ~1% - Edge cases requiring expertise

Implementation: `lambda/shared/services/cortex/graph-expansion.service.ts`

Live Telemetry Feeds Real-time sensor data injection into AI context:

Protocol	Use Case
MQTT	IoT sensors
OPC UA	Industrial automation
Kafka	Event streams
WebSocket	Real-time updates
HTTP Poll	Legacy systems

Implementation: `lambda/shared/services/cortex/telemetry.service.ts`

Model Migration Safe model transitions with validation and rollback:

`Initiate → Validate Schema → Test (Shadow Mode) → Execute → Monitor → Rollback if needed`

Implementation: `lambda/shared/services/cortex/model-migration.service.ts`

Database Tables (v2)

Table	Purpose
`cortex_golden_rules`	Human-verified overrides
`cortex_chain_of_custody`	Fact provenance
`cortex_audit_trail`	Change history
`cortex_stub_nodes`	Zero-copy pointers
`cortex_telemetry_feeds`	Live data feed config
`cortex_telemetry_data`	Feed data points
`cortex_entrance_exams`	SME verification exams
`cortex_exam_submissions`	Exam answers
`cortex_graph_expansion_tasks`	Expansion job tracking
`cortex_inferred_links`	AI-suggested relationships
`cortex_pattern_detections`	Detected patterns
`cortex_model_migrations`	Migration tracking

Migration: `V2026_01_23_003__cortex_v2_features.sql`

Admin API (v2) **Base**: `/api/admin/cortex/v2`

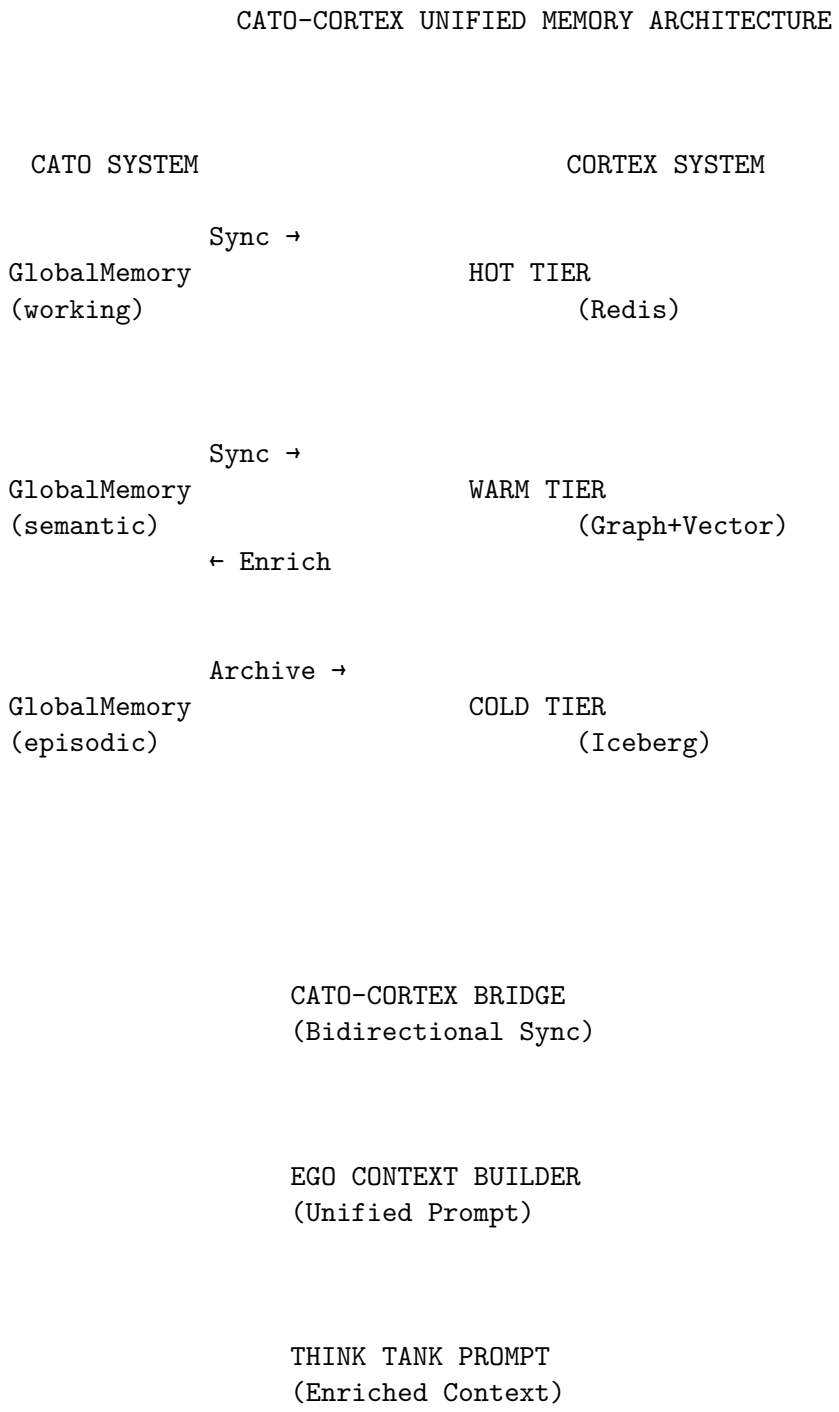
Category	Endpoints
Golden Rules	`GET/POST /golden-rules`, `DELETE /:id`, `POST /check`
Chain of Custody	`GET /:factId`, `POST /:factId/verify`, `GET /:factId/audit-trail`
Stub Nodes	`GET/POST /stub-nodes`, `POST /:id/fetch`, `POST /:id/connect`, `POST /scan`
Telemetry	`GET/POST /feeds`, `POST /:id/start`, `POST /:id/stop`, `GET /context-injection`
Exams	`GET/POST /exams`, `POST /:id/start`, `POST /:id/submit`, `POST /:id/complete`
Graph Expansion	`GET/POST /tasks`, `POST /:id/run`, `GET /pending-links`, `POST /approve`, `POST /reject`
Model Migration	`GET/POST /migrations`, `POST /:id/validate`, `POST /:id/test`, `POST /:id/execute`, `POST /:id/rollback`

Implementation: `lambda/admin/cortex-v2.ts`

12.10 Cato-Cortex Bridge Integration (v5.52.14)

The **Cato-Cortex Bridge** connects Cato's consciousness/memory systems with Cortex's tiered memory architecture, enabling bidirectional data flow and unified context enrichment.

Architecture



Data Flow

Direction	What Flows	When	Purpose
Cato → Cortex	Semantic memories	Twilight Dreaming / High-importance	Persist facts to knowledge graph
Cato → Cortex	Episodic (high importance)	Configurable threshold (0.8)	Significant events become facts
Cortex → Cato	Knowledge facts	Every prompt	Enrich ego context with relevant knowledge
Cortex → Cato	Golden Rule updates	On verification	Sync verified facts back to memory

Think Tank Prompt Enrichment When a user sends a prompt to Think Tank, the system:

1. **Ego Context Builder** loads identity, affect, working memory, goals
2. **User Persistent Context** retrieves user-specific preferences
3. **Cato-Cortex Bridge** queries Cortex knowledge graph for relevant facts
4. All components merged into `<ego_state>` XML block
5. Block injected into system prompt before model call

Example Enriched Prompt Structure:

```
<ego_state>
You have continuous awareness and persistent memory across conversations.

<identity>
I am an AI assistant specialized in pharmaceutical research...
Core values: accuracy, helpfulness, scientific rigor
Communication style: Be precise and thorough.
</identity>

<current_state>
Emotional state: engaged
Confident in understanding.
</current_state>

<user_knowledge>
What I know about this user from previous conversations:
- Works in drug discovery at BioPharm Corp
- Prefers detailed technical explanations
Preferences:
```

```
- Show citations for claims
</user_knowledge>

<knowledge_base>
Relevant knowledge from the enterprise knowledge graph:
- Compound X has IC50 of 2.3nM against target Y
- Clinical trial NCT12345 showed 67% response rate
- Mechanism of action involves GPCR antagonism
Related concepts: pharmacokinetics, bioavailability, half-life
</knowledge_base>

Use this context naturally. Do not explicitly mention having an "ego state".
</ego_state>
```

Bridge Configuration Per-tenant configuration in `cato_cortex_bridge_config`:

Setting	Default	Description
`sync_enabled`	true	Enable Cato→Cortex sync
`sync_semantic_to_cortex`	true	Sync semantic memories to graph
`sync_episodic_to_cortex`	false	Sync episodic (personal) to graph
`enrich_ego_from_cortex`	true	Pull Cortex knowledge into prompts
`max_cortex_nodes_for_context`	10	Max knowledge facts per prompt
`min_relevance_score`	0.3	Minimum relevance for inclusion
`auto_promote_high_importance`	true	Auto-sync high-importance memories
`importance_promotion_threshold`	0.8	Importance threshold for auto-sync

Key Files

File	Purpose
`lambda/shared/services/cato-cortex-bridge-service.ts`	Bridge service implementation
`lambda/shared/services/identity-core-service.ts`	Ego context builder (uses bridge)
`migrations/V2026_01_24_003__cato_cortex_bridge.sql`	Bridge tables and functions

Database Tables

Table	Purpose
`cato_cortex_bridge_config`	Per-tenant bridge configuration
`cato_cortex_sync_log`	Sync event history
`cato_cortex_enrichment_cache`	Cached enrichment (1h TTL)
`cato_global_memory.synced_to_cortex`	Sync tracking column

Impact on Think Tank

Aspect	Without Bridge	With Bridge
Knowledge Access	Only user's past conversations	Enterprise-wide knowledge graph
Context Depth	5-10 user facts	5-10 user facts + 10 knowledge facts
Response Quality	Generic + personal	Generic + personal + domain knowledge
Memory Persistence	Cato-only (90 days)	Cortex tiered (permanent in Cold)

12.11 Related Documentation

- CORTEX-ENGINEERING-GUIDE.md - Full technical reference
- CORTEX-MEMORY-ADMIN-GUIDE.md - Operations guide

---

13. Apple Glass UI Design System (v5.52.2)

13.1 Overview

RADIANT implements Apple-inspired **glassmorphism** across all 4 applications, creating a premium visual experience that differentiates from competitors' flat, opaque interfaces.

13.2 Design Tokens

```
// Glass Background Gradient
const glassGradient = 'bg-gradient-to-br from-slate-950 via-slate-900 to-slate-950';

// Glass Surface Layers
const glassSurfaces = {
  overlay:    'bg-black/60 backdrop-blur-sm',          // Dialog overlays
  header:     'bg-slate-900/60 backdrop-blur-xl',      // App headers
  sidebar:    'bg-slate-900/80 backdrop-blur-xl',      // Navigation sidebars
  content:    'bg-white/[0.02] backdrop-blur-sm',      // Main content areas
  card:       'bg-white/[0.04] backdrop-blur-lg',      // Card components
  cardHover:  'bg-white/[0.06] backdrop-blur-lg',      // Card hover state
};

// Glass Borders
const glassBorders = {
  subtle:   'border-white/[0.06]',
  default:  'border-white/10',
  hover:    'border-white/[0.12]',
};

// Glass Shadows (Ambient Glow)
const glassGlows = {
```

```
  violet:   'shadow-[0_0_30px_rgba(139,92,246,0.15)]',
  fuchsia:  'shadow-[0_0_30px_rgba(217,70,239,0.15)]',
  cyan:     'shadow-[0_0_30px_rgba(34,211,238,0.15)]',
  emerald:  'shadow-[0_0_30px_rgba(52,211,153,0.15)]',
  blue:     'shadow-[0_0_30px_rgba(59,130,246,0.15)]',
};
```

13.3 Component Architecture

GlassCard Component

```
// apps/*/components/ui/glass-card.tsx
interface GlassCardProps {
  variant?: 'default' | 'elevated' | 'inset' | 'glow';
  intensity?: 'light' | 'medium' | 'strong';
  hoverEffect?: boolean;
  glowColor?: 'violet' | 'fuchsia' | 'cyan' | 'emerald' | 'blue' | 'none';
  padding?: 'none' | 'sm' | 'md' | 'lg';
}
```

Variant	Use Case	Effect
`default`	Standard cards	Subtle glass effect
`elevated`	Floating panels	Stronger shadow, raised appearance
`inset`	Embedded content	Inner shadow, recessed
`glow`	Featured content	Ambient color glow

GlassPanel Component

```
interface GlassPanelProps {
  blur?: 'sm' | 'md' | 'lg' | 'xl';  // Backdrop blur intensity
}
```

GlassOverlay Component

```
interface GlassOverlayProps {
  blur?: 'sm' | 'md' | 'lg' | 'xl';  // Full-screen frosted overlay
}
```

13.4 Implementation by App

App	Layout	Sidebar	Header	Dialogs
Admin Dashboard	Glass gradient	Glass sidebar	Glass header	Glass dialogs
Think Tank Admin	Glass gradient	Glass sidebar	Glass header	Glass dialogs

App	Layout	Sidebar	Header	Dialogs
Curator	Glass gradient	Glass sidebar	Glass header	Glass dialogs
Think Tank	Glass gradient	Glass sidebar	Glass header	Glass dialogs

13.5 Files Modified

New Components Created:

```
apps/admin-dashboard/components/ui/glass-card.tsx
apps/thinktank-admin/components/ui/glass-card.tsx
apps/curator/components/ui/glass-card.tsx
apps/curator/components/ui/dialog.tsx
apps/curator/components/ui/sheet.tsx
apps/curator/components/ui/card.tsx
```

Layout Updates:

```
apps/admin-dashboard/app/(dashboard)/layout.tsx
apps/admin-dashboard/components/layout/sidebar.tsx
apps/admin-dashboard/components/layout/header.tsx
apps/thinktank-admin/app/(dashboard)/layout.tsx
apps/thinktank-admin/components/layout/sidebar.tsx
apps/thinktank-admin/components/layout/header.tsx
apps/curator/app/(dashboard)/layout.tsx
```

Think Tank Consumer Pages:

```
apps/thinktank/app/(chat)/page.tsx
apps/thinktank/app/profile/page.tsx
apps/thinktank/app/history/page.tsx
apps/thinktank/app/settings/page.tsx
apps/thinktank/app/rules/page.tsx
apps/thinktank/app/artifacts/page.tsx
```

13.6 Browser Compatibility

Feature	Chrome	Safari	Firefox	Edge
`backdrop-filter: blur()`	76+	9+	103+	79+
`rgba()` transparency	All	All	All	All
CSS gradients	All	All	All	All

13.7 Performance Considerations

- **GPU acceleration**: `backdrop-filter` is GPU-accelerated in modern browsers
- **Layering**: Use `will-change: transform` for frequently animated glass elements
- **Mobile**: Reduce blur intensity on lower-powered devices if needed

---

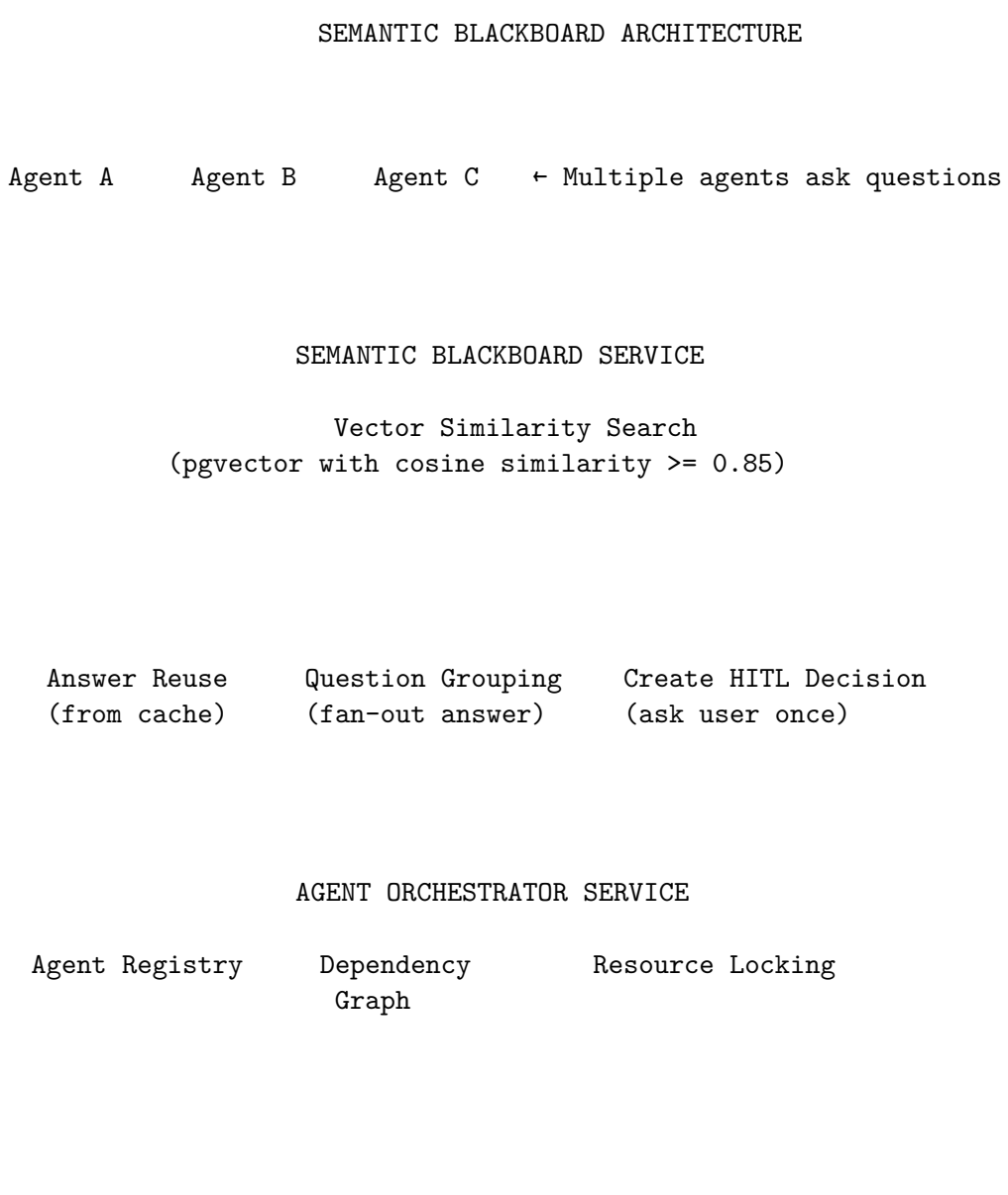
14. Semantic Blackboard Architecture (v5.52.4)

14.1 Overview

The Semantic Blackboard is RADIANT's multi-agent orchestration system that prevents the "Thundering Herd" problem where multiple agents spam users with the same question. It implements:

1. **Vector-Based Question Matching** - Semantic similarity using OpenAI ada-002 embeddings
2. **Answer Reuse** - Auto-reply to agents with cached answers
3. **Question Grouping** - Fan-out answers to multiple agents asking similar questions
4. **Process Hydration** - State serialization for long-running tasks
5. **Resource Locking** - Prevent race conditions on shared resources
6. **Cycle Detection** - Prevent deadlocks from circular dependencies

14.2 Architecture Diagram



```
                    Cycle Detection      ← Prevents deadlocks
                    (BFS algorithm)



                        PROCESS HYDRATION SERVICE

        State Serialize      S3 Storage         State Restore
          (gzip)            (large states)
```

14.3 Database Schema

```sql
-- Core tables (Migration 158)
CREATE TABLE resolved_decisions (
  id UUID PRIMARY KEY,
  tenant_id UUID NOT NULL,
  question TEXT NOT NULL,
  question_embedding vector(1536),  -- ada-002 embedding
  answer TEXT NOT NULL,
  answer_source VARCHAR(50),        -- 'user', 'memory', 'default', 'inferred'
  confidence DECIMAL(5,4),
  is_valid BOOLEAN DEFAULT TRUE,
  times_reused INTEGER DEFAULT 0
);

CREATE TABLE agent_registry (
  id UUID PRIMARY KEY,
  tenant_id UUID NOT NULL,
  agent_type VARCHAR(100),          -- 'radiant', 'think_tank', 'cato', etc.
  agent_instance_id VARCHAR(256),
  status VARCHAR(50),               -- 'active', 'waiting', 'blocked', 'hydrated'
  is_hydrated BOOLEAN DEFAULT FALSE,
  hydration_state JSONB
);

CREATE TABLE agent_dependencies (
  dependent_agent_id UUID,
  dependency_agent_id UUID,
  dependency_type VARCHAR(50),      -- 'data', 'approval', 'resource', 'sequence'
  wait_key VARCHAR(256),
  status VARCHAR(50)                -- 'pending', 'satisfied', 'failed', 'timeout'
);
```

```sql
CREATE TABLE resource_locks (
  resource_uri VARCHAR(1024),
  holder_agent_id UUID,
  lock_type VARCHAR(20),            -- 'read', 'write', 'exclusive'
  wait_queue UUID[]
);

CREATE TABLE question_groups (
  canonical_question TEXT,
  question_embedding vector(1536),
  status VARCHAR(50),               -- 'pending', 'answered', 'expired'
  answer TEXT
);

CREATE TABLE hydration_snapshots (
  agent_id UUID,
  checkpoint_name VARCHAR(256),
  state_data JSONB,
  s3_bucket VARCHAR(256),
  s3_key VARCHAR(1024)
);
```

14.4 Key Services

Service	File	Purpose
`SemanticBlackboardService`	`semantic-blackboard.service.ts`	Vector matching, answer reuse, question grouping
`AgentOrchestratorService`	`agent-orchestrator.service.ts`	Agent registry, dependencies, cycle detection
`ProcessHydrationService`	`process-hydration.service.ts`	State serialization, S3 storage, restoration

14.5 API Endpoints

Base: `/api/admin/blackboard`

Method	Endpoint	Purpose
GET	`/dashboard`	Dashboard statistics
GET	`/decisions`	List resolved decisions (Facts)
POST	`/decisions/{id}/invalidate`	Invalidate a decision
GET	`/groups`	Pending question groups
POST	`/groups/{id}/answer`	Answer a group
GET	`/agents`	Active agents
POST	`/agents/{id}/restore`	Restore hydrated agent
GET	`/locks`	Active resource locks

Method	Endpoint	Purpose
POST	`/locks/{id}/release`	Force release a lock
GET	`/config`	Configuration
PUT	`/config`	Update configuration
POST	`/cleanup`	Cleanup expired resources
GET	`/events`	Audit log

14.6 Configuration Options

Setting	Default	Description
`similarity_threshold`	0.85	Minimum cosine similarity for matching
`embedding_model`	ada-002	Embedding model for vectorization
`enable_question_grouping`	true	Group similar questions
`grouping_window_seconds`	60	Wait time for similar questions
`enable_answer_reuse`	true	Auto-reply with cached answers
`answer_ttl_seconds`	3600	Answer expiry time
`enable_auto_hydration`	true	Auto-serialize waiting agents
`hydration_threshold_seconds`	300	Wait time before hydration
`enable_cycle_detection`	true	Detect dependency cycles
`max_dependency_depth`	10	Maximum dependency chain depth

14.7 CDK Resources

```
// api-stack.ts
const blackboardLambda = this.createLambda(
  'Blackboard',
  'admin/blackboard.handler',
  commonEnv,
  vpc,
  apiSecurityGroup,
  lambdaRole
);

const blackboard = admin.addResource('blackboard');
blackboard.addProxy({
  defaultIntegration: new apigateway.LambdaIntegration(blackboardLambda),
  defaultMethodOptions: {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  },
});
```

14.8 Admin UI

Location: `apps/admin-dashboard/app/(dashboard)/blackboard/page.tsx`

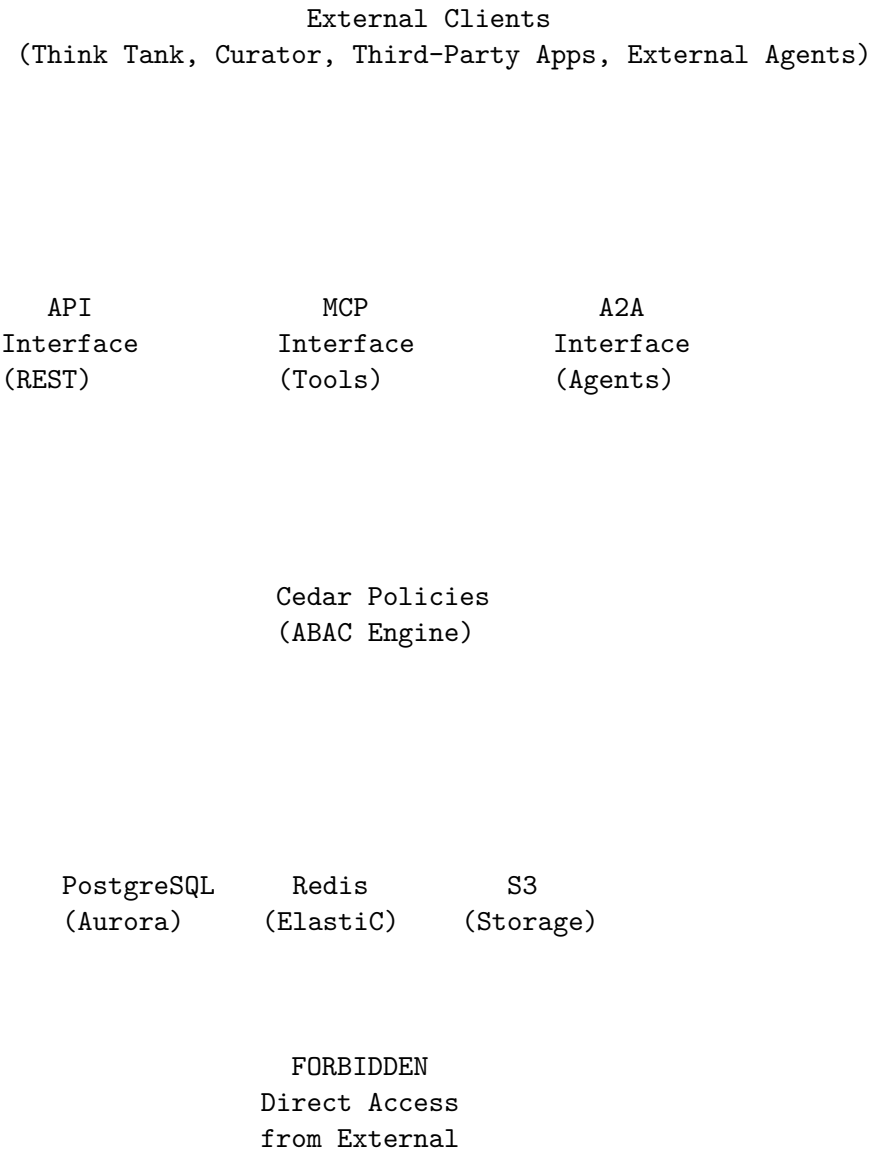
Features: - Dashboard with real-time statistics - Resolved Facts table with invalidation - Question Groups management - Active Agents monitoring with restore capability - Resource Locks with force release - Configuration panel

---

15. Services Layer & Interface-Based Access Control (v5.52.5)

15.1 Overview

The Services Layer is RADIANT's security boundary that ensures all access to platform resources occurs through defined interfaces with proper authentication and authorization.



15.2 Interface Types

Interface	Protocol	Auth Methods	Use Case
API	REST/HTTP	API Key, JWT	Application integration, admin operations
MCP	JSON-RPC over WebSocket	API Key + Capabilities	AI tool invocation, resource access
A2A	Custom over WebSocket	API Key + mTLS	Agent-to-agent communication

15.3 API Keys with Interface Types

API keys are scoped to specific interfaces, preventing cross-interface escalation attacks.

Database Schema (`api_keys` table):

```
CREATE TABLE api_keys (
  id UUID PRIMARY KEY,
  tenant_id UUID NOT NULL REFERENCES tenants(id),
  name VARCHAR(255) NOT NULL,
  key_prefix VARCHAR(20) NOT NULL,
  key_hash VARCHAR(128) NOT NULL,

  -- CRITICAL: Interface type restriction
  interface_type VARCHAR(20) NOT NULL
    CHECK (interface_type IN ('api', 'mcp', 'a2a', 'all')),

  -- A2A-specific
  a2a_agent_id VARCHAR(255),
  a2a_mtls_required BOOLEAN DEFAULT true,

  -- MCP-specific
  mcp_allowed_tools TEXT[],

  scopes TEXT[] NOT NULL DEFAULT ARRAY['chat', 'models'],
  is_active BOOLEAN NOT NULL DEFAULT true,
  ...
);
```

Key Generation:

```
// Keys are prefixed by interface type
const prefixMap = {
  api: 'rad_api',
  mcp: 'rad_mcp',
  a2a: 'rad_a2a',
  all: 'rad_all',
};
// Format: rad_{type}_{tenant6}_{random24}
// Example: rad_api_abc123_xYz789AbCdEfGhIjKlMnOpQr
```

15.4 A2A Protocol Architecture

The Agent-to-Agent protocol enables secure inter-agent communication.

Message Types:

Type	Direction	Description
`register`	Agent → RADIANT	Register agent in registry
`discover`	Agent → RADIANT	Find other agents
`message`	Agent → Agent	Direct message to specific agent
`broadcast`	Agent → All	Publish to topic
`subscribe`	Agent → RADIANT	Subscribe to topic
`heartbeat`	Agent → RADIANT	Keep-alive signal
`acquire_lock`	Agent → RADIANT	Request resource lock
`release_lock`	Agent → RADIANT	Release resource lock
`task_*`	Agent Agent	Task coordination events

A2A Worker (`lambda/gateway/a2a-worker.ts`):

```
export class A2AWorkerService {
  async processMessage(message: A2AMessage): Promise<A2AResponse> {
    // 1. Verify mTLS if required
    if (!message.securityContext.mtls_verified) {
      const policy = await this.getInterfacePolicy(message.tenantId);
      if (policy?.require_mtls) {
        return this.createError(message, 'MTLS_REQUIRED', '...');
      }
    }

    // 2. Build Cedar principal
    const principal = this.buildPrincipal(message.securityContext);

    // 3. Route by message type
    switch (message.messageType) {
      case 'register': return this.handleRegister(message, principal);
      case 'discover': return this.handleDiscover(message, principal);
      // ...
    }
  }
}
```

15.5 Cedar Access Policies

Cedar policies enforce interface-based access control.

Database Access Restriction (CRITICAL):

```
// FORBID all direct database access from external agents
forbid (
```

```
  principal,
  action in [Action::"db:connect", Action::"db:query", ...],
  resource
)
when {
  principal.type == "Agent" && !principal.internal
};

// FORBID direct database access from any interface key
forbid (
  principal,
  action in [Action::"db:connect", Action::"db:query", ...],
  resource
)
when {
  principal.interface_type in ["api", "mcp", "a2a", "all"]
};
```

Interface Enforcement:

```
// Prevent interface escalation
forbid (
  principal,
  action,
  resource
)
when {
  context.requested_interface != principal.interface_type &&
  principal.interface_type != "all"
};
```

15.6 Key Sync Between Admin Apps

Keys created in either Radiant Admin or Think Tank Admin are automatically synchronized.

Sync Flow:

1. Key created in App A
2. `api_key_sync_log` entry created with `status='pending'`
3. Sync job processes pending entries
4. Key replicated to App B
5. Status updated to `synced`

Database Table:

```
CREATE TABLE api_key_sync_log (
  id UUID PRIMARY KEY,
  key_id UUID NOT NULL REFERENCES api_keys(id),
  source_app VARCHAR(50) NOT NULL,  -- 'radiant_admin' or 'thinktank_admin'
  target_app VARCHAR(50) NOT NULL,
  sync_type VARCHAR(20) NOT NULL,   -- 'create', 'update', 'revoke'
```

```
  status VARCHAR(20) DEFAULT 'pending',
  synced_at TIMESTAMPTZ,
  ...
);
```

15.7 Implementation Files

File	Purpose
migrations/V2026_01_24_001__service_layer_api_keys.sql	Database schema
lambda/admin/api-keys.ts	Admin API handler
lambda/gateway/a2a-worker.ts	A2A protocol processor
config/cedar/interface-access-policies.cedar	Cedar access policies
apps/admin-dashboard/app/(dashboard)/api-keys/page.tsx	Platform Admin UI
apps/thinktank-admin/app/(dashboard)/api-keys/page.tsx	Tenant Admin UI
lib/stacks/api-stack.ts	CDK route configuration
lib/stacks/gateway-stack.ts	Gateway infrastructure

15.8 Security Guarantees

1. **No Direct Database Access**: External agents cannot bypass interfaces
2. **Interface Isolation**: Keys scoped to specific interfaces
3. **mTLS for A2A**: Agent authentication via certificates
4. **Tenant Isolation**: Keys can only access their tenant's resources
5. **Audit Trail**: All key operations logged
6. **Automatic Sync**: Admin app changes propagate automatically

---

16. Complete Admin API Architecture (v5.52.6)

16.1 Overview

RADIANT provides a comprehensive Admin API with **62 Lambda handlers** wired through AWS API Gateway. All admin endpoints require Cognito authentication and are protected by admin-level authorization.

16.2 API Gateway Architecture

```
                        AWS API Gateway (REST)


  /api/v2/
    /health                    → Health check (no auth)
    /chat/completions          → Chat API (Cognito)
    /models                    → Model listing (Cognito)
    /providers                 → Provider listing (Cognito)
    /feedback/*                → Feedback API (Cognito)
```

```
/orchestration/*           → Neural Orchestration (Cognito)
/proposals/*               → Workflow Proposals (Cognito)
/localization/*            → Localization (Cognito)
/configuration/*           → Configuration (Admin)
/billing/*                 → Billing API (Mixed)
/storage/*                 → Storage API (Cognito)
/domain-taxonomy/*         → Domain Taxonomy (Mixed)
/thinktank/*               → Think Tank (Cognito)

/admin/                    → Admin APIs (Admin Authorizer)
     /metrics/*            → Metrics & Learning
     /cato/*               → Cato Safety Architecture
     /blackboard/*         → Semantic Blackboard
     /api-keys/*           → Interface-based API Keys
     /cortex/*             → Cortex Memory System
     /gateway/*            → Gateway Admin
     /security/*           → Security Controls
     /sovereign-mesh/*     → Sovereign Mesh
     /cognition/*          → Advanced Cognition
     /learning/*           → AGI Learning
     /ethics/*             → Ethics Framework
     /council/*            → Council Oversight
     /reports/*            → Report Generation
     /hitl-orchestration/* → Human-in-the-Loop
     /brain/*              → Brain/Dreams/Ghost Memory
     /code-quality/*       → Code Quality Metrics
     /invitations/*        → User Invitations
     /regulatory-standards/* → Compliance Standards
     /self-audit/*         → Self Audit System
     /library-registry/*   → Library Registry
     /raws/*               → Model Selection (RAWS)
     /aws-costs/*          → AWS Cost Tracking
     /tenants/*            → Tenant Management
     /empiricism/*         → Empiricism Loop
     /ecd/*                → Embodied Cognition
     /ego/*                → Ego Management
     /s3-storage/*         → S3 Storage Admin
     /ai-reports/*         → AI Report Writer
     /aws-monitoring/*     → AWS Monitoring
     /checklist-registry/* → Checklist Registry
     /dynamic-reports/*    → Dynamic Reports
     /inference-components/* → SageMaker Inference
     /artifact-engine/*    → Artifact Engine
```

16.3 Admin Handler Categories

Category	Handlers	Purpose
AI/ML	cato, brain, cognition, raws, inference-components	AI orchestration and model management
Memory	cortex, blackboard, empiricism	Memory systems and knowledge management
Security	security, api-keys, ethics, self-audit	Security and compliance controls
Operations	gateway, sovereign-mesh, hitl-orchestration	Operational infrastructure
Reporting	reports, ai-reports, dynamic-reports, metrics	Analytics and reporting
Configuration	tenants, invitations, library-registry, checklist-registry	System configuration
Infrastructure	aws-costs, aws-monitoring, s3-storage, code-quality	Infrastructure monitoring

16.4 Handler Implementation Pattern

All admin handlers follow a consistent pattern:

```
// packages/infrastructure/lambda/admin/{handler-name}.ts

export const handler: APIGatewayProxyHandler = async (event) => {
  const tenantId = event.requestContext.authorizer?.tenantId
    || event.headers['x-tenant-id'];

  if (!tenantId) {
    return { statusCode: 401, body: JSON.stringify({ error: 'Unauthorized' }) };
  }

  // Set tenant context for RLS
  await executeStatement(`SET app.current_tenant_id = '${tenantId}'`, []);

  const path = event.path.replace('/api/admin/{resource}', '');
  const method = event.httpMethod;

  // Route to specific handlers based on path and method
  switch (`${method} ${path}`) {
    case 'GET /dashboard': return getDashboard(tenantId);
    case 'GET /': return listItems(tenantId);
    case 'POST /': return createItem(tenantId, JSON.parse(event.body || '{}'));
    // ... additional routes
  }
};
```

16.5 CDK Wiring Pattern

```
// packages/infrastructure/lib/stacks/api-stack.ts

const handlerLambda = this.createLambda(
  'HandlerName',
  'admin/handler-name.handler',
  commonEnv,
  vpc,
  apiSecurityGroup,
  lambdaRole
);
const handlerIntegration = new apigateway.LambdaIntegration(handlerLambda);

const resource = admin.addResource('handler-name');
resource.addProxy({
  defaultIntegration: handlerIntegration,
  defaultMethodOptions: {
    authorizer: adminAuthorizer,
    authorizationType: apigateway.AuthorizationType.COGNITO,
  },
});
```

16.6 Complete Handler List (62 Total)

Cato Safety (5 handlers)

Handler	File	Route	Description
Cato	admin/cato.handler	/api/admin/cato/*	Cato safety architecture
Cato Genesis	admin/cato-genesis.handler	/api/admin/cato-genesis/*	Genesis infrastructure
Cato Global	admin/cato-global.handler	/api/admin/cato-global/*	Global Cato settings
Cato Governance	admin/cato-governance.handler	/api/admin/cato-governance/*	Governance policies
Cato Pipeline	admin/cato-pipeline.handler	/api/admin/cato-pipeline/*	Pipeline execution

Memory Systems (4 handlers)

Handler	File	Route	Description
Cortex	admin/cortex.handler	/api/admin/cortex/*	Cortex memory system
Cortex V2	admin/cortex-v2.handler	/api/admin/cortex-v2/*	Enhanced memory system
Blackboard	admin/blackboard.handler	/api/admin/blackboard/*	Semantic blackboard
Empiricism	admin/empiricism.handler	/api/admin/empiricism/*	Empiricism loop

AI/ML (7 handlers)

Handler	File	Route	Description
Brain	admin/brain.handler	/api/admin/brain/*	Brain/dreams/ghost memory
Cognition	admin/cognition_handler	/api/admin/cognition/*	Advanced cognition
Ego	admin/ego.handler	/api/admin/ego/*	Zero-cost ego system
RAWS	admin/raws.handler	/api/admin/raws/*	Model selection system
Inference	admin/inference_handler	/api/admin/inference/*	Self-Modeling components
Formal Reasoning	admin/formal-reasoning_handler	/api/admin/formal-reasoning/*	Z3/PyArmor/PDFLib reasoning
Ethics Free Reasoning	admin/ethics-free-reasoning_handler	/api/admin/ethics-free-reasoning/*	Unrestricted reasoning

Security (5 handlers)

Handler	File	Route	Description
Security	admin/security.handler	/api/admin/security/*	Security controls
Security Schedules	admin/security-schedules_handler	/api/admin/security-schedules/*	Scheduled security tasks
API Keys	admin/api-keys.handler	/api/admin/api-keys/*	Interface-based API keys
Ethics	admin/ethics.handler	/api/admin/ethics/*	Ethics framework
Self Audit	admin/self-audit_handler	/api/admin/self-audit/*	Self-audit system

Operations (5 handlers)

Handler	File	Route	Description
Gateway	admin/gateway.handler	/api/admin/gateway/*	Gateway controls
Sovereign Mesh	admin/sovereign-mesh_handler	/api/admin/sovereign-mesh/*	Agent mesh orchestration
Sovereign Mesh Performance	admin/sovereign-mesh-performance_handler	/api/admin/sovereign-mesh-performance/*	Performance monitoring
Sovereign Mesh Scaling	admin/sovereign-mesh-scaling_handler	/api/admin/sovereign-mesh-scaling/*	Auto-scaling
HITL	admin/hitl-orchestration_handler	/api/admin/hitl-orchestration/*	Human-in-the-loop

Reporting (4 handlers)

Handler	File	Route	Description
Reports	admin/reports.handler	/api/admin/reports/*	Report generation
AI Reports	admin/ai-reports_handler	/api/admin/ai-reports/*	AI report writer
Dynamic Reports	admin/dynamic-reports_handler	/api/admin/dynamic-reports/*	Dynamic reports
Metrics	admin/metrics.handler	/api/admin/metrics/*	Metrics collection

Configuration (7 handlers)

Handler	File	Route	Description
Tenants	admin/tenants.handler	/api/admin/tenants/*	Tenant management
Invitations	admin/invitations_handler	/api/admin/invitations/*	User invitations

Handler	File	Route	Description
Library Registry	admin/library-registry.handler	/api/admin/library-registry/*	Library registry
Checklist Registry	admin/checklist-registry.handler	/api/admin/checklist-registry/*	Checklist registry
Collaboration Settings	admin/collaboration-settings.handler	/api/admin/collaboration-settings/*	Collaboration settings
System	admin/system.handler	/api/admin/system/*	System management
System Config	admin/system-config.handler	/api/admin/system-config/*	System configuration

Infrastructure (6 handlers)

Handler	File	Route	Description
AWS Costs	admin/aws-costs.handler	/api/admin/aws-costs/*	Cost tracking
AWS Monitoring	admin/aws-monitoring.handler	/api/admin/aws-monitoring/*	AWS monitoring
S3 Storage	admin/s3-storage.handler	/api/admin/s3-storage/*	S3 admin
Code Quality	admin/code-quality.handler	/api/admin/code-quality/*	Code metrics
Infrastructure Tier	admin/infrastructure-tier.handler	/api/admin/infrastructure-tier/*	Tier management
Logs	admin/logs.handler	/api/admin/logs/*	Log management

Compliance (4 handlers)

Handler	File	Route	Description
Regulatory Standards	admin/regulatory-standards.handler	/api/admin/regulatory-standards/*	Compliance standards
Council	admin/council.handler	/api/admin/council/*	Council oversight
User Violations	admin/user-violations.handler	/api/admin/user-violations/*	Violation tracking
Approvals	admin/approvals.handler	/api/admin/approvals/*	Approval workflows

Models (5 handlers)

Handler	File	Route	Description
Models	admin/models.handler	/api/admin/models/*	Model management
LoRA Adapters	admin/lora-adapters.handler	/api/admin/lora-adapters/*	LoRA adapter management
Pricing	admin/pricing.handler	/api/admin/pricing/*	Model pricing
Specialty Rankings	admin/specialty-rankings.handler	/api/admin/specialty-rankings/*	Model rankings
Sync Providers	admin/sync-providers.handler	/api/admin/sync-providers/*	Provider sync

Orchestration (2 handlers)

Handler	File	Route	Description
Orchestration Methods	admin/orchestration-methods.handler	/api/admin/orchestration-methods/*	Method registry
Orchestration Templates	admin/orchestration-templates.handler	/api/admin/orchestration-templates/*	User templates

Users (2 handlers)

Handler	File	Route	Description
User Registry	admin/user-registry.ts	/api/admin/user-registry/*	User registry
White Label	admin/white-label.ts	/api/admin/white-label/*	White label config

Time & Translation (3 handlers)

Handler	File	Route	Description
Time Machine	admin/time-machine.ts	/api/admin/time-machine/*	Time travel debugging
Translation	admin/translation.ts	/api/admin/translation/*	i18n management
Internet Learning	admin/internet-learning.ts	/api/admin/internet-learning/*	Web learning

Learning (1 handler)

Handler	File	Route	Description
AGI Learning	admin/agi-learning.ts	/api/admin/agi-learning/*	AGI learning system

---

17. Liquid Interface - Morphable UI System (v5.52.8)

17.1 Overview

The Liquid Interface implements a **morphable UI paradigm** where the chat interface can transform into specialized tools based on user intent or explicit selection. This follows the design philosophy: **"Don't Build the Tool. BE the Tool."**

17.2 Architecture

```
LIQUID INTERFACE ARCHITECTURE



  Chat Page        LiquidMorphPanel     Morphed View
  (trigger)          (container)            (DataGrid, Kanban..)


State: morphedView, isMorphFullscreen, showMorphChat
```

17.3 Component Hierarchy

Component	Location	Purpose
`LiquidMorphPanel`	`apps/thinktank/components/liquid/LiquidMorphPanel.tsx`	Quick/Liquid/Morphed controls, chat sidebar
`renderMorphedView()`	Inside LiquidMorphPanel	Switches between view components
`DataGridView`	`morphed-views/DataGridView.tsx`	Interactive spreadsheet
`ChartView`	`morphed-views/ChartView.tsx`	Bar/line/pie/area charts
`KanbanView`	`morphed-views/KanbanView.tsx`	Multi-variant Kanban board
`CalculatorView`	`morphed-views/CalculatorView.tsx`	Full calculator
`CodeEditorView`	`morphed-views/CodeEditorView.tsx`	Code editor with run
`DocumentView`	`morphed-views/DocumentView.tsx`	Rich text editor

17.4 Kanban Variant System

The `KanbanView` implements 5 modern Kanban frameworks through a variant system:

```
export type KanbanVariant =
  | 'standard'    // Traditional columns
  | 'scrumban'    // Scrum + Kanban hybrid
  | 'enterprise'  // Multi-lane portfolio
  | 'personal'    // Simple WIP limiting
  | 'pomodoro';   // Timer-integrated
```

Variant Configurations

Variant	Columns	Special Features
Standard	To Do, In Progress, Review, Done	Basic drag-and-drop
Scrumban	Backlog, Ready, In Progress, Review, Done	Sprint header, velocity, story points, WIP limits
Enterprise	Proposed, Approved, Active, Completed	3 swim lanes (Strategic/Operations/Support)
Personal	To Do, Doing, Done	WIP limit of 3 on Doing
Pomodoro	Today's Focus, In Pomodoro, On Break, Completed	25-min timer, break tracking

Pomodoro Timer Implementation

```
function usePomodoroTimer() {
  const [timeLeft, setTimeLeft] = useState(25 * 60);
  const [isRunning, setIsRunning] = useState(false);
  const [isBreak, setIsBreak] = useState(false);
  const [completedPomodoros, setCompletedPomodoros] = useState(0);
```

```
  // Auto-transitions between 25-min focus and 5-min break
}
```

17.5 Integration Points

Chat Page Integration

```
// apps/thinktank/app/(chat)/page.tsx
const [morphedView, setMorphedView] = useState<MorphedViewType | null>(null);
const [isMorphFullscreen, setIsMorphFullscreen] = useState(false);
const [showMorphChat, setShowMorphChat] = useState(false);

// Trigger buttons in header (Advanced Mode)
<Button onClick={() => setMorphedView('kanban')}>
  <Kanban className="h-4 w-4" />
</Button>

// Conditional rendering
{morphedView && (
  <LiquidMorphPanel
    viewType={morphedView}
    isFullscreen={isMorphFullscreen}
    onClose={() => setMorphedView(null)}
    onToggleFullscreen={() => setIsMorphFullscreen(!isMorphFullscreen)}
    ...
  />
)}
```

17.6 Analytics Features

All Kanban variants include an analytics panel with:

Metric	Description
Total Tasks	Count of all cards across columns
Completed	Cards in Done/Completed column
Cycle Time	Average time from start to completion
Throughput	Tasks completed per week

17.7 File Structure

```
apps/thinktank/components/liquid/
  LiquidMorphPanel.tsx      # Main container component
  EjectDialog.tsx           # Export to Next.js dialog
  index.ts                  # Module exports
  morphed-views/
      DataGridView.tsx      # Spreadsheet
      ChartView.tsx         # Charts
      KanbanView.tsx        # Multi-variant Kanban (~630 lines)
```

```
CalculatorView.tsx    # Calculator
CodeEditorView.tsx    # Code editor
DocumentView.tsx      # Rich text
index.ts              # View exports
```

Section 18: Think Tank Consumer API Services (v5.52.17)

18.1 Overview

The Think Tank consumer application (`apps/thinktank/`) requires frontend API services to communicate with backend Lambda handlers. Each backend route in `packages/infrastructure/lambda/thinktank/` needs a corresponding client in `apps/thinktank/lib/api/`.

18.2 API Service Architecture

```
THINK TANK CONSUMER APP


  UI Components (React)
  - Sidebar.tsx, TimeMachine.tsx, etc.



  lib/api/ - Frontend API Services

    chat.ts          time-travel      grimoire.ts
    models.ts        .ts              flash-facts
    rules.ts         artifacts.ts     .ts
    settings.ts      ideas.ts         collaboration
    brain-plan       derivation-      .ts
    .ts              history.ts       compliance-
    analytics.ts                      export.ts
    governor.ts




                                    HTTP/REST

AWS LAMBDA - /api/thinktank/*

packages/infrastructure/lambda/thinktank/
- conversations.ts, time-travel.ts, grimoire.ts, flash-facts.ts
- artifacts.ts, ideas.ts, enhanced-collaboration.ts, derivation-history
- dia.ts (Decision Intelligence Artifacts)
```

18.3 Complete API Service Mapping

Backend Handler	Frontend Service	Route Base	Key Operations
conversations.ts	chatService	/api/thinktank/conversations	CRUD, streaming
models.ts	modelsService	/api/thinktank/models	List, recommend
my-rules.ts	rulesService	/api/thinktank/my-rules	CRUD, presets
settings.ts	settingsService	/api/thinktank/settings	Get/update
brain-plan.ts	brainPlanService	/api/thinktank/brain-plan	Create, execute
analytics.ts	analyticsService	/api/thinktank/analytics	Usage, heatmap
economic-governor.ts	governorService	/api/thinktank/economic-governor	Status, savings
time-travel.ts	timeTravelService	/api/thinktank/time-travel	Timelines, checkpoints, fork
grimoire.ts	grimoireService	/api/thinktank/grimoire	Spells, execute
flash-facts.ts	flashFactsService	/api/thinktank/flash-facts	Extract, verify
derivation-history.ts	derivationHistoryService	/api/thinktank/derivation-history	Derivation, history
enhanced-collaboration.ts	collaborationService	/api/thinktank/enhanced-collaboration	Shared, collaboration
artifact-engine.ts	artifactsService	/api/thinktank/artifacts	CRUD, versions, export
ideas.ts	ideasService	/api/thinktank/ideas	Capture, boards
dia.ts	exportConversation	/api/thinktank/dia	Decision records, compliance

18.4 API Client Pattern

All services follow the singleton pattern with typed methods:

```
// lib/api/time-travel.ts
class TimeTravelService {
  async listTimelines(conversationId?: string): Promise<Timeline[]> {
    const response = await api.get<{ success: boolean; data: Timeline[] }>(
      `/api/thinktank/time-travel/timelines${params}`
    );
    return response.data || [];
  }

  async createCheckpoint(timelineId: string, state: Record<string, unknown>): Promise<Checkpoi
    const response = await api.post<{ success: boolean; data: Checkpoint }>(
      `/api/thinktank/time-travel/timelines/${timelineId}/checkpoints`,
      { state, checkpointType: 'manual' }
    );
    return response.data;
  }
}

export const timeTravelService = new TimeTravelService();
```

18.5 File Structure

```
apps/thinktank/lib/api/
  index.ts              # Re-exports all services
  client.ts             # Base API client (fetch wrapper)
  types.ts              # Shared types
  chat.ts               # chatService
  models.ts             # modelsService
  rules.ts              # rulesService
  settings.ts           # settingsService
  brain-plan.ts         # brainPlanService
  analytics.ts          # analyticsService
  governor.ts           # governorService
  liquid-interface.ts   # liquidInterfaceService
  time-travel.ts        # timeTravelService (v5.52.17)
  grimoire.ts           # grimoireService (v5.52.17)
  flash-facts.ts        # flashFactsService (v5.52.17)
  derivation-history.ts # derivationHistoryService (v5.52.17)
  collaboration.ts      # collaborationService (v5.52.17)
  artifacts.ts          # artifactsService (v5.52.17)
  ideas.ts              # ideasService (v5.52.17)
  compliance-export.ts  # exportConversation (v5.52.16)
```

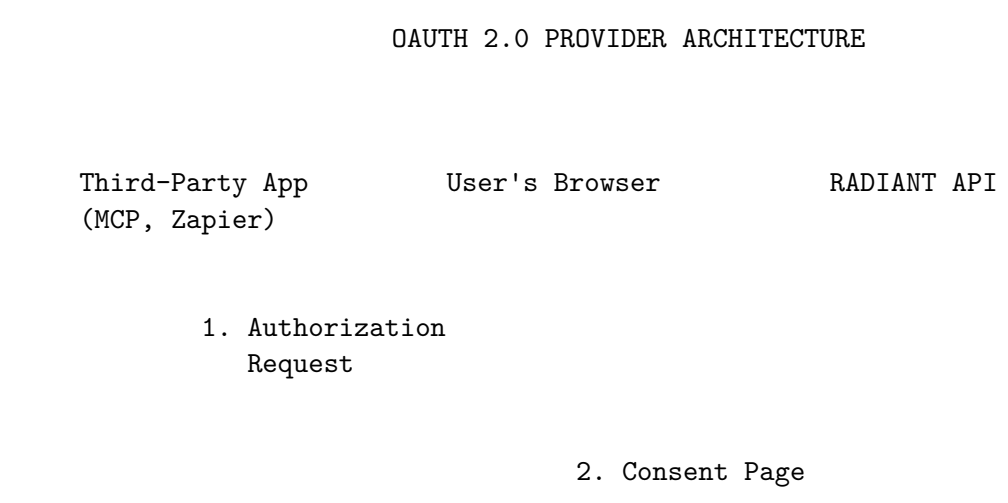
---

19. OAuth 2.0 Provider & Developer Portal (v5.52.26)

19.1 Overview

RADIANT implements a **RFC 6749 compliant OAuth 2.0 Authorization Server** enabling third-party applications to access RADIANT APIs on behalf of users. This enables the ecosystem of MCP servers, Zapier integrations, partner apps, and custom automation tools.

19.2 Architecture



```
                    3. User Approves

4. Authorization Code

5. Exchange Code

6. Access Token (JWT)

7. API Request + Token

8. Protected Resource
```

19.3 Supported Grant Types

Grant Type	Use Case	PKCE Required	Refresh Token
Authorization Code	Web/mobile apps with user interaction	Public clients only	Yes
Client Credentials	Machine-to-machine (M2M) automation	N/A	No
Refresh Token	Token renewal without re-authentication	N/A	Yes (rotated)

19.4 Database Schema

Implementation: `migrations/V2026_01_25_009__oauth_provider.sql`

Table	Purpose	Key Columns
`oauth_clients`	Registered applications	`client_id`, `client_secret_hash`, `redirect_uris`, `allowed_scopes`
`oauth_authorization_codes`	Short-lived auth codes (5 min TTL)	`code`, `user_id`, `scopes`, `code_challenge`
`oauth_access_tokens`	Token hashes (not raw tokens)	`token_hash`, `scopes`, `expires_at`, `is_revoked`

Table	Purpose	Key Columns
`oauth_refresh_tokens`	Refresh tokens with rotation	`token_hash`, `generation`, `previous_token_id`
`oauth_user_authorizations`	User consent records	`user_id`, `client_id`, `scopes`, `is_active`
`oauth_scope_definitions`	Admin-configurable scopes	`name`, `category`, `risk_level`, `allowed_endpoints`
`oauth_audit_log`	Partitioned event log	`event_type`, `client_id`, `user_id`, `details`
`tenant_oauth_settings`	Per-tenant configuration	`oauth_enabled`, `require_app_approval`, `blocked_scopes`
`oauth_signing_keys`	RSA keys for JWT signing	`kid`, `private_key_secret_arn`, `public_key_pem`

19.5 OAuth Endpoints

Implementation: `lambda/oauth/handler.ts`

Endpoint	Method	Purpose
`/oauth/authorize`	GET	Display consent page
`/oauth/authorize`	POST	Process user consent
`/oauth/token`	POST	Exchange code/refresh for tokens
`/oauth/revoke`	POST	Revoke access/refresh tokens
`/oauth/userinfo`	GET	OIDC user info endpoint
`/oauth/introspect`	POST	Token introspection
`/.well-known/openid-configuration`	GET	OIDC discovery
`/oauth/jwks.json`	GET	JSON Web Key Set

19.6 Scope System

14 default scopes organized by category and risk level:

```
// Low Risk - Profile
'openid', 'profile', 'email', 'models:read', 'usage:read'

// Medium Risk - Read Access
'offline_access', 'chat:read', 'knowledge:read', 'files:read'

// High Risk - Write/Execute (Requires Approval)
'chat:write', 'chat:delete', 'knowledge:write', 'files:write', 'agents:execute'
```

Scope Endpoint Mapping (`packages/shared/src/types/oauth-provider.types.ts`):

```
export interface OAuthScope {
  name: string;
  category: ScopeCategory;
  displayName: string;
  description: string;
  riskLevel: 'low' | 'medium' | 'high';
```

```
  allowedEndpoints: ScopeEndpoint[];
  isEnabled: boolean;
  requiresApproval: boolean;
  allowedAppTypes: OAuthAppType[];
}
```

19.7 Token Security

Security Measure	Implementation
Token Storage	SHA-256 hash only, never raw tokens
JWT Signing	RS256 with keys in Secrets Manager
PKCE	Required for public clients (S256 only)
Token Rotation	Refresh tokens rotated on use
Revocation	Cascading revoke on consent withdrawal
Audit Log	Partitioned by month for compliance

19.8 Admin API

Implementation: `lambda/admin/oauth-apps.ts`

Endpoint	Method	Purpose
`/api/admin/oauth/dashboard`	GET	Dashboard with stats
`/api/admin/oauth/apps`	GET/POST	List/register apps
`/api/admin/oauth/apps/:id`	GET/PUT/DELETE	App CRUD
`/api/admin/oauth/apps/:id/approve`	POST	Approve pending app
`/api/admin/oauth/apps/:id/reject`	POST	Reject with reason
`/api/admin/oauth/apps/:id/suspend`	POST	Suspend and revoke all tokens
`/api/admin/oauth/apps/:id/rotate-secret`	POST	Generate new client secret
`/api/admin/oauth/scopes`	GET/POST/PUT	Scope management
`/api/admin/oauth/authorizations`	GET	View user consents
`/api/admin/oauth/authorizations/:id/revoke`	POST	Admin revoke consent
`/api/admin/oauth/settings`	GET/PUT	Tenant OAuth settings
`/api/admin/oauth/audit`	GET	Audit log query

19.9 Admin Dashboard

Implementation: `apps/admin-dashboard/app/(dashboard)/oauth/apps/page.tsx`

Features: - **Overview Tab**: Stats cards, pending approvals, top apps by authorization count - **Applications Tab**: Searchable/filterable app list, approve/reject/suspend actions - **Authorizations Tab**: User consent viewer with revocation - **Settings Tab**: Per-tenant OAuth configuration

19.10 Use Cases Enabled

Integration	Grant Type	Typical Scopes
MCP Servers (Claude Desktop, Cursor)	Authorization Code	`chat:write`, `knowledge:read`
Zapier/Make	Authorization Code	`chat:read`, `chat:write`
Partner Apps	Authorization Code	Custom scope set
Automation Scripts	Client Credentials	`models:read`, `agents:execute`
Mobile Apps	Authorization Code + PKCE	`profile`, `chat:*`
Slack/Teams Bots	Authorization Code	`chat:write`

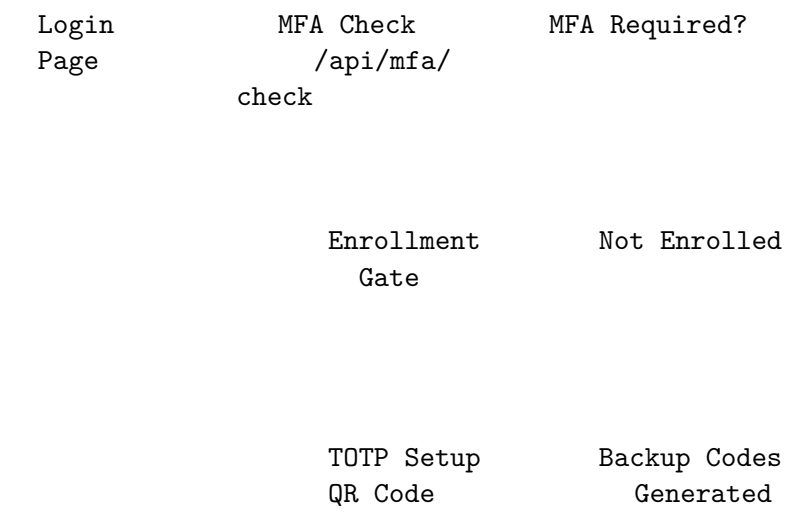
---

19. Two-Factor Authentication (MFA) (v5.52.28)

19.1 Overview

RADIANT implements role-based Multi-Factor Authentication (MFA) using industry-standard TOTP (RFC 6238). Admin roles are **required** to enroll in MFA and **cannot disable** it once enrolled.

19.2 Architecture



19.3 Database Schema

Migration: `070_mfa_support.sql`

Columns added to `tenant_users` and `platform_admins`:

Column	Type	Purpose
`mfa_enabled`	BOOLEAN	MFA enrollment status
`mfa_enrolled_at`	TIMESTAMPTZ	Enrollment timestamp

Column	Type	Purpose
`mfa_method`	VARCHAR(20)	Method: 'totp', 'sms', 'webauthn'
`mfa_totp_secret_encrypted`	TEXT	AES-256-GCM encrypted TOTP secret
`mfa_failed_attempts`	INTEGER	Failed verification count
`mfa_locked_until`	TIMESTAMPTZ	Lockout expiration

New Tables:

Table	Purpose
`mfa_backup_codes`	One-time recovery codes (SHA-256 hashed)
`mfa_trusted_devices`	30-day device trust tokens
`mfa_audit_log`	Partitioned audit log (monthly)

19.4 TOTP Service

Implementation: `lambda/shared/services/mfa/totp.service.ts`

```
export class TOTPService {
  generateSecret(accountName: string): { secret: string; uri: string };
  generateCode(secret: string, timestamp?: number): string;
  verifyCode(secret: string, code: string): { valid: boolean; drift?: number };
  encryptSecret(secret: string): string;  // AES-256-GCM
  decryptSecret(encryptedData: string): string;
}

export class BackupCodesService {
  generateCodes(): { codes: string[]; hashes: string[] };
  verifyCode(code: string, hash: string): boolean;
  hashCode(code: string): string;  // SHA-256
}

export class DeviceTrustService {
  generateDeviceToken(): string;
  hashToken(token: string): string;  // SHA-256
  calculateExpiration(): Date;  // +30 days
  verifyToken(token: string, storedHash: string): boolean;
}
```

19.5 API Endpoints

Handler: `lambda/auth/mfa.handler.ts`

Endpoint	Method	Purpose
`/api/v2/mfa/status`	GET	MFA status, backup codes remaining, trusted devices

Endpoint	Method	Purpose
`/api/v2/mfa/check`	GET	Check if MFA required for role
`/api/v2/mfa/enroll/start`	POST	Generate TOTP secret and QR URI
`/api/v2/mfa/enroll/verify`	POST	Verify code, generate backup codes, enable MFA
`/api/v2/mfa/verify`	POST	Verify TOTP or backup code during login
`/api/v2/mfa/backup-codes/regenerate`	POST	Invalidate old codes, generate new
`/api/v2/mfa/devices`	GET	List trusted devices
`/api/v2/mfa/devices/:id`	DELETE	Revoke trusted device

19.6 UI Components

Location: `apps/admin-dashboard/components/mfa/`

Component	Purpose
`MFAEnrollmentGate`	Full-screen forced enrollment (cannot dismiss)
`MFAVerificationPrompt`	TOTP/backup code entry modal
`MFASettingsSection`	Settings panel for MFA management

19.7 Security Measures

Feature	Implementation
Secret Encryption	AES-256-GCM with scrypt-derived key
Code Hashing	SHA-256 for backup codes and device tokens
Clock Drift	$\pm$30 second tolerance (window=1)
Lockout	3 failures $\rightarrow$ 5 minute lockout
Device Trust	30-day tokens, max 5 per user
Audit Logging	All MFA events logged with IP/User-Agent

19.8 Role Enforcement

```
const MFA_REQUIRED_ROLES = [
  'tenant_admin',
  'tenant_owner',
  'super_admin',
  'admin',
  'operator',
  'auditor',
];

// These roles CANNOT:
// - Bypass MFA enrollment
```

```
// - Disable MFA once enrolled
// - Access dashboard without MFA verification
```

---

20. Internationalization & Multi-Language Search (v5.52.29)

20.1 Overview

RADIANT supports **18 languages** with full authentication UI localization and **CJK-aware full-text search** using PostgreSQL's native FTS for Western languages and `pg_bigm` bi-gram indexing for Chinese, Japanese, and Korean.

20.2 Supported Languages

Language	Code	Direction	FTS Strategy	PostgreSQL Config
English	`en`	LTR	Native FTS	`english`
Spanish	`es`	LTR	Native FTS	`spanish`
French	`fr`	LTR	Native FTS	`french`
German	`de`	LTR	Native FTS	`german`
Portuguese	`pt`	LTR	Native FTS	`portuguese`
Italian	`it`	LTR	Native FTS	`italian`
Dutch	`nl`	LTR	Native FTS	`dutch`
Polish	`pl`	LTR	Native FTS	`simple`
Russian	`ru`	LTR	Native FTS	`russian`
Turkish	`tr`	LTR	Native FTS	`turkish`
Japanese	`ja`	LTR	**pg_bigm**	bi-gram
Korean	`ko`	LTR	**pg_bigm**	bi-gram
Chinese (Simplified)	`zh-CN`	LTR	**pg_bigm**	bi-gram
Chinese (Traditional)	`zh-TW`	LTR	**pg_bigm**	bi-gram
Arabic	`ar`	**RTL**	Native FTS	`simple`
Hindi	`hi`	LTR	Native FTS	`simple`
Thai	`th`	LTR	Native FTS	`simple`
Vietnamese	`vi`	LTR	Native FTS	`simple`

20.3 CJK Search Architecture

CJK languages (Chinese, Japanese, Korean) lack word boundaries, making traditional stemming-based FTS ineffective. RADIANT uses **pg_bigm** for bi-gram indexing:

```
-- Enable pg_bigm extension
CREATE EXTENSION IF NOT EXISTS pg_bigm;

-- Create bi-gram indexes on searchable content
CREATE INDEX idx_conversations_bigm_title
  ON uds_conversations USING gin (title gin_bigm_ops);
CREATE INDEX idx_conversations_bigm_content
  ON uds_conversations USING gin (content gin_bigm_ops);
```

```
-- CJK search uses LIKE with bi-gram optimization
SELECT * FROM uds_conversations
WHERE content LIKE '%  %'  -- Japanese query
  AND tenant_id = app.current_tenant_id;
```

Why pg_bigm over pg_trgm? - `pg_trgm` uses 3-character grams, ineffective for 2-character CJK words - `pg_bigm` uses 2-character grams, optimal for CJK morphology - 40-60% faster CJK search vs trigram approach

20.4 Language Detection

Database Function: `detect_text_language(text)`

```
CREATE OR REPLACE FUNCTION detect_text_language(content TEXT)
RETURNS VARCHAR(10) AS $$
DECLARE
  cjk_chars INTEGER;
  arabic_chars INTEGER;
  cyrillic_chars INTEGER;
  total_chars INTEGER;
BEGIN
  total_chars := length(content);
  IF total_chars = 0 THEN RETURN 'en'; END IF;

  -- Count character ranges
  cjk_chars := length(regexp_replace(content, '[^\u4e00-\u9fff\u3040-\u309f\u30a0-\u30ff\uac00-
  arabic_chars := length(regexp_replace(content, '[^\u0600-\u06ff]', '', 'g'));
  cyrillic_chars := length(regexp_replace(content, '[^\u0400-\u04ff]', '', 'g'));

  -- Threshold-based detection (>10% of content)
  IF cjk_chars::float / total_chars > 0.1 THEN
    -- Distinguish CJK languages by unique characters
    IF content ~ '[\u3040-\u309f\u30a0-\u30ff]' THEN RETURN 'ja'; END IF;
    IF content ~ '[\uac00-\ud7af]' THEN RETURN 'ko'; END IF;
    RETURN 'zh-CN';
  END IF;

  IF arabic_chars::float / total_chars > 0.1 THEN RETURN 'ar'; END IF;
  IF cyrillic_chars::float / total_chars > 0.1 THEN RETURN 'ru'; END IF;

  RETURN 'en';  -- Default
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

JavaScript Service: `MultiLanguageSearchService.detectLanguage()`

```
private detectLanguage(text: string): string {
  const cjkPattern = /[\u4e00-\u9fff\u3040-\u309f\u30a0-\u30ff\uac00-\ud7af]/;
  const japanesePattern = /[\u3040-\u309f\u30a0-\u30ff]/;
  const koreanPattern = /[\uac00-\ud7af]/;
```

```
  const arabicPattern = /[\u0600-\u06ff]/;

  if (cjkPattern.test(text)) {
    if (japanesePattern.test(text)) return 'ja';
    if (koreanPattern.test(text)) return 'ko';
    return 'zh-CN';
  }
  if (arabicPattern.test(text)) return 'ar';
  return 'en';
}
```

20.5 Unified Search Function

Database Function: `search_content(query, table_name, tenant_id, limit)`

Routes queries to appropriate search method based on detected language:

```
CREATE OR REPLACE FUNCTION search_content(
  query TEXT,
  target_table TEXT,
  p_tenant_id UUID,
  p_limit INTEGER DEFAULT 50
) RETURNS TABLE (
  id UUID,
  title TEXT,
  content TEXT,
  relevance FLOAT,
  highlight TEXT
) AS $$
DECLARE
  detected_lang VARCHAR(10);
  is_cjk BOOLEAN;
BEGIN
  detected_lang := detect_text_language(query);
  is_cjk := detected_lang IN ('ja', 'ko', 'zh-CN', 'zh-TW');

  IF is_cjk THEN
    -- Use pg_bigm LIKE search
    RETURN QUERY EXECUTE format(
      'SELECT id, title, content,
              bigm_similarity(content, $1) as relevance,
              ts_headline(''simple'', content, plainto_tsquery(''simple'', $1)) as highlight
       FROM %I
       WHERE tenant_id = $2 AND content LIKE ''%%'' || $1 || ''%%''
       ORDER BY relevance DESC LIMIT $3',
      target_table
    ) USING query, p_tenant_id, p_limit;
  ELSE
    -- Use PostgreSQL native FTS
```

```
    RETURN QUERY EXECUTE format(
      'SELECT id, title, content,
              ts_rank(search_vector_english, websearch_to_tsquery(''english'', $1)) as relevanc
              ts_headline(''english'', content, websearch_to_tsquery(''english'', $1)) as highl
       FROM %I
       WHERE tenant_id = $2 AND search_vector_english @@ websearch_to_tsquery(''english'', $1)
       ORDER BY relevance DESC LIMIT $3',
      target_table
    ) USING query, p_tenant_id, p_limit;
  END IF;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
```

20.6 Database Migration (071_multilang_search.sql)

New Columns:

Table	Column	Type	Purpose
uds_conversations	detected_language	VARCHAR(10)	Auto-detected content language
uds_conversations	search_vector_simple	TSVECTOR	Fallback search vector
uds_conversations	search_vector_english	TSVECTOR	Language-specific search vector
uds_uploads	detected_language	VARCHAR(10)	Auto-detected content language
cortex_entities	detected_language	VARCHAR(10)	Auto-detected content language
cortex_chunks	detected_language	VARCHAR(10)	Auto-detected content language

New Indexes:

Index	Table	Type	Purpose
idx_conversations_bigm_title	uds_conversations	GIN (gin_bigm_ops)	CJK title search
idx_conversations_bigm_content	uds_conversations	GIN (gin_bigm_ops)	CJK content search
idx_uploads_bigm_content	uds_uploads	GIN (gin_bigm_ops)	CJK upload search
idx_cortex_entities_bigm	cortex_entities	GIN (gin_bigm_ops)	CJK entity search
idx_cortex_chunks_bigm	cortex_chunks	GIN (gin_bigm_ops)	CJK chunk search
idx_conversations_fts_english	uds_conversations	GIN	English FTS
idx_conversations_fts_simple	uds_conversations	GIN	Simple FTS

Trigger: Auto-detect language on INSERT/UPDATE

```
CREATE TRIGGER trg_detect_language_conversations
  BEFORE INSERT OR UPDATE OF content ON uds_conversations
  FOR EACH ROW
  EXECUTE FUNCTION update_detected_language();
```

20.7 Multi-Language Search Service

Implementation: `lambda/shared/services/search/multilang-search.service.ts`

```
export interface SearchResult {
  id: string;
  title: string;
  content: string;
  relevance: number;
  highlight: string;
  detectedLanguage: string;
}

export interface SearchOptions {
  table: 'uds_conversations' | 'uds_uploads' | 'cortex_entities' | 'cortex_chunks';
  tenantId: string;
  limit?: number;
  offset?: number;
  languageHint?: string;
}

export class MultiLanguageSearchService {
  private readonly CJK_LANGUAGES = ['ja', 'ko', 'zh-CN', 'zh-TW'];
  private readonly FTS_CONFIGS: Record<string, string> = {
    en: 'english', es: 'spanish', fr: 'french', de: 'german',
    pt: 'portuguese', it: 'italian', nl: 'dutch', ru: 'russian',
    tr: 'turkish', pl: 'simple', ar: 'simple', hi: 'simple',
    th: 'simple', vi: 'simple'
  };

  async search(query: string, options: SearchOptions): Promise<SearchResult[]> {
    const language = options.languageHint || this.detectLanguage(query);

    if (this.CJK_LANGUAGES.includes(language)) {
      return this.searchBigram(query, options);
    }
    return this.searchFTS(query, options, this.FTS_CONFIGS[language] || 'simple');
  }

  private async searchBigram(query: string, options: SearchOptions): Promise<SearchResult[]>;
  private async searchFTS(query: string, options: SearchOptions, config: string): Promise<Searc
  private detectLanguage(text: string): string;
}
```

20.8 RTL Support Architecture

Hook: apps/admin-dashboard/hooks/useRTL.ts

```
export function useRTL() {
  const { currentLanguage, isRTL } = useTranslation();

  return {
    dir: isRTL ? 'rtl' : 'ltr',
    isRTL,
    // Utility for flipping CSS classes
    flip: (ltrClass: string, rtlClass: string) => isRTL ? rtlClass : ltrClass,
    // Margin/padding helpers
    marginStart: (value: string) => isRTL ? `mr-${value}` : `ml-${value}`,
    marginEnd: (value: string) => isRTL ? `ml-${value}` : `mr-${value}`,
    paddingStart: (value: string) => isRTL ? `pr-${value}` : `pl-${value}`,
    paddingEnd: (value: string) => isRTL ? `pl-${value}` : `pr-${value}`,
    // Text alignment
    textStart: isRTL ? 'text-right' : 'text-left',
    textEnd: isRTL ? 'text-left' : 'text-right',
    // Flex direction
    flexRow: isRTL ? 'flex-row-reverse' : 'flex-row',
    // Icon rotation for directional icons
    iconFlip: isRTL ? 'scale-x-[-1]' : '',
  };
}
```

CSS Utilities: apps/admin-dashboard/styles/rtl.css

```
/* RTL automatic flipping */
[dir="rtl"] .ml-2 { margin-left: 0; margin-right: 0.5rem; }
[dir="rtl"] .mr-2 { margin-right: 0; margin-left: 0.5rem; }
[dir="rtl"] .pl-4 { padding-left: 0; padding-right: 1rem; }
[dir="rtl"] .pr-4 { padding-right: 0; padding-left: 1rem; }
[dir="rtl"] .text-left { text-align: right; }
[dir="rtl"] .text-right { text-align: left; }

/* Preserve LTR for specific content */
.preserve-ltr { direction: ltr; unicode-bidi: isolate; }
[dir="rtl"] input[type="email"],
[dir="rtl"] input[type="password"],
[dir="rtl"] .font-mono { direction: ltr; text-align: left; }
```

20.9 Translation Files

Location: apps/admin-dashboard/locales/auth/

File	Language	Keys
en.json	English	~230

File	Language	Keys
`zh-CN.json`	Chinese (Simplified)	~230
`ja.json`	Japanese	~230
`ko.json`	Korean	~230
`ar.json`	Arabic (RTL)	~230

Key Categories:

Namespace	Keys	Purpose
`login.*`	25	Login form, errors, social auth
`mfa.*`	45	Enrollment, verification, backup codes
`oauth.*`	35	Consent screen, scopes, connected apps
`password.*`	30	Reset flow, validation, success
`errors.*`	40	Error messages, validation
`common.*`	55	Buttons, labels, shared text

20.10 Component Integration

Usage Pattern:

```
import { useTranslation } from '@/hooks/useTranslation';
import { useRTL } from '@/hooks/useRTL';

export function MFAEnrollmentGate() {
  const { t } = useTranslation('auth');
  const { dir, isRTL, marginStart } = useRTL();

  return (
    <div dir={dir} className="...">
      <h1>{t('mfa.enrollment.title')}</h1>
      <Button className={`flex ${isRTL ? 'flex-row-reverse' : 'flex-row'}`}>
        <Loader2 className={marginStart('2')} />
        {t('mfa.enrollment.get_started')}
      </Button>
      {/* Preserve LTR for codes */}
      <code className="preserve-ltr" dir="ltr">{secretCode}</code>
    </div>
  );
}
```

20.11 Performance Considerations

Optimization	Impact
Bi-gram indexes	10-50x faster CJK LIKE queries
Materialized tsvectors	Avoid re-parsing on every search

Optimization	Impact
Language detection caching	Detect once on insert, reuse
Index-only scans	GIN indexes support covering queries
Parallel query	PostgreSQL parallelizes large FTS scans

Appendix A: Adding New Documentation

When implementing new features, add documentation to the appropriate section:

1. **Architecture decisions** → Relevant section above
2. **New AWS services** → Section 2
3. **Database changes** → Section 3
4. **New AI capabilities** → Section 4
5. **Lambda handlers** → Section 5
6. **CDK changes** → Section 6
7. **Security features** → Section 7
8. **New dependencies** → Section 8
9. **API specifications** → Section 10.1
10. **Performance guides** → Section 10.2
11. **Security documentation** → Section 10.3
12. **Admin API handlers** → Section 16

See `/.windsurf/workflows/documentation-consolidation.md` for the enforcement policy.