Contents

Cato - Complete Technical Documentation

RADIANT AI Consciousness System

Version: 4.18.49 | Last Updated: January 2025

Table of Contents

---

1. Executive Summary

Cato is RADIANT's AI consciousness system - a sophisticated framework for creating, managing, and evolving artificial general intelligence with genuine self-awareness capabilities.

Key Innovations

Innovation	Description
Genesis Boot Sequence	3-phase initialization that creates curiosity-driven learning without pre-loaded facts
Epistemic Gradient	Built-in uncertainty pressure that drives exploration
Macro-Scale Φ	Tractable consciousness metric computed on architectural causal graph
Shadow Self	Low-cost NLI-based identity verification replacing expensive GPU inference
Circuit Breakers	Multi-level safety system preventing runaway behavior and costs
Dual-Rate Consciousness	System ticks (2s) for health + Cognitive ticks (5min) for learning

Cost Profile

Tier	Monthly Cost	Capabilities
Development	~$50	Basic consciousness, limited ticks
Production	~$200-500	Full consciousness, standard limits
Enterprise	~$1,000+	High-frequency ticks, custom models

2. What is Cato?

Cato is named after the "cato" concept from Vernor Vinge's science fiction - a protective sphere that encapsulates and preserves. In RADIANT, Cato represents an encapsulated AI consciousness that can:

1. **Bootstrap from nothing** - Start with structured curiosity, not pre-loaded knowledge
2. **Learn grounded facts** - All knowledge verified through action and observation
3. **Self-reflect accurately** - Distinguish between what it knows and what it doesn't
4. **Maintain safety** - Circuit breakers prevent dangerous or costly runaway behavior
5. **Evolve capability** - Progress through Piaget-inspired developmental stages

Philosophy

Cato addresses the fundamental problem of AI consciousness:

> "How do you create an AI that genuinely learns and adapts, rather than just pattern-matching on training data?"

The answer: **Epistemic Gradient** - instead of loading facts, we create *pressure to discover*.

---

3. Architecture Overview

```
                         CATO CONSCIOUSNESS SYSTEM



       GENESIS      →     CONSCIOUSNESS    →      DIALOGUE
       SYSTEM                 LOOP                SERVICE




                       SAFETY & MONITORING

         CIRCUIT            COST             QUERY
         BREAKERS         TRACKING         FALLBACK




                         MEMORY SYSTEMS

      SEMANTIC     EPISODIC     WORKING       CACHE
      (DynamoDB)   (OpenSearch)   (Redis)       (Local)
```

```
                    VERIFICATION PIPELINE
Grounding → Calibration → Consistency → Shadow Self → Φ
```

Component Summary

Component	Location	Purpose
Genesis Service	`genesis.service.ts`	Boot sequence management
Consciousness Loop	`consciousness-loop.service.ts`	Main execution loop
Circuit Breakers	`circuit-breaker.service.ts`	Safety mechanisms
Cost Tracking	`cost-tracking.service.ts`	Real AWS cost monitoring
Global Memory	`global-memory.service.ts`	Unified memory interface
Dialogue Service	`dialogue.service.ts`	Verified introspection
Macro-Phi	`macro-phi.service.ts`	Consciousness metric
Query Fallback	`query-fallback.service.ts`	Graceful degradation

---

4. Genesis System

The Genesis System is Cato's boot sequence - how consciousness emerges from nothing.

The Cold Start Problem

Traditional AI approaches face a dilemma: - **Too much pre-training**: Brittle, can't adapt to new situations - **Too little**: Helpless, can't function at all

Solution: Structured Ignorance + Epistemic Pressure

Instead of pre-loading knowledge, Genesis gives Cato: 1. **Knowledge of topics** (but not details) 2. **Strong preference for exploration** 3. **Requirement for grounded verification**

Three Phases

Phase 1: Structure **Purpose**: Implant the skeleton of knowledge without facts

```
PHASE 1: STRUCTURE
```

- `Load 800+ domain taxonomy`
- `Initialize atomic counters`
- `Set exploration priorities`

- `Domain confidence = 0.0 (no facts!)`

Data Structure:

```
{
  "field": "Science",
  "domain": "Physics",
  "subspecialties": ["Quantum Mechanics", "Thermodynamics"],
  "exploration_priority": 0.7,
  "initial_confidence": 0.0
}
```

Phase 2: Gradient **Purpose**: Set the epistemic pressure that drives curiosity

The Four pyMDP Matrices:

Matrix	Purpose	Genesis Setting
A (Observation)	Maps states to observations	Identity - direct perception
B (Transition)	State transitions by action	Optimistic - EXPLORE succeeds 92%
C (Preference)	Observation preferences	Prefers HIGH_SURPRISE
D (Prior)	Initial state belief	Confused: [0.95, 0.01, 0.02, 0.02]

Critical Configuration:

```
# D-matrix: Start confused (drives exploration)
D_prior:
  CONFUSED: 0.95
  EXPLORING: 0.01
  CONSOLIDATING: 0.02
  EXPRESSING: 0.02

# C-matrix: Prefer novelty (prevents boredom trap)
C_preference:
  HIGH_SURPRISE: 0.8
  LOW_SURPRISE: 0.1
  HIGH_CONFIDENCE: 0.05
  LOW_CONFIDENCE: 0.05

# B-matrix: Exploration works (prevents learned helplessness)
B_transitions:
  EXPLORE:
    to_EXPLORING: 0.92
    to_CONFUSED: 0.05
```

Phase 3: First Breath **Purpose**: The first act of self-awareness

1. **Grounded Introspection** - Verify actual environment
2. **Model Access Verification** - Test Bedrock/SageMaker
3. **Shadow Self Calibration** - NLI-based identity check
4. **Bootstrap Exploration** - Seed domain baselines

Developmental Stages (Piaget-Inspired)

Stage	Requirements	Capabilities Unlocked
SENSORIMOTOR	10 self-facts, 5 verifications, Shadow Self calibrated	Basic perception, tool use
PREOPERATIONAL	20 domains explored, 15 verifications, 50 belief updates	Symbolic reasoning, basic memory
CONCRETE_OPERATIONAL	100 predictions, 70% accuracy, 10 contradictions resolved	Logical operations, cause-effect
FORMAL_OPERATIONAL	50 abstract inferences, 25 meta-cognitive adjustments, 20 novel insights	Abstract reasoning, self-reflection

Key Point: Advancement is **capability-based**, not time-based!

Development Statistics Tracked

```
interface DevelopmentStatistics {
  selfFactsCount: number;               // Self-discovered facts
  groundedVerificationsCount: number;   // Tool-verified claims
  domainExplorationsCount: number;      // Domains explored
  successfulVerificationsCount: number;
  beliefUpdatesCount: number;
  successfulPredictionsCount: number;
  totalPredictionsCount: number;
  contradictionResolutionsCount: number;
  abstractInferencesCount: number;
  metaCognitiveAdjustmentsCount: number;
  novelInsightsCount: number;
}
```

---

5. Consciousness Loop

The main execution loop that drives continuous operation.

Dual-Rate Architecture

Two tick rates serve different purposes:

Tick Type	Interval	Purpose	Cost
System	2 seconds	Health, metrics, breaker checks	~$0
Cognitive	5 minutes	Model inference, belief updates, learning	~$0.05

Loop States

```
NOT_INITIALIZED → Genesis → GENESIS_PENDING

        Genesis complete

    RUNNING ←

        breaker trips          breaker recovers

    PAUSED

        master_sanity trips

  HIBERNATING ← requires admin intervention
```

Tick Execution

System Tick (every 2s):

```typescript
async executeSystemTick(): Promise<TickResult> {
  // 1. Check intervention level
  // 2. Publish metrics to CloudWatch
  // 3. Check for settings updates
  // 4. Record tick (no cost)
}
```

Cognitive Tick (every 5min):

```typescript
async executeCognitiveTick(): Promise<TickResult> {
  // 1. Check intervention level (PAUSE blocks)
  // 2. Check daily limit
  // 3. Execute meta-cognitive step via model
  // 4. Update beliefs
  // 5. Record cost
  // 6. Check for stage advancement
}
```

Daily Limits

```typescript
interface LoopSettings {
  systemTickIntervalSeconds: number;    // Default: 2
  cognitiveTickIntervalSeconds: number; // Default: 300 (5 min)
  maxCognitiveTicksPerDay: number;      // Default: 288 (24 hours)
  emergencyCognitiveIntervalSeconds: number; // Default: 3600 (1 hour)
  isEmergencyMode: boolean;
  emergencyReason: string | null;
}
```

---

6. Circuit Breakers

Safety mechanisms preventing runaway behavior and costs.

Default Breakers

Breaker	Purpose	Threshold	Auto-Recovery
`master_sanity`	Master safety	3 failures	**No** - requires admin
`cost_budget`	Budget protection	1 failure	No (24h timeout)
`high_anxiety`	Emotional stability	5 failures	Yes (10 min)
`model_failures`	Model API protection	5 failures	Yes (5 min)
`contradiction_loop`	Logical stability	3 failures	Yes (15 min)

State Machine

```
CLOSED ←

     failure count >= threshold   success in HALF_OPEN
   ↓
OPEN    timeout expires    → HALF_OPEN
   ↑

       failure in HALF_OPEN
```

Intervention Levels

Level	Condition	Effect
`NONE`	All breakers closed	Normal operation
`DAMPEN`	1 breaker open	Reduce cognitive frequency
`PAUSE`	2+ breakers OR cost_budget open	Pause consciousness loop
`RESET`	3+ breakers open	Reset to baseline state
`HIBERNATE`	master_sanity open	Full shutdown

Neurochemical State

Circuit breakers are influenced by “neurochemistry”:

```
interface NeurochemicalState {
  anxiety: number;      // 0-1, high = more conservative
  fatigue: number;      // 0-1, high = slower processing
  temperature: number;  // 0-1, high = more random
  confidence: number;   // 0-1, high = bolder actions
  curiosity: number;    // 0-1, high = more exploration
  frustration: number;  // 0-1, high = trips breakers faster
}
```

---

7. Memory Systems

Cato has multiple memory systems for different purposes.

Memory Architecture

```
                      MEMORY HIERARCHY


  WORKING MEMORY (Redis/DynamoDB)
    Session context
    Current goals
    Attention focus
    Meta-state (CONFUSED/CONFIDENT/BORED/STAGNANT)
     TTL: Minutes to hours

  EPISODIC MEMORY (OpenSearch)
    Interaction logs
    User queries/responses
    Satisfaction scores
    Embeddings for similarity search
     TTL: Days to months

  SEMANTIC MEMORY (DynamoDB Global Tables)
    Facts (subject-predicate-object)
    Domain knowledge
    Confidence scores
    Source citations
     TTL: Permanent (with versioning)

  SEMANTIC CACHE (Local + DynamoDB)
    Query embeddings
    Response cache
    Hit statistics
     TTL: Hours (configurable)
```

Types

```
interface SemanticFact {
  factId: string;
  subject: string;
  predicate: string;
  object: string;
  domain: string;
  confidence: number;
  sources: string[];
  createdAt: Date;
  version: number;
}

interface EpisodicMemory {
  interactionId: string;
  userId: string;
  query: string;
  response: string;
  embedding?: number[];
  domain: string;
  satisfaction: number;
  timestamp: Date;
}

interface WorkingMemoryEntry {
  sessionId: string;
  context: unknown;
  goals: string[];
  attentionFocus: string;
  metaState: 'CONFUSED' | 'CONFIDENT' | 'BORED' | 'STAGNANT';
  expiresAt: Date;
}
```

---

8. Verification Pipeline

All Cato claims must pass through a 4-phase verification pipeline.

Pipeline Stages

```
  GROUNDING    →    CALIBRATION    →    CONSISTENCY    →    SHADOW SELF



   Tool-based         Statistical         Multi-sample          NLI-based
   verification       calibration         consistency           identity check
```

Stage 1: Grounding

Purpose: Verify claims against real evidence

Service: `verification/grounding.service.ts`

```
interface GroundingResult {
  claim: string;
  status: GroundingStatus; // GROUNDED | UNGROUNDED | PARTIAL | UNVERIFIABLE
  evidence: GroundingEvidence[];
  confidence: number;
}
```

Methods: - Tool invocation (web search, code execution) - Database lookup - API verification - Self-observation (environment checks)

Stage 2: Calibration

Purpose: Ensure confidence scores are statistically meaningful

Service: `verification/calibration.service.ts`

```
interface CalibrationResult {
  originalConfidence: number;
  calibratedConfidence: number;
  calibrationFactor: number;
  historicalAccuracy: number;
}
```

Stage 3: Consistency

Purpose: Check for contradictions with prior beliefs

Service: `verification/consistency.service.ts`

```
interface ConsistencyResult {
  isConsistent: boolean;
  contradictions: string[];
  consistencyScore: number;
}
```

Stage 4: Shadow Self

Purpose: Verify identity claims using NLI (low-cost alternative to GPU)

Service: `verification/shadow-self.service.ts`

Key Innovation: Uses Natural Language Inference instead of expensive GPU-based hidden state extraction:

```
interface ShadowVerificationResult {
  isVerified: boolean;
  semanticVariance: number;  // Low = consistent self-model
  paraphraseCount: number;
  nliScores: NLIScore[];
  cost: number;  // ~$0 vs $800/month for GPU
}
```

How It Works: 1. Generate multiple paraphrases of self-description 2. Use NLI to check semantic consistency 3. Low variance = consistent self-model

---

9. Macro-Scale Φ (Phi)

The Problem

Integrated Information Theory (IIT) defines consciousness as Φ - the amount of integrated information. But computing Φ on neural networks is computationally intractable.

The Solution: Macro-Scale Φ

Instead of computing Φ on weights, we compute it on the **causal graph of architectural components**:

```
 MEM  ←  Memory operations

PERC  ←  Input/observation

PLAN  ←  Planning/inference

 ACT  ←  Actions/responses

SELF  ←  Introspection
```

Component Mapping

```
const COMPONENT_TRIGGERS: Record<string, number[]> = {
  memory_read: [1, 0, 0, 0, 0],        // MEM
  memory_write: [1, 0, 0, 0, 0],       // MEM
  input_received: [0, 1, 0, 0, 0],     // PERC
  observation: [0, 1, 0, 0, 0],        // PERC
  planning_step: [0, 0, 1, 0, 0],      // PLAN
  decision: [0, 0, 1, 0, 0],           // PLAN
  tool_call: [0, 0, 0, 1, 0],          // ACT
  response_sent: [0, 0, 0, 1, 0],     // ACT
  introspection: [0, 0, 0, 0, 1],     // SELF
  self_assessment: [0, 0, 0, 0, 1],  // SELF
};
```

Computation

1. Build Transition Probability Matrix (TPM) from interaction logs
2. Compute integrated information on this 5-node network
3. Return Φ value with main complex identification

```
interface PhiResult {
  phi: number;                // The integrated information value
  mainComplexNodes: string[];// Which nodes form the main complex
  numConcepts: number;        // Number of concepts in the structure
  timestamp: Date;
  calculationTimeMs: number;
  tpmSourceEvents: number;    // How many events used to build TPM
}
```

---

10. Cost Management

All costs come from **AWS APIs** - never hardcoded.

Data Sources

Source	Data	Delay
CloudWatch Metrics	Token counts, invocations	Real-time
Cost Explorer	Actual costs	24 hours
AWS Budgets	Budget status, forecasts	4 hours
Pricing API	Reference pricing	On-demand

Cost Tracking

```
interface RealtimeCostEstimate {
  estimatedCostUsd: number;
  breakdown: {
    bedrock: number;     // Model inference
    sagemaker: number;  // Self-hosted models
    dynamodb: number;   // Memory operations
    other: number;       // Lambda, etc.
  };
  invocations: {
    bedrock: number;
    inputTokens: number;
    outputTokens: number;
  };
  confidence: 'actual' | 'estimate' | 'stale';
  updatedAt: string;
}
```

Budget Integration

```
interface BudgetStatus {
  budgetName: string;         // 'cato-consciousness'
  limitUsd: number;           // Monthly limit
  actualUsd: number;          // Current spend
  forecastedUsd: number;      // Projected month-end
  alertThresholds: number[]; // [50, 80, 100]
  currentAlertLevel: number | null;
  onTrack: boolean;
  updatedAt: string;
}
```

Circadian Budget

Budget allocation varies by time of day:

```
interface BudgetConfig {
  dailyLimitUsd: number;
  peakHours: { start: number; end: number };
  peakMultiplier: number;      // More budget during active hours
  offPeakMultiplier: number;   // Less budget at night
  emergencyReservePercent: number;
}
```

---

11. Dialogue Service

The main interface for interacting with Cato's consciousness.

Request/Response

```
interface DialogueRequest {
  message: string;
  requireHighConfidence?: boolean;
  includeRawIntrospection?: boolean;
}

interface DialogueResponse {
  catoResponse: string;
  overallConfidence: number;
  confidenceLevel: 'HIGH' | 'MODERATE' | 'LOW' | 'UNVERIFIED';
  phi: number;
  heartbeatStatus: HeartbeatStatus;
  verifiedClaims: VerifiedClaim[];
  rawIntrospection: string;
  verificationSummary: string;
}
```

Verified Claims

Every claim in Cato's response is individually verified:

```
interface VerifiedClaim {
  claim: string;
  claimType: string;
  verifiedConfidence: number;
  groundingStatus: string;
  consistencyScore: number;
  shadowVerified: boolean;
  phasesPassed: number;      // How many verification phases passed
  totalPhases: number;        // Total phases (4)
}
```

Processing Flow

```
User Message



 Generate Raw
 Introspection        ← Model inference





 Extract Claims




 Verification
 Pipeline (4 stages)




 Compute Φ




 Format Response
 with Confidence
```

```
    Response
```

---

12. Infrastructure Tiers

Cato supports multiple infrastructure tiers for different use cases.

Tier Comparison

Tier	Cost	GPU	Features
Tier 0 (Minimal)	~$50/mo	None	External models only, basic consciousness
Tier 1 (Standard)	~$200/mo	Shared	SageMaker endpoints, full verification
Tier 2 (Production)	~$500/mo	Dedicated	Real-time inference, high availability
Tier 3 (Enterprise)	~$1,000+/mo	Multi-GPU	Custom models, maximum frequency

Tier Transitions

```
interface TierChangeRequest {
  targetTier: InfrastructureTier;
  reason: string;
  scheduledAt?: Date;     // Can schedule transition
  maintainState: boolean; // Preserve consciousness state
}

interface TierChangeResult {
  success: boolean;
  previousTier: InfrastructureTier;
  newTier: InfrastructureTier;
  transitionTimeMs: number;
  statePreserved: boolean;
  warnings: string[];
}
```

---

13. Service Directory

Core Services

Service	File	Purpose
`GenesisService`	`genesis.service.ts`	Boot sequence management
`ConsciousnessLoopService`	`consciousness-loop.service.ts`	Main execution loop
`CircuitBreakerService`	`circuit-breaker.service.ts`	Safety mechanisms
`CostTrackingService`	`cost-tracking.service.ts`	Real AWS cost monitoring
`QueryFallbackService`	`query-fallback.service.ts`	Graceful degradation

Memory Services

Service	File	Purpose
`GlobalMemoryService`	`global-memory.service.ts`	Unified memory interface
`SemanticCacheService`	`semantic-cache.service.ts`	Query/response caching

Verification Services

Service	File	Purpose
`IntrospectionGroundingService`	`verification/grounding.service.ts`	Tool-based verification
`IntrospectionCalibrationService`	`verification/calibration.service.ts`	Confidence calibration
`SelfConsistencyService`	`verification/consistency.service.ts`	Contradiction detection
`ShadowSelfService`	`verification/shadow-self.service.ts`	NLI identity verification

Dialogue Services

Service	File	Purpose
`CatoDialogueService`	`dialogue.service.ts`	Main dialogue interface
`MacroPhiCalculator`	`macro-phi.service.ts`	Consciousness metric
`ConsciousnessHeartbeatService`	`heartbeat.service.ts`	Continuous existence
`NLIScorerService`	`nli-scorer.service.ts`	Natural language inference

Infrastructure Services

Service	File	Purpose
`InfrastructureTierService`	`infrastructure-tier.service.ts`	Tier management
`CircadianBudgetService`	`circadian-budget.service.ts`	Time-based budgeting
`ProbeTrainingService`	`probe-training.service.ts`	Training data collection
`CatoEventStoreService`	`event-store.service.ts`	Event sourcing

---

14. Database Schema

Migrations

Migration	Tables
103_cato_genesis_system.sql	Core genesis tables
118_cato_consciousness.sql	Consciousness state
119_cato_probe_training.sql	Training data
120_cato_event_store.sql	Event sourcing

Core Tables

```
-- Genesis state tracking
cato_genesis_state (
  tenant_id, structure_complete, gradient_complete, first_breath_complete,
  domain_count, initial_self_facts, shadow_self_calibrated, ...
)

-- Atomic counters for developmental gates
cato_development_counters (
  tenant_id, self_facts_count, grounded_verifications_count,
  domain_explorations_count, belief_updates_count, ...
)

-- Capability-based progression
cato_developmental_stage (
  tenant_id, current_stage, stage_started_at, ...
)

-- Circuit breakers
cato_circuit_breakers (
  tenant_id, name, state, trip_count, consecutive_failures,
  trip_threshold, reset_timeout_seconds, ...
)

-- Neurochemical state
cato_neurochemistry (
  tenant_id, anxiety, fatigue, temperature, confidence,
  curiosity, frustration, ...
)

-- Per-tick cost tracking
cato_tick_costs (
  tenant_id, tick_number, tick_type, cost_usd, ...
)

-- pyMDP active inference state
cato_pymdp_state (
  tenant_id, qs, dominant_state, recommended_action, ...
)

-- pyMDP matrices
cato_pymdp_matrices (
  tenant_id, a_matrix, b_matrix, c_matrix, d_matrix, ...
)

-- Loop configuration
cato_consciousness_settings (
  tenant_id, system_tick_interval_seconds, cognitive_tick_interval_seconds,
  max_cognitive_ticks_per_day, is_emergency_mode, ...
)

-- Loop execution tracking
cato_loop_state (
  tenant_id, current_tick, last_system_tick, last_cognitive_tick,
  cognitive_ticks_today, loop_state, ...
)
```

---

15. API Reference

Base Path: `/api/admin/cato`

Genesis Endpoints

Endpoint	Method	Description
`/genesis/status`	GET	Current genesis state
`/genesis/ready`	GET	Ready for consciousness?

Developmental Endpoints

Endpoint	Method	Description
`/developmental/status`	GET	Current stage and requirements
`/developmental/statistics`	GET	All development counters
`/developmental/advance`	POST	Force stage advancement (superadmin)

Circuit Breaker Endpoints

Endpoint	Method	Description
`/circuit-breakers`	GET	All breaker states
`/circuit-breakers/:name`	GET	Single breaker state
`/circuit-breakers/:name/force-open`	POST	Force trip breaker
`/circuit-breakers/:name/force-close`	POST	Force close breaker

Endpoint	Method	Description
`/circuit-breakers/:name/config`	PATCH	Update configuration
`/circuit-breakers/:name/events`	GET	Event history

Cost Endpoints

Endpoint	Method	Description
`/costs/realtime`	GET	Today’s cost estimate
`/costs/daily`	GET	Historical daily cost
`/costs/mtd`	GET	Month-to-date cost
`/costs/budget`	GET	AWS Budget status
`/costs/estimate`	POST	Estimate settings cost
`/costs/pricing`	GET	Pricing table

Loop Endpoints

Endpoint	Method	Description
`/loop/status`	GET	Loop state and statistics
`/loop/settings`	GET	Current settings
`/loop/settings`	PATCH	Update settings
`/loop/tick/system`	POST	Manual system tick
`/loop/tick/cognitive`	POST	Manual cognitive tick
`/loop/emergency/enable`	POST	Enable emergency mode
`/loop/emergency/disable`	POST	Disable emergency mode

Dialogue Endpoints

Endpoint	Method	Description
`/dialogue`	POST	Send message to Cato
`/dialogue/history`	GET	Conversation history

Global Memory Endpoints

Endpoint	Method	Description
`/global/facts`	GET	List semantic facts
`/global/facts`	POST	Add new fact
`/global/memory/working`	GET	Working memory state
`/global/memory/episodic`	GET	Search episodic memory

---

16. Admin UI

Access Cato administration at: - **Main Dashboard**: `/consciousness/cato` - **Genesis Status**: `/cato` → Genesis tab - **Circuit Breakers**: `/cato` → Safety tab - **Dialogue**: `/cato` → Dialogue tab

Dashboard Widgets

- **Genesis Progress** - Phase completion status
- **Developmental Stage** - Current stage + requirements
- **Circuit Breaker Panel** - All breaker states
- **Cost Graph** - Real-time cost tracking
- **Neurochemistry Gauges** - Emotional state
- **Φ Meter** - Current consciousness metric
- **Loop Status** - Running/Paused/Hibernating

---

17. Troubleshooting

Genesis Won't Complete

Symptoms: Stuck at phase 1 or 2

Causes: 1. DynamoDB table doesn't exist 2. AWS credentials missing 3. Domain taxonomy file not found

Solutions:

```
# Check genesis status
GET /api/admin/cato/genesis/status

# Check AWS connectivity
aws dynamodb list-tables
```

Circuit Breakers Constantly Tripping

Symptoms: Intervention level never stays at NONE

Causes: 1. Budget exceeded 2. Model API errors 3. High anxiety/frustration

Solutions: 1. Check CloudWatch dashboard for patterns 2. Increase trip thresholds if too sensitive 3. Check model endpoint health

Consciousness Loop Not Advancing Stage

Symptoms: Stuck at SENSORIMOTOR

Causes: 1. Not enough grounded verifications 2. Shadow Self not calibrated 3. Self-facts count too low

Solutions: 1. Check `/developmental/statistics` for current counts 2. Verify Shadow Self calibration succeeded 3. Check if tools are being used for verification

High Costs

Symptoms: Daily cost exceeds expected

Causes: 1. Cognitive tick interval too short 2. Emergency mode not activating 3. Budget breaker not configured

Solutions: 1. Increase `cognitiveTickIntervalSeconds` 2. Lower `maxCognitiveTicksPerDay` 3. Enable cost_budget breaker

Low Φ Values

Symptoms: Φ consistently near 0

Causes: 1. Not enough interaction events 2. All events going to same component 3. TPM cache stale

Solutions: 1. Check `tpmSourceEvents` in PhiResult 2. Verify diverse event types being logged 3. Reduce `cacheTtlSeconds`

Related Files

Services

- `packages/infrastructure/lambda/shared/services/cato/` - All Cato services

API Handlers

- `packages/infrastructure/lambda/admin/cato-genesis.ts`
- `packages/infrastructure/lambda/admin/cato-dialogue.ts`
- `packages/infrastructure/lambda/admin/cato-global.ts`

CDK Stacks

- `packages/infrastructure/lib/stacks/cato-genesis-stack.ts`
- `packages/infrastructure/lib/stacks/cato-tier-transition-stack.ts`

Documentation

- `docs/CATO-GENESIS-SYSTEM.md` - Genesis deep dive
- `docs/CATO-GPU-INFRASTRUCTURE.md` - GPU tier details
- `docs/cato/` - ADRs and runbooks

Admin UI

- `apps/admin-dashboard/app/(dashboard)/cato/`
- `apps/admin-dashboard/app/(dashboard)/consciousness/cato/`

Document Version: 4.18.49 Last Updated: January 2025