# Contents



# RADIANT PROMPT-35: Mission Control Human-in-the-Loop (HITL) System

**For: Windsurf / Claude Opus 4.5 Version:** 4.19.2 **Status:** Implementation Complete

---

## Executive Summary

Implement a complete Human-in-the-Loop (HITL) system for RADIANT's Think Tank multi-agent swarm orchestration. The system enables human oversight of AI-generated decisions, particularly for high-stakes domains (medical, financial, legal).

---

## Critical Implementation Requirements

### MANDATORY RULES

1. **NO STUBS, MOCKS, OR PLACEHOLDERS** - All code must be production-ready
2. **NO TODO COMMENTS** - Everything must be fully implemented

3. **Signal names MUST match** - Python `wait_for_input(name=f"human_decision_{decision_id}")` must match TypeScript `sendSignal(executionId, signalId)`
4. **RLS enforcement** - Use `SET app.tenant_id` (session-level) with `RESET` in finally block
5. **S3 offloading** - Never pass raw JSON to Flyte; upload to S3 Bronze layer first
6. **`@dynamic(cache=False)`** - Required on swarm execution to prevent zombie cache
7. **True swarm loop** - Iterate ALL agents, not just `agents[0]`

---

### Phase 1: Database Migration

**File:** `packages/infrastructure/migrations/V2026_01_07_001__mission_control_schema.sql`

Create 4 tables with RLS policies:

```sql
-- 1. pending_decisions: Core HITL decision tracking
CREATE TABLE pending_decisions (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  tenant_id UUID NOT NULL REFERENCES tenants(id),
  session_id UUID NOT NULL,
  question TEXT NOT NULL,
  context JSONB NOT NULL DEFAULT '{}',
  options JSONB DEFAULT '[]',
  topic_tag VARCHAR(255),
  domain VARCHAR(50) NOT NULL CHECK (domain IN ('medical', 'financial', 'legal', 'general')),
  urgency VARCHAR(20) DEFAULT 'normal' CHECK (urgency IN ('low', 'normal', 'high', 'critical'))
  status VARCHAR(20) DEFAULT 'pending' CHECK (status IN ('pending', 'resolved', 'expired', 'es
  timeout_seconds INTEGER NOT NULL,
  expires_at TIMESTAMPTZ NOT NULL,
  flyte_execution_id VARCHAR(256) NOT NULL,
  flyte_node_id VARCHAR(256) NOT NULL,
  cato_escalation_id UUID,
  resolution VARCHAR(50),
  guidance TEXT,
  resolved_by UUID,
  resolved_at TIMESTAMPTZ,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- 2. decision_audit: Complete audit trail
CREATE TABLE decision_audit (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  decision_id UUID NOT NULL REFERENCES pending_decisions(id),
  tenant_id UUID NOT NULL REFERENCES tenants(id),
  action VARCHAR(50) NOT NULL,
  actor_id UUID,
  actor_type VARCHAR(50) NOT NULL,
  details JSONB NOT NULL DEFAULT '{}',
```

```sql
  created_at TIMESTAMPTZ DEFAULT NOW()
);

-- 3. decision_domain_config: Per-domain timeout and escalation settings
CREATE TABLE decision_domain_config (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  tenant_id UUID REFERENCES tenants(id),
  domain VARCHAR(50) NOT NULL,
  default_timeout_seconds INTEGER NOT NULL,
  escalation_timeout_seconds INTEGER NOT NULL,
  auto_escalate BOOLEAN DEFAULT TRUE,
  escalation_channel VARCHAR(100),
  escalation_target TEXT,
  required_roles TEXT[] DEFAULT '{}',
  allow_auto_resolve BOOLEAN DEFAULT FALSE,
  require_guidance BOOLEAN DEFAULT TRUE,
  sanitize_phi BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW(),
  UNIQUE(tenant_id, domain)
);

-- 4. websocket_connections: Active WebSocket connections
CREATE TABLE websocket_connections (
  connection_id VARCHAR(256) PRIMARY KEY,
  tenant_id UUID NOT NULL REFERENCES tenants(id),
  user_id UUID,
  subscribed_domains TEXT[] DEFAULT '{}',
  connected_at TIMESTAMPTZ DEFAULT NOW(),
  last_heartbeat TIMESTAMPTZ DEFAULT NOW()
);

-- RLS Policies (CRITICAL: Use app.tenant_id, not current_tenant_id)
ALTER TABLE pending_decisions ENABLE ROW LEVEL SECURITY;
CREATE POLICY pending_decisions_tenant_isolation ON pending_decisions
  USING (tenant_id = current_setting('app.tenant_id', true)::uuid);

ALTER TABLE decision_audit ENABLE ROW LEVEL SECURITY;
CREATE POLICY decision_audit_tenant_isolation ON decision_audit
  USING (tenant_id = current_setting('app.tenant_id', true)::uuid);

-- Default domain configurations
INSERT INTO decision_domain_config (tenant_id, domain, default_timeout_seconds, escalation_time
  (NULL, 'medical', 300, 60, true, 'pagerduty'),
  (NULL, 'financial', 600, 120, true, 'pagerduty'),
  (NULL, 'legal', 900, 180, true, 'email'),
  (NULL, 'general', 1800, 300, true, 'slack');
```

---

### Phase 2: CDK Infrastructure Stack

**File:** `packages/infrastructure/lib/stacks/mission-control-stack.ts`

Create CDK stack with: - Mission Control API Lambda (REST endpoints) - WebSocket Handler Lambda (connection management) - Timeout Cleanup Lambda (scheduled, every 1 minute) - Redis Bridge ECS Fargate Service - API Gateway REST API - API Gateway WebSocket API - EventBridge scheduled rule for cleanup

**Key Environment Variables:**

```
environment: {
  DB_SECRET_ARN: dbSecret.secretArn,
  REDIS_HOST: redis.attrRedisEndpointAddress,
  REDIS_PORT: redis.attrRedisEndpointPort,
  FLYTE_ADMIN_URL: 'http://flyte.internal:30080',
  WEBSOCKET_API_ENDPOINT: webSocketApi.apiEndpoint,
  PAGERDUTY_ROUTING_KEY: pagerdutyKey.secretValue,
  SLACK_WEBHOOK_URL: slackWebhook.secretValue,
}
```

---

### Phase 3: RadiantSwarm Service

**File:** `packages/infrastructure/lambda/shared/services/swarm/radiant-swarm.ts`

Scatter-gather orchestrator with: - Parallel agent execution via `Promise.all()` - HITL integration via FlyteLauncher - Redis event publishing - Per-agent safety checks via Cato

**Critical Code Pattern:**

```
// CORRECT: Iterate ALL agents
const agentPromises = request.agents.map(agent =>
  this.executeAgent(agent, request.task)
);
const results = await Promise.all(agentPromises);

// WRONG: Only first agent
const result = await this.executeAgent(request.agents[0], request.task);
```

---

### Phase 4: FlyteLauncher Service

**File:** `packages/infrastructure/lambda/shared/services/swarm/flyte-launcher.ts`

Flyte workflow integration with: - S3 input upload to Bronze layer - Workflow launch via Flyte Admin API - Signal sending for HITL resolution - Execution abort for timeouts

**S3 Offloading Pattern:**

```typescript
// Upload input to S3 (CRITICAL: Never pass raw JSON to Flyte)
const s3Key = `flyte-inputs/${tenantId}/${swarmId}/input.json`;
await s3.putObject({
  Bucket: 'radiant-bronze',
  Key: s3Key,
  Body: JSON.stringify(inputData),
});

// Pass only s3_uri to Flyte
await this.launchWorkflow('think_tank_hitl_workflow', {
  s3_uri: `s3://radiant-bronze/${s3Key}`,
  swarm_id: swarmId,
  tenant_id: tenantId,
  // ...
});
```

---

### Phase 5: Python Flyte Workflow

**File:** packages/flyte/workflows/think_tank_workflow.py

True swarm parallelism with HITL:

```python
from flytekit import workflow, task, dynamic, wait_for_input
from datetime import timedelta

@task
def load_input_from_s3_task(s3_uri: str) -> Dict[str, Any]:
    """Load input data from S3 Bronze layer."""
    import boto3
    s3 = boto3.client('s3')
    bucket, key = s3_uri.replace('s3://', '').split('/', 1)
    response = s3.get_object(Bucket=bucket, Key=key)
    return json.loads(response['Body'].read().decode('utf-8'))

@dynamic(cache=False)  # CRITICAL: Prevents zombie cache
def execute_swarm(agents: List[AgentConfig], task_data: TaskData) -> List[AgentResult]:
    """Execute all agents in parallel."""
    results = []
    for agent in agents:  # CRITICAL: Iterate ALL agents
        result = execute_single_agent_task(agent=agent, task_data=task_data)
        results.append(result)
    return results

@workflow
def think_tank_hitl_workflow(
    s3_uri: str,
    swarm_id: str,
```

```
    tenant_id: str,
    session_id: str,
    user_id: str,
    hitl_domain: str,
) -> Dict[str, Any]:
    # 1. Load input from S3
    input_data = load_input_from_s3_task(s3_uri=s3_uri)

    # 2. Execute agent swarm (true parallel)
    agent_results = execute_swarm(
        agents=input_data['agents'],
        task_data=input_data['task']
    )

    # 3. Synthesize results
    synthesis = synthesize_results(agent_results=agent_results)

    # 4. Create decision and wait for human input
    decision_id = create_pending_decision(...)

    # Signal name MUST match what API sends
    human_decision = wait_for_input(
        name=f"human_decision_{decision_id}",  # CRITICAL: Use decision_id
        timeout=timedelta(seconds=timeout_seconds),
        expected_type=dict,
    )

    # 5. Incorporate human guidance
    final_response = perform_deep_reasoning(synthesis, human_decision)

    return {"status": "completed", "response": final_response}
```

---

### Phase 6: Mission Control API Lambda

**File:** `packages/infrastructure/lambda/functions/mission-control/index.ts`

REST endpoints: - `GET /decisions` - List pending decisions (with filters) - `GET /decisions/:id` - Get single decision - `POST /decisions/:id/resolve` - Resolve decision (sends Flyte signal) - `GET /stats` - Dashboard statistics - `GET /config` - Domain configuration - `PUT /config` - Update domain configuration

**RLS Pattern (CRITICAL):**

```
async function withTenantContext<T>(
  db: Client,
  tenantId: string,
  fn: () => Promise<T>
): Promise<T> {
```

```
  try {
    // SET at session level, not transaction
    await db.query(`SET app.tenant_id = $1`, [tenantId]);
    return await fn();
  } finally {
    // ALWAYS reset in finally block
    await db.query(`RESET app.tenant_id`);
  }
}
```

---

### Phase 7: WebSocket Handler Lambda

**File:** `packages/infrastructure/lambda/functions/websocket/connection-handler.ts`

Handle WebSocket lifecycle: - `$connect` - Store connection in Redis + DB - `$disconnect` - Remove connection - `$default` - Handle ping/pong, subscriptions - Scheduled cleanup of stale connections (>5 min no heartbeat)

---

### Phase 8: Redis Bridge Service

**File:** `packages/services/redis-bridge/src/index.ts`

ECS Fargate service that: - Subscribes to Redis pub/sub channels - Broadcasts events to WebSocket connections via API Gateway Management API

**Channels:**

```
decision_pending:{tenant_id}   → New decision created
decision_resolved:{tenant_id}  → Decision resolved
decision_expired:{tenant_id}   → Decision timed out
swarm_event:{tenant_id}        → Swarm execution events
```

---

### Phase 9: Timeout Cleanup Lambda

**File:** `packages/infrastructure/lambda/functions/timeout-cleanup/index.ts`

Scheduled function (every 1 minute): 1. Find expired decisions 2. Update status to 'expired' 3. Create audit record 4. Publish Redis event 5. Escalate via PagerDuty/Slack (medical/financial = PagerDuty) 6. Abort Flyte execution 7. Update linked Cato escalation

---

### Phase 10: Decision UI Components

**Files:** - `apps/admin-dashboard/src/components/decisions/DecisionSidebar.tsx` - `apps/admin-dashboard/src/components/decisions/DecisionDetail.tsx`

Features: - Live countdown timer with color-coded urgency - Domain-specific badges ( Medical, Financial, Legal) - Resolution form (Approve/Modify/Reject) - Audit trail display - Real-time updates via WebSocket

---

### Phase 11: Supporting Components

**Files:** - `apps/admin-dashboard/src/hooks/useWebSocket.ts` - WebSocket hook with reconnect - `apps/admin-dashboard/src/lib/api/mission-control.ts` - Type-safe API client

---

### Phase 12: Cato Integration

**File:** `packages/infrastructure/lambda/shared/services/cato/hitl-integration.service.ts`

When Cato's Epistemic Recovery fails after max attempts:

```
if (recoveryAttempts >= MAX_RECOVERY_ATTEMPTS) {
  const result = await catoHitlIntegration.createCatoEscalationWithHitl({
    tenantId,
    sessionId,
    userId,
    originalTask,
    rejectionHistory,
    recoveryAttempts,
    lastError,
    flyteExecutionId,
    flyteNodeId,
    context,
  });
}
```

---

### Phase 13: Integration Tests

**File:** `packages/infrastructure/__tests__/mission-control.test.ts`

Test coverage for: - RadiantSwarm parallel execution - FlyteLauncher workflow management - Cato HITL integration - Signal name matching - Domain configuration

---

### Phase 14: Deployment Scripts

**Files:** - `packages/flyte/scripts/register-workflows.sh` - Flyte workflow registration - `tools/scripts/deploy-mission-control.sh` - End-to-end deployment

---

### Phase 15: Documentation

Update admin guides: - `docs/RADIANT-ADMIN-GUIDE.md` - Add Section 48: Mission Control HITL - `docs/THINKTANK-ADMIN-GUIDE.md` - Add Section 32: Swarm Orchestration & Flyte Operations

---

### Verification Checklist

Before marking implementation complete:

- ☐ All 4 database tables created with RLS policies
- ☐ CDK stack deploys without errors
- ☐ RadiantSwarm executes ALL agents (not just first)
- ☐ FlyteLauncher uploads to S3 Bronze layer
- ☐ Python workflow uses `@dynamic(cache=False)`
- ☐ Signal name format is `human_decision_{decision_id}`
- ☐ Mission Control API uses `SET app.tenant_id` (not transaction-level)
- ☐ WebSocket connections stored in both Redis and DB
- ☐ Redis Bridge broadcasts to WebSocket clients
- ☐ Timeout Lambda escalates to PagerDuty for medical/financial
- ☐ UI components have live countdown timers
- ☐ Cato integration creates HITL decisions on epistemic recovery failure
- ☐ Tests pass for signal name matching
- ☐ Documentation updated in both admin guides
- ☐ No TODO comments or placeholder code

---

### Environment Variables Reference

| Variable | Description |
|---|---|
| `DB_SECRET_ARN` | Secrets Manager ARN for database credentials |
| `REDIS_HOST` | Redis/ElastiCache host |
| `REDIS_PORT` | Redis port (default: 6379) |
| `FLYTE_ADMIN_URL` | Flyte Admin API URL |
| `WEBSOCKET_API_ENDPOINT` | WebSocket API Gateway endpoint |
| `PAGERDUTY_ROUTING_KEY` | PagerDuty Events API v2 routing key |
| `SLACK_WEBHOOK_URL` | Slack Incoming Webhook URL |

---

### File Summary

| Phase | File | Lines |
|---|---|---|
| 1 | `migrations/V2026_01_07_001__mission_control_schema.sql` | ~360 |
| 2 | `lib/stacks/mission-control-stack.ts` | ~300 |
| 3 | `lambda/shared/services/swarm/radiant-swarm.ts` | ~460 |

| Phase | File | Lines |
|---|---|---|
| 4 | lambda/shared/services/swarm/flyte-launcher.ts | ~300 |
| 5 | packages/flyte/workflows/think_tank_workflow.py | ~500 |
| 6 | lambda/functions/mission-control/index.ts | ~520 |
| 7 | lambda/functions/websocket-connection-handler.ts | ~300 |
| 8 | packages/services/redis-bridge/src/index.ts | ~220 |
| 9 | lambda/functions/timeout-cleanup/index.ts | ~280 |
| 10 | components/decisions/DecisionSidebar.tsx | ~300 |
| 10 | components/decisions/DecisionDetail.tsx | ~340 |
| 11 | hooks/useWebSocket.ts | ~130 |
| 11 | lib/api/mission-control.ts | ~140 |
| 12 | lambda/shared/services/cato/itl-integration.service.ts | ~220 |
| 13 | __tests__/mission-control.test.ts | ~400 |
| 14 | packages/flyte/scripts/register-workflows.sh | ~90 |
| 14 | tools/scripts/deploy-mission-control.sh | ~180 |

**Total:** ~5,040 lines of production-ready code

---

### Cross-AI Validated Fixes

These issues were identified and fixed during implementation:

1. **Signal Mismatch Fix**: Signal ID now uses `decision_id`, not `agent_id`
2. **Zombie Cache Fix**: Added `@dynamic(cache=False)` to swarm execution
3. **S3 Offloading**: Input payloads uploaded to S3 Bronze layer
4. **RLS Session Fix**: Using session-level `SET app.tenant_id`, not transaction-level
5. **True Swarm Loop**: Iterating all agents, not just `agents[0]`
6. **PHI Sanitization**: All decision content sanitized before human review

---

*Implementation Status: COMPLETE*