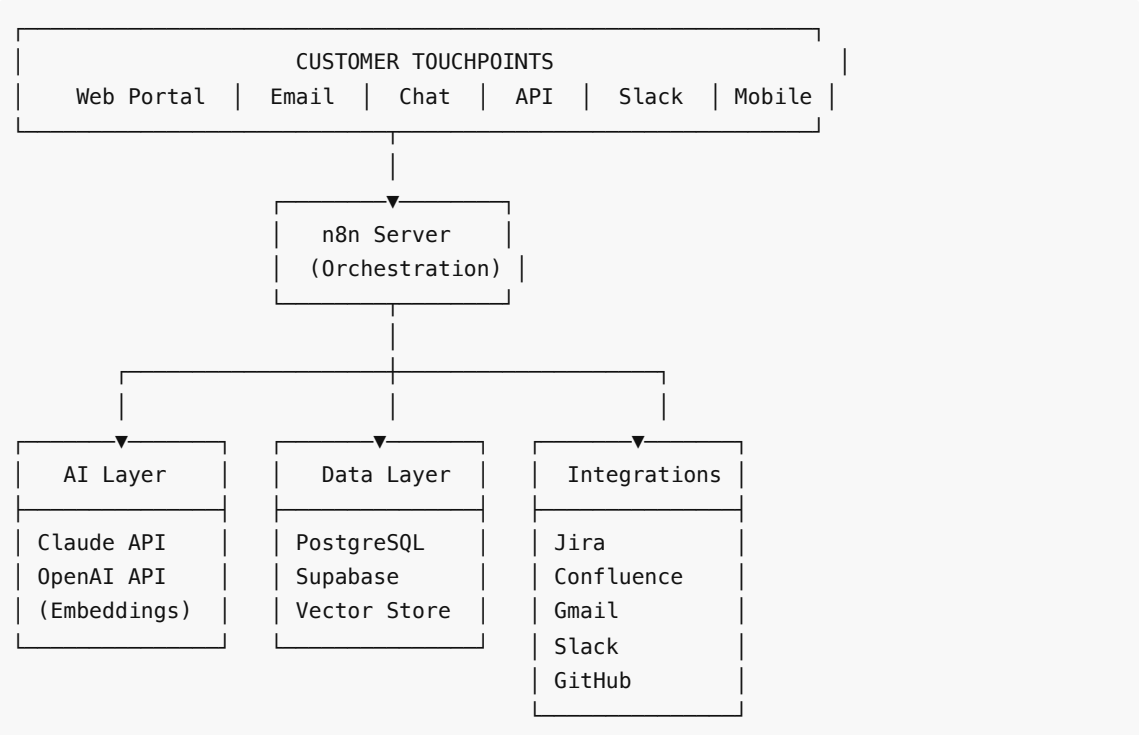
CX-Catalyst - Administrator Guide

Complete setup, configuration, and maintenance guide for system administrators.

Table of Contents

Architecture Overview

System Components



Component Requirements

Component	Minimum	Recommended
n8n	v1.0+	v1.5+ (latest)
PostgreSQL	14+	15+
Node.js	18+	20+ LTS
Memory	2GB	4GB+
Storage	20GB	50GB+

Installation Requirements

Infrastructure Requirements

Component	Minimum	Recommended
n8n	v1.0+	v1.5+ (latest)
PostgreSQL	14+ with pgvector	15+ with pgvector
Node.js	18+	20+ LTS
Memory	2GB	4GB+
Storage	20GB	50GB+

Required API Credentials

Service	Purpose	Required
Anthropic (Claude)	AI classification and solution generation	Yes
OpenAI	Vector embeddings (text-embedding-3-small)	Yes
Supabase	PostgreSQL database with pgvector	Yes
Confluence	Knowledge base wiki	Yes
Slack	Team notifications and review queue	Yes
Gmail	Email support channel	Optional
Jira	Bug tracking and escalations	Optional
Grafana	Monitoring dashboard	Optional

Security Prerequisites

- HTTPS enabled for all endpoints
- API keys stored in n8n credentials store (never in workflow JSON or code)
- Network access between n8n, Supabase, and external APIs
- Webhook endpoints accessible from customer-facing channels

Installation

Option 1: n8n Cloud (Recommended)

1. Sign up at n8n.io
2. Create workspace
3. Skip to Credential Management

Option 2: Self-Hosted (Docker)

```
# Create directory
mkdir n8n-support && cd n8n-support

# Create docker-compose.yml
cat > docker-compose.yml << 'EOF'
version: '3.8'

services:
  n8n:
    image: n8nio/n8n:latest
    restart: always
    ports:
      - "5678:5678"
    environment:
      - N8N_HOST=your-domain.com
      - N8N_PORT=5678
      - N8N_PROTOCOL=https
      - NODE_ENV=production
      - WEBHOOK_URL=https://your-domain.com/
      - N8N_ENCRYPTION_KEY=${N8N_ENCRYPTION_KEY}
      - DB_TYPE=postgresdb
      - DB_POSTGRESDB_HOST=${DB_HOST}
      - DB_POSTGRESDB_PORT=5432
      - DB_POSTGRESDB_DATABASE=n8n
      - DB_POSTGRESDB_USER=${DB_USER}
      - DB_POSTGRESDB_PASSWORD=${DB_PASSWORD}
    volumes:
      - n8n_data:/home/node/.n8n

volumes:
  n8n_data:
EOF

# Create .env file
cat > .env << 'EOF'
N8N_ENCRYPTION_KEY=your-32-char-encryption-key-here
DB_HOST=your-postgres-host
DB_USER=n8n
DB_PASSWORD=secure-password-here
EOF
```

```
# Start n8n
docker-compose up -d
```

Option 3: Self-Hosted (npm)

```
# Install n8n globally
npm install n8n -g

# Set environment variables
export N8N_HOST=your-domain.com
export N8N_PROTOCOL=https
export DB_TYPE=postgresdb
export DB_POSTGRESDB_HOST=your-db-host
export DB_POSTGRESDB_DATABASE=n8n
export DB_POSTGRESDB_USER=n8n
export DB_POSTGRESDB_PASSWORD=secure-password

# Start n8n
n8n start
```

Database Setup

Create Application Database

Connect to PostgreSQL and run:

```
-- Create database for support system
CREATE DATABASE support_system;

-- Connect to database
\c support_system;

-- Enable UUID extension
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
```

Run Schema Migration

Execute the full schema from the design document. Key tables:

```
-- Customers table
CREATE TABLE customers (
    customer_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    name VARCHAR(255),
    account_tier VARCHAR(50) DEFAULT 'standard',
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);
```

```sql
-- Cases table
CREATE TABLE cases (
    case_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    customer_id UUID REFERENCES customers(customer_id),
    channel VARCHAR(50) NOT NULL,
    description TEXT NOT NULL,
    category VARCHAR(100),
    subcategory VARCHAR(100),
    priority VARCHAR(20),
    confidence_score DECIMAL(3,2),
    status VARCHAR(50) DEFAULT 'new',
    resolution_type VARCHAR(50),
    escalated BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW(),
    triaged_at TIMESTAMP,
    resolved_at TIMESTAMP,
    closed_at TIMESTAMP,
    resolution_time_minutes INTEGER,
    satisfaction_score INTEGER,
    metadata JSONB DEFAULT '{}'
);

-- Case interactions
CREATE TABLE case_interactions (
    interaction_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    case_id UUID REFERENCES cases(case_id) ON DELETE CASCADE,
    actor_type VARCHAR(50),
    actor_id VARCHAR(255),
    message TEXT,
    interaction_type VARCHAR(50),
    created_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

-- Knowledge base articles
CREATE TABLE kb_articles (
    article_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    title VARCHAR(500) NOT NULL,
    category VARCHAR(100),
    subcategory VARCHAR(100),
    tags TEXT[],
    auto_generated BOOLEAN DEFAULT FALSE,
    human_validated BOOLEAN DEFAULT FALSE,
    view_count INTEGER DEFAULT 0,
    success_count INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW(),
    vector_store_id VARCHAR(255),
    confluence_page_id VARCHAR(255),
    metadata JSONB DEFAULT '{}'
);
```

```sql
-- Error codes
CREATE TABLE error_codes (
    error_code VARCHAR(100) PRIMARY KEY,
    product VARCHAR(100),
    description TEXT,
    severity VARCHAR(20),
    diagnostic_steps TEXT[],
    resolution_steps TEXT[],
    automated_fix_available BOOLEAN DEFAULT FALSE,
    automation_script TEXT,
    kb_article_id UUID REFERENCES kb_articles(article_id),
    occurrence_count INTEGER DEFAULT 0,
    last_seen TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

-- Health metrics
CREATE TABLE health_metrics (
    metric_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    metric_name VARCHAR(100),
    metric_value DECIMAL,
    unit VARCHAR(50),
    threshold_warning DECIMAL,
    threshold_critical DECIMAL,
    status VARCHAR(20),
    recorded_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

-- Proactive alerts
CREATE TABLE proactive_alerts (
    alert_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    alert_type VARCHAR(100),
    severity VARCHAR(20),
    description TEXT,
    affected_customers UUID[],
    root_cause_hypothesis TEXT,
    actions_taken TEXT[],
    prevented_cases_estimate INTEGER,
    jira_ticket_id VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW(),
    resolved_at TIMESTAMP,
    metadata JSONB DEFAULT '{}'
);

-- Review queue
CREATE TABLE review_queue (
    review_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    case_id UUID REFERENCES cases(case_id),
    ai_solution TEXT,
```

```
    ai_confidence DECIMAL(3,2),
    status VARCHAR(50) DEFAULT 'pending',
    assigned_to VARCHAR(255),
    created_at TIMESTAMP DEFAULT NOW(),
    reviewed_at TIMESTAMP,
    reviewer_comments TEXT,
    final_solution TEXT,
    metadata JSONB DEFAULT '{}'
);

-- Workflow executions
CREATE TABLE workflow_executions (
    execution_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    workflow_name VARCHAR(255),
    case_id UUID REFERENCES cases(case_id),
    status VARCHAR(50),
    start_time TIMESTAMP DEFAULT NOW(),
    end_time TIMESTAMP,
    duration_ms INTEGER,
    error_message TEXT,
    metadata JSONB DEFAULT '{}'
);

-- Agent feedback
CREATE TABLE agent_feedback (
    feedback_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    case_id UUID REFERENCES cases(case_id),
    agent_name VARCHAR(100),
    agent_output TEXT,
    human_correction TEXT,
    feedback_type VARCHAR(50),
    created_at TIMESTAMP DEFAULT NOW(),
    metadata JSONB DEFAULT '{}'
);

-- Create indexes
CREATE INDEX idx_customers_email ON customers(email);
CREATE INDEX idx_customers_tier ON customers(account_tier);
CREATE INDEX idx_cases_customer ON cases(customer_id);
CREATE INDEX idx_cases_status ON cases(status);
CREATE INDEX idx_cases_category ON cases(category);
CREATE INDEX idx_cases_created_at ON cases(created_at DESC);
CREATE INDEX idx_cases_priority ON cases(priority);
CREATE INDEX idx_interactions_case ON case_interactions(case_id);
CREATE INDEX idx_kb_category ON kb_articles(category);
CREATE INDEX idx_kb_tags ON kb_articles USING GIN(tags);
CREATE INDEX idx_error_codes_product ON error_codes(product);
CREATE INDEX idx_health_metrics_name ON health_metrics(metric_name);
CREATE INDEX idx_alerts_created_at ON proactive_alerts(created_at DESC);
CREATE INDEX idx_review_queue_status ON review_queue(status);
```

```
CREATE INDEX idx_workflow_exec_workflow ON workflow_executions(workflow_name);
CREATE INDEX idx_agent_feedback_agent ON agent_feedback(agent_name);
```

Verify Setup

```
-- Check tables exist
SELECT table_name FROM information_schema.tables
WHERE table_schema = 'public';

-- Insert test customer
INSERT INTO customers (email, name, account_tier)
VALUES ('test@example.com', 'Test User', 'standard')
RETURNING customer_id;
```

---

n8n Configuration

Environment Variables

Set these in n8n settings or environment:

```
# Core Settings
N8N_HOST=your-domain.com
N8N_PORT=5678
N8N_PROTOCOL=https
N8N_ENCRYPTION_KEY=your-32-character-encryption-key

# Webhook Configuration
WEBHOOK_URL=https://your-domain.com/
N8N_WEBHOOK_BASE_URL=https://your-domain.com

# Database (for n8n itself)
DB_TYPE=postgresdb
DB_POSTGRESDB_HOST=db-host
DB_POSTGRESDB_PORT=5432
DB_POSTGRESDB_DATABASE=n8n
DB_POSTGRESDB_USER=n8n_user
DB_POSTGRESDB_PASSWORD=n8n_password
DB_POSTGRESDB_SCHEMA=public

# Execution Settings
EXECUTIONS_TIMEOUT=3600
EXECUTIONS_TIMEOUT_MAX=7200

# Application-Specific
JIRA_CLOUD_ID=your-jira-cloud-id
GITHUB_REPO=your-org/your-repo
LOG_AGGREGATOR_URL=http://logs:9200
LEADERSHIP_EMAIL=leadership@company.com
SUPPORT_TEAM_EMAIL=support@company.com
```

```
# Thresholds
CONFIDENCE_THRESHOLD_HIGH=0.85
CONFIDENCE_THRESHOLD_MEDIUM=0.60
REVIEW_TIMEOUT_HOURS=2
```

Workflow Settings

In n8n Settings > Workflow:

- **Timezone:** Set to your operational timezone
- **Execution Timeout:** 3600 seconds (1 hour)
- **Error Workflow:** Create and assign error handler

Credential Management

Required Credentials

Create these in n8n (**Settings** > **Credentials**):

1. PostgreSQL (Support Database)

- **Name:** `Support Database`
- **Host:** your-db-host
- **Database:** support_system
- **User:** support_user
- **Password:** secure-password
- **SSL:** Enable for production

2. Anthropic API

- **Name:** `Anthropic API`
- **API Key:** sk-ant-api03-...

Get key from: console.anthropic.com

3. OpenAI API

- **Name:** `OpenAI API`
- **API Key:** sk-...

Get key from: platform.openai.com

4. Supabase

- **Name:** `Supabase Vector Store`
- **Project URL:** https://xxx.supabase.co
- **Service Role Key:** eyJ...

Get from: Supabase Dashboard > Settings > API

5. Slack

- **Name:** `Slack Bot`
- **Type:** OAuth2
- **Client ID:** from Slack app
- **Client Secret:** from Slack app

- **Scopes:** chat:write, channels:read, reactions:write

Create app at: api.slack.com/apps

6. Gmail

- **Name:** `Support Gmail`
- **Type:** OAuth2
- **Client ID:** from Google Cloud
- **Client Secret:** from Google Cloud
- **Scopes:** gmail.send, gmail.readonly

Create credentials at: Google Cloud Console

7. Jira (Optional)

For Atlassian MCP integration:

- Cloud ID from Jira admin
- API token from id.atlassian.com

8. Confluence HTTP Basic Auth

For Workflow 5 KB article creation:

- **Name:** `Confluence HTTP Basic Auth`
- **Type:** HTTP Basic Auth
- **User:** Your Atlassian email address
- **Password:** API token from id.atlassian.com

Note: This is used by HTTP Request nodes for Confluence REST API calls (not the dedicated Confluence nodes).

Workflow Deployment

Import Order

Deploy workflows in this order:

1. **workflow-1-smart-intake-triage.json** (Foundation)
2. **workflow-2-self-service-resolution.json** (Depends on 1)
3. **workflow-4-collaborative-support.json** (Depends on 1)
4. **workflow-3-proactive-detection.json** (Independent)
5. **workflow-5-continuous-learning.json** (Depends on all)

Import Process

```
# Using n8n CLI (if available)
n8n import:workflow --input=workflow-1-smart-intake-triage.json

# Or via UI:
# 1. Workflows > Import from File
# 2. Select JSON file
# 3. Click Import
```

Post-Import Configuration

For each workflow:

1. **Assign Credentials**
    - Click nodes with red warning icons
    - Select appropriate credential
    - Save
2. **Update Environment References**
    - Check Code nodes for hardcoded values
    - Replace with `$env.VARIABLE_NAME`
3. **Verify Webhook Paths**
    - Ensure unique paths per workflow
    - Note production URLs after activation
4. **Test Before Activation**
    - Use "Execute Workflow" with test data
    - Verify each node succeeds
    - Check database writes

Activation Sequence

1. Activate Workflow 1 first
2. Test with sample request
3. Activate Workflow 2
4. Test self-service path
5. Continue with remaining workflows

Integration Setup

Slack Integration

Create Slack App

1. Go to api.slack.com/apps
2. Click "Create New App"
3. Choose "From scratch"
4. Name: "Support AI Bot"

Configure Permissions

OAuth & Permissions > Scopes:

- `chat:write`
- `chat:write.public`
- `channels:read`
- `reactions:write`
- `reactions:read`

Install to Workspace

1. Click "Install to Workspace"

2. Authorize the app
3. Copy Bot Token (xoxb-...)

Create Channels

```
#support-review    - Case review queue
#support-alerts    - Critical alerts
#support-metrics   - Daily reports
#support-general   - Team discussion
```

Gmail Integration

Create OAuth Credentials

1. Go to Google Cloud Console
2. Create project or select existing
3. Enable Gmail API
4. Create OAuth 2.0 credentials
5. Set authorized redirect URI: `https://your-n8n.com/rest/oauth2-credential/callback`

Configure in n8n

1. Create Gmail credential
2. Enter Client ID and Secret
3. Click "Connect" to authorize
4. Select the support email account

Jira Integration

Get Cloud ID

```
curl -u email:api_token \
  https://your-domain.atlassian.net/_edge/tenant_info
```

Create API Token

1. Go to id.atlassian.com
2. Create API token
3. Store securely

Configure Project

1. Create project key: `SUP` (Support)
2. Create issue types: Bug, Task, Incident
3. Add custom field: "Support Case ID"

Vector Store Configuration

Supabase Setup

Enable pgvector

In Supabase SQL Editor:

```sql
-- Enable vector extension
CREATE EXTENSION IF NOT EXISTS vector;

-- Create documents table
CREATE TABLE documents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    content TEXT,
    metadata JSONB,
    embedding VECTOR(1536)
);

-- Create similarity search function
CREATE OR REPLACE FUNCTION match_documents (
    query_embedding VECTOR(1536),
    match_threshold FLOAT,
    match_count INT
)
RETURNS TABLE (
    id UUID,
    content TEXT,
    metadata JSONB,
    similarity FLOAT
)
LANGUAGE SQL STABLE
AS $$
    SELECT
        documents.id,
        documents.content,
        documents.metadata,
        1 - (documents.embedding <=> query_embedding) AS similarity
    FROM documents
    WHERE 1 - (documents.embedding <=> query_embedding) > match_threshold
    ORDER BY documents.embedding <=> query_embedding
    LIMIT match_count;
$$;

-- Create index for faster search
CREATE INDEX ON documents USING ivfflat (embedding vector_cosine_ops)
WITH (lists = 100);
```

Load Initial KB Content

```javascript
// Example: Add KB article to vector store
const { createClient } = require('@supabase/supabase-js');
const OpenAI = require('openai');

const supabase = createClient(
    process.env.SUPABASE_URL,
    process.env.SUPABASE_SERVICE_KEY
);
```

```
const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });

async function addKBArticle(title, content, category) {
    // Generate embedding
    const response = await openai.embeddings.create({
        model: 'text-embedding-3-large',
        input: content
    });

    const embedding = response.data[0].embedding;

    // Insert document
    await supabase.from('documents').insert({
        content: content,
        metadata: {
            title,
            category,
            created_at: new Date().toISOString()
        },
        embedding
    });
}
```

Analytics & Reporting

Dashboard Overview

CX-Catalyst provides monitoring and analytics through multiple channels:

- **Grafana Dashboard** - Real-time metrics and visualization
- **Confluence Reports** - Daily insights posted to wiki
- **Slack (#support-metrics)** - Daily summary notifications
- **Email Reports** - Leadership reports generated by Workflow 5

Key Metrics

Case Metrics

Metric	Description	Source
Cases Received	Total volume by period	`cases` table
Resolution Rate	% of cases resolved	`cases.status`
Self-Service Rate	% resolved without human intervention	`cases.resolution_type`
Avg Resolution Time	Time from creation to resolution	`cases.resolution_time_minutes`
Satisfaction Score	Average customer rating (1-5)	`cases.satisfaction_score`

AI Performance Metrics

Metric	Description	Target

Classification Accuracy	Correct category assignment	85%+
Confidence Distribution	High/Medium/Low breakdown	60%+ high confidence
Self-Service Success	Auto-resolved without escalation	85%+ for simple issues
KB Match Rate	Issues with relevant KB article	70%+

Token Usage

Workflow	Avg Tokens/Execution	Model
WF1: Intake & Triage	~1,000	Claude Sonnet
WF2: Self-Service	~3,000	Claude Sonnet
WF4: Collaborative	~4,000	Claude Sonnet
WF5: Learning	~10,000	Claude Sonnet
Embedding Generation	~500	text-embedding-3-small

Monitor actual usage via the Grafana dashboard or the token tracking workflow in the `dashboard/` directory.

Custom Reports

SQL Queries for Common Reports

```
-- Daily case volume (last 30 days)
SELECT DATE(created_at) as date, COUNT(*) as cases
FROM cases
WHERE created_at > NOW() - INTERVAL '30 days'
GROUP BY DATE(created_at)
ORDER BY date;

-- Resolution rate by confidence tier
SELECT
    CASE
        WHEN confidence_score >= 0.85 THEN 'High (85-100%)'
        WHEN confidence_score >= 0.60 THEN 'Medium (60-84%)'
        ELSE 'Low (<60%)'
    END as tier,
    COUNT(*) as total,
    COUNT(CASE WHEN status = 'resolved' THEN 1 END) as resolved,
    ROUND(AVG(resolution_time_minutes), 1) as avg_minutes
FROM cases
WHERE created_at > NOW() - INTERVAL '30 days'
GROUP BY tier;

-- KB article effectiveness
SELECT title,
       COUNT(*) as times_retrieved,
```

```
       AVG(c.satisfaction_score) as avg_satisfaction
FROM confluence_kb kb
JOIN cases c ON c.metadata->>'kb_article_used' = kb.page_id
WHERE c.created_at > NOW() - INTERVAL '30 days'
GROUP BY title
ORDER BY times_retrieved DESC
LIMIT 20;
```

Grafana Dashboard Setup

See `dashboard/README.md` for complete setup instructions. The Grafana dashboard provides:

- **Case Volume Panel** - Time series of incoming cases
- **Resolution Breakdown** - Pie chart of resolution types
- **Response Time Histogram** - Distribution of resolution times
- **Token Usage Tracking** - API cost monitoring by workflow
- **System Health** - Workflow execution success rates

Automated Reports (Workflow 5)

Workflow 5 generates daily reports including:

- Top recurring issues and trends
- Knowledge base gap identification
- Bug pattern detection
- AI performance analysis
- Process improvement recommendations

Reports are delivered via:

- Email (HTML formatted) to leadership
- Slack (#support-metrics) as daily summary
- Confluence as wiki page

Monitoring & Logging

n8n Execution Logs

Access at: **Workflows** > **Executions**

Key metrics to monitor:

- Execution success rate
- Average execution time
- Error frequency by workflow
- Queue depth

Custom Monitoring

Create monitoring workflow to track:

```
-- Cases by status (last 24h)
SELECT status, COUNT(*)
FROM cases
```

```sql
WHERE created_at > NOW() - INTERVAL '24 hours'
GROUP BY status;

-- Resolution times by category
SELECT category,
       AVG(resolution_time_minutes) as avg_time,
       COUNT(*) as count
FROM cases
WHERE resolved_at IS NOT NULL
  AND created_at > NOW() - INTERVAL '7 days'
GROUP BY category;

-- AI confidence accuracy
SELECT
    CASE
        WHEN confidence_score >= 0.85 THEN 'high'
        WHEN confidence_score >= 0.60 THEN 'medium'
        ELSE 'low'
    END as confidence_tier,
    AVG(CASE WHEN status = 'resolved' THEN 1 ELSE 0 END) as resolution_rate
FROM cases
WHERE created_at > NOW() - INTERVAL '30 days'
GROUP BY confidence_tier;
```

Alerting

Set up alerts for:

Condition	Threshold	Action
Workflow failure rate	>5%	Slack alert
Avg resolution time	>30 min	Email team lead
Queue depth	>50 pending	Scale warning
AI error rate	>1%	Check API limits
DB connections	>80%	Infrastructure alert

Backup & Recovery

Database Backups

Automated Backups (Supabase)

Enabled by default. Retention:

- Free tier: 7 days
- Pro tier: 30 days

Manual Backup

```
# Full backup
pg_dump -h db-host -U postgres -d support_system > backup_$(date +%Y%m%d).sql

# Backup specific tables
pg_dump -h db-host -U postgres -d support_system -t cases -t customers >
critical_backup.sql
```

Workflow Backups

Export all workflows periodically:

```
# Using n8n CLI
n8n export:workflow --all --output=workflows_backup/

# Or manually export from UI
# Workflows > ... menu > Download
```

Recovery Procedures

Restore Database

```
# Full restore
psql -h db-host -U postgres -d support_system < backup_20260115.sql

# Partial restore (specific tables)
psql -h db-host -U postgres -d support_system < critical_backup.sql
```

Restore Workflows

```
# Import workflow backup
n8n import:workflow --input=workflows_backup/

# Or via UI: Import from File
```

Security

Access Control

n8n User Roles

- **Owner:** Full admin access
- **Admin:** Manage workflows and credentials
- **Member:** Execute workflows, view results

Database Permissions

```
-- Create read-only user for reporting
CREATE USER report_user WITH PASSWORD 'secure-password';
GRANT SELECT ON ALL TABLES IN SCHEMA public TO report_user;
```

```sql
-- Create service user for n8n
CREATE USER n8n_service WITH PASSWORD 'secure-password';
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO n8n_service;
```

API Key Rotation

Rotate keys quarterly:

1. Generate new key in service console
2. Add new credential in n8n (don't delete old)
3. Update workflows to use new credential
4. Test all workflows
5. Delete old credential
6. Revoke old key in service console

Secrets Management

Never store secrets in:

- Workflow JSON files
- Code node contents
- Git repositories

Always use:

- n8n credentials store
- Environment variables
- Secret management services

Network Security

- **HTTPS:** Required for all endpoints
- **IP Allowlisting:** Restrict webhook access if possible
- **Rate Limiting:** Implement at load balancer level
- **WAF:** Consider for public-facing webhooks

Troubleshooting

Common Issues

Webhook Not Responding

1. Check workflow is Active
2. Verify webhook URL is correct
3. Check n8n logs for errors
4. Test with curl: `curl -X POST webhook-url`

Database Connection Failed

1. Verify credentials
2. Check network/firewall rules
3. Confirm database is running
4. Test connection manually: `psql -h host -U user -d database`

AI API Errors

Error	Cause	Solution
401	Invalid API key	Regenerate and update credential
429	Rate limited	Add delays, check quota
500	Service error	Retry, check status page
Timeout	Long request	Increase timeout, simplify prompt

Vector Search Empty Results

1. Verify documents exist in store
2. Check embedding dimensions match (1536)
3. Test query manually in Supabase
4. Verify similarity threshold isn't too high

n8n Expression/JSON Errors

Error	Cause	Solution
"JSON parameter needs to be valid JSON"	Unescaped quotes/newlines in HTTP body	Pre-escape content in Code node (see below)
"Cannot read properties of undefined"	Wrong node reference after data transformation	Use `$('NodeName').first().json` to reference specific upstream node
"syntax error at or near" in SQL	Single quotes in expressions	Use parameterized queries or quadruple-escape quotes
Split In Batches processes all at once	Wrong output connected	Connect "loop" output (index 1) to processing node, not "done" (index 0)
Code node runs multiple times	Parallel inputs without Merge	Add Merge node before Code node for parallel query results

JSON Escaping Pattern (for HTTP Request bodies):

```
// In Code node before HTTP Request
const jsonSafeContent = content
  .replace(/\\/g, '\\\\')
  .replace(/"/g, '\\"')
  .replace(/\n/g, '\\n')
  .replace(/\r/g, '\\r')
  .replace(/\t/g, '\\t');

return { json: { escaped_content: jsonSafeContent } };
```

Then in HTTP Request, use `{{ $json.escaped_content }}` (without `=` prefix).

Confluence API Errors

Error	Cause	Solution
"Space does not exist"	Wrong space key	Verify space key exists (e.g., PKB, CS, ER)
"A page with this title already exists"	Duplicate title in space	Implement upsert logic (DEV-19) or delete existing page
401 Unauthorized	Invalid credentials	Check HTTP Basic Auth: email + API token from id.atlassian.com

Jira API Errors

Error	Cause	Solution
"Project not found"	Wrong project key	Verify project key (e.g., DEV, SUP)
"Issue type not found"	Invalid issue type	Check issue type name matches exactly (Bug, Story, Task)
Field validation errors	Missing required fields	Use getJiraIssueTypeMetaWithFields to check required fields

Debug Mode

Enable detailed logging:

```
# Docker
docker logs n8n-container -f

# npm install
export N8N_LOG_LEVEL=debug
n8n start
```

Support Resources

- n8n Documentation: docs.n8n.io
- n8n Community: community.n8n.io
- Anthropic Docs: docs.anthropic.com
- OpenAI Docs: platform.openai.com/docs

Maintenance

Daily Tasks

- [ ] Review execution logs for errors
- [ ] Check queue depth
- [ ] Verify scheduled workflows ran

Weekly Tasks

- [ ] Review performance metrics
- [ ] Check API usage and costs

- [ ] Clear old execution logs
- [ ] Review and approve KB updates

Monthly Tasks

- [ ] Rotate API keys
- [ ] Review and archive old cases
- [ ] Update agent prompts if needed
- [ ] Analyze escalation patterns
- [ ] Database maintenance (VACUUM, ANALYZE)
- [ ] Review and update documentation

Quarterly Tasks

- [ ] Full system audit
- [ ] Security review
- [ ] Capacity planning
- [ ] Integration health check
- [ ] Disaster recovery test

Appendix: Useful Queries

Case Analytics

```sql
-- Daily case volume
SELECT DATE(created_at), COUNT(*)
FROM cases
WHERE created_at > NOW() - INTERVAL '30 days'
GROUP BY DATE(created_at)
ORDER BY DATE(created_at);

-- Category distribution
SELECT category,
       COUNT(*) as total,
       ROUND(COUNT(*)::numeric / SUM(COUNT(*)) OVER () * 100, 2) as percentage
FROM cases
GROUP BY category
ORDER BY total DESC;

-- Resolution rate by confidence tier
SELECT
    CASE
        WHEN confidence_score >= 0.85 THEN 'High (85-100%)'
        WHEN confidence_score >= 0.60 THEN 'Medium (60-84%)'
        ELSE 'Low (<60%)'
    END as tier,
    COUNT(*) as total,
    COUNT(CASE WHEN status = 'resolved' THEN 1 END) as resolved,
    ROUND(AVG(resolution_time_minutes), 1) as avg_resolution_min
FROM cases
```

```
WHERE created_at > NOW() - INTERVAL '30 days'
GROUP BY tier;
```

System Health

```
-- Execution statistics
SELECT workflow_name,
       COUNT(*) as executions,
       COUNT(CASE WHEN status = 'success' THEN 1 END) as success,
       ROUND(AVG(duration_ms), 0) as avg_duration_ms
FROM workflow_executions
WHERE start_time > NOW() - INTERVAL '24 hours'
GROUP BY workflow_name;

-- Agent performance
SELECT agent_name,
       feedback_type,
       COUNT(*) as count
FROM agent_feedback
WHERE created_at > NOW() - INTERVAL '30 days'
GROUP BY agent_name, feedback_type;
```
