next8n

WORKFLOW STANDARDS GUIDE

Workflow Automation Delivery Framework

ENTERPRISE EDITION

Version: 2.0

Date: December 28, 2025

Author: Mirza Iqbal

Contact: mirza.iqbal@next8n.com

Table of Contents

Workflow Standards Guide

Complete Standards and Best Practices for n8n Workflow Development

Overview

This guide establishes comprehensive standards for building professional, maintainable, and scalable n8n workflows. Following these standards ensures consistency across projects, simplifies handover, and reduces technical debt.

```
+======================================================================+
|                                                                      |
|   "STANDARDS ARE NOT RESTRICTIONS - THEY ARE FOUNDATIONS"           |
|                                                                      |
|   Consistent standards enable faster development, easier            |
|   debugging, smoother handovers, and more reliable workflows.       |
|                                                                      |
+======================================================================+
```

1. Naming Conventions

1.1 Workflow Naming

Pattern: `[Client]-[Function]-[SubFunction]-[Version]`

```
EXAMPLES:

Production Workflows:
- Acme-LeadCapture-EmailSequence-v1
- Acme-OrderProcessing-Fulfillment-v2
- Acme-CustomerSupport-TicketRouting-v1

Sub-Workflows:
- Acme-Sub-DataValidation
- Acme-Sub-ErrorNotification
- Acme-Sub-SlackMessaging

Test/Development:
- Acme-DEV-LeadCapture-Experimental
- Acme-TEST-OrderProcessing-Debug

Templates:
- Template-WebhookHandler-Basic
- Template-AIChat-Standard
```

Naming Rules:

ELEMENT	FORMAT	EXAMPLE
Client Name	PascalCase	Acme, BlueCorp
Function	PascalCase	LeadCapture, OrderProcessing
Sub-Function	PascalCase	EmailSequence, Fulfillment
Version	v + number	v1, v2, v10
Environment	UPPERCASE	DEV, TEST, PROD

Prohibited Naming:

```
BAD EXAMPLES:
- "new workflow"           (non-descriptive)
- "test 2"                 (no context)
- "Copy of workflow"       (lazy naming)
- "workflow_final_FINAL"   (version chaos)
- "johns workflow"         (personal naming)
```

1.2 Node Naming

Pattern: `[Action]-[Target]-[Qualifier]`

```
STANDARD NODE NAMES:

Triggers:
- Webhook-IncomingLead
- Schedule-DailyReport
- Trigger-NewOrder

Data Operations:
- Get-CustomerData
- Set-Variables
- Filter-ActiveUsers
- Sort-ByDate
- Merge-AllResults

Integrations:
- Slack-SendNotification
- Email-SendConfirmation
- Sheets-AppendRow
- CRM-UpdateContact

Conditionals:
- If-IsNewCustomer
- Switch-OrderType
- If-HasEmail

Processing:
- Transform-DataFormat
- Parse-JSONResponse
- Calculate-Totals

AI Nodes:
- AI-GenerateResponse
- AI-ClassifyIntent
- AI-ExtractEntities

Error Handling:
- Catch-APIErrors
- Fallback-DefaultResponse
- Notify-OnFailure
```

Node Naming Table:

NODE TYPE	PREFIX	EXAMPLES
Webhook	Webhook-	Webhook-FormSubmit
Schedule	Schedule-	Schedule-Hourly
HTTP Request	HTTP- or API-	HTTP-GetProducts
Code	Code-	Code-TransformData
If	If-	If-IsValid
Switch	Switch-	Switch-Category
Set	Set-	Set-Defaults
Function	Fn-	Fn-CalculateTotal
AI	AI-	AI-ChatResponse
Slack	Slack-	Slack-PostMessage
Email	Email-	Email-SendWelcome
Database	DB-	DB-InsertRecord

1.3 Credential Naming

Pattern: `[Service]-[Client]-[Purpose]-[Environment]`

```
EXAMPLES:

Production:
- Slack-Acme-Notifications-Prod
- Google-Acme-Sheets-Prod
- Stripe-Acme-Payments-Prod

Development/Testing:
- Slack-Acme-Testing-Dev
- Google-Acme-Sandbox-Dev

Shared Services:
- SMTP-Acme-Transactional
- AI-Acme-ChatGeneration
```

Credential Naming Rules:

```
DO:
- Include client name for multi-client setups
- Indicate environment (Prod/Dev/Test)
- Describe the purpose/scope
- Use consistent capitalization

DON'T:
- Use generic names ("My API Key")
- Include actual credential values
- Use personal identifiers
- Mix naming conventions
```

1.4 Variable Naming

In n8n Expressions and Code Nodes:

```
// Variables (camelCase)
const customerEmail = $json.email;
const orderTotal = calculateTotal(items);
const isNewCustomer = checkCustomerStatus(id);

// Constants (UPPER_SNAKE_CASE)
const MAX_RETRIES = 3;
const API_TIMEOUT = 30000;
const DEFAULT_CURRENCY = 'USD';

// Boolean variables (is/has/can/should prefix)
const isValid = validateInput(data);
const hasPermission = checkAccess(user);
const canProceed = isValid && hasPermission;

// Arrays (plural nouns)
const customers = [];
const orderItems = [];
const errorMessages = [];

// Objects (singular descriptive nouns)
const customerData = {};
const apiResponse = {};
const configOptions = {};
```

1.5 Tag Naming

Pattern: `[Category]:[Value]`

```
STANDARD TAGS:

Client Tags:
- client:acme
- client:bluecorp
- client:internal

Status Tags:
- status:active
- status:deprecated
- status:testing
- status:template

Type Tags:
- type:main-workflow
- type:sub-workflow
- type:utility
- type:scheduled

Integration Tags:
- integration:slack
- integration:google
- integration:crm

Priority Tags:
- priority:critical
- priority:high
- priority:normal
```

2. Workflow Organization

2.1 Folder Structure

Recommended Organization:

```
CLIENT NAME/
|
+-- Production/
|   +-- [Main workflows]
|
+-- Sub-Workflows/
|   +-- [Reusable components]
|
+-- Testing/
|   +-- [Test versions]
|
+-- Deprecated/
|   +-- [Old versions kept for reference]
|
+-- Templates/
    +-- [Reusable patterns]
```

2.2 Node Layout Standards

Visual Organization Rules:

```
LEFT TO RIGHT FLOW:
+--------+      +----------+      +--------+      +---------+
|Trigger | ---> | Process  | ---> | Output | ---> | Notify  |
+--------+      +----------+      +--------+      +---------+

BRANCHING:
                    +-- [Branch A] --+
                   /                  \
+--------+   +-----+                   +-----+   +--------+
|Trigger |-->|If   |                   |Merge|->| Output |
+--------+   +-----+                   +-----+   +--------+
                   \                  /
                    +-- [Branch B] --+

VERTICAL ALIGNMENT:
- Parallel branches aligned vertically
- Main flow on a horizontal line
- Error handling below main flow
```

Spacing Guidelines:

ELEMENT	SPACING
Horizontal gap between nodes	150-200 pixels
Vertical gap for branches	100-150 pixels
Error handling offset	Below main flow
Groups	Clear visual boundaries

2.3 Node Grouping

Use Sticky Notes for Sections:

```
+-----------------------------------------+
| SECTION: Data Validation                |
| Purpose: Validates incoming webhook data |
+-----------------------------------------+
|                                         |
|  [Node] --> [Node] --> [Node]           |
|                                         |
+-----------------------------------------+

+-----------------------------------------+
| SECTION: API Processing                 |
| Purpose: Handles external API calls     |
+-----------------------------------------+
|                                         |
|  [Node] --> [Node] --> [Node]           |
|                                         |
+-----------------------------------------+
```

2.4 Sub-Workflow Usage

When to Create Sub-Workflows:

```
CREATE SUB-WORKFLOW WHEN:
- Logic is used in 3+ workflows
- Process is complex (10+ nodes)
- Component needs independent testing
- Functionality may change independently
- Error handling is specialized

EXAMPLES OF SUB-WORKFLOWS:
- Error notification handler
- Data validation pipeline
- Common API wrapper
- Logging utility
- Rate limiting handler
```

Sub-Workflow Naming:

```
Pattern: [Client]-Sub-[FunctionName]

Examples:
- Acme-Sub-SlackNotification
- Acme-Sub-DataValidation
- Acme-Sub-ErrorHandler
- Acme-Sub-RateLimiter
```

---

3. Node Labeling Standards

3.1 Sticky Note Requirements

Every Workflow Must Have:

```
+======================================================================+
| WORKFLOW: Acme-LeadCapture-EmailSequence-v1                          |
+======================================================================+
| Purpose: Captures leads from website form and sends email sequence  |
|                                                                      |
| Trigger: Webhook from marketing website                              |
| Output: Lead added to CRM, welcome email sent                        |
|                                                                      |
| Dependencies:                                                        |
| - Slack-Acme-Notifications-Prod                                      |
| - Email-Acme-Transactional-Prod                                      |
| - CRM-Acme-Production                                                |
|                                                                      |
| Last Updated: 2024-01-15                                             |
| Author: [Name]                                                       |
+======================================================================+
```

3.2 Section Labels

Required Section Labels:

SECTION	LABEL CONTENT
Input	Data source and format expected
Validation	What is being validated and why
Processing	Business logic description
Output	Where data goes and format
Error Handling	How errors are handled

Example Section Note:

```
+-----------------------------------------+
| SECTION: Input Validation               |
|                                         |
| Validates:                              |
| - Email format (required)               |
| - Phone format (optional)               |
| - Company name present                  |
|                                         |
| Invalid data: Logs error, sends         |
| notification, stops workflow            |
+-----------------------------------------+
```

3.3 Complex Node Annotations

For AI Nodes:

```
+-----------------------------------------+
| AI: Generate Support Response           |
|                                         |
| Model: [Model name]                     |
| Temperature: 0.7                        |
| Max Tokens: 500                         |
|                                         |
| Purpose: Generate helpful customer      |
| response based on ticket content        |
|                                         |
| Fallback: Standard template response    |
+-----------------------------------------+
```

For Code Nodes:

```
+-----------------------------------------+
| CODE: Transform Order Data              |
|                                         |
| Input: Raw order from API               |
| Output: Formatted order object          |
|                                         |
| Logic:                                  |
| 1. Extracts relevant fields             |
| 2. Calculates totals                    |
| 3. Formats for CRM                      |
|                                         |
| Error: Returns empty object with error  |
+-----------------------------------------+
```

4. Documentation Requirements

4.1 Workflow Header Documentation

Required Header Sticky Note:

```
+======================================================================+
| WORKFLOW DOCUMENTATION                                               |
+======================================================================+
|                                                                      |
| Name: [Workflow Name]                                                |
| Version: [v1, v2, etc.]                                              |
| Created: [Date]                                                      |
| Last Modified: [Date]                                                |
| Author: [Name]                                                       |
|                                                                      |
| PURPOSE:                                                             |
| [2-3 sentence description of what this workflow does]                |
|                                                                      |
| TRIGGER:                                                             |
| [How the workflow is triggered]                                      |
|                                                                      |
| INPUTS:                                                              |
| - [Input 1 and format]                                               |
| - [Input 2 and format]                                               |
|                                                                      |
| OUTPUTS:                                                             |
| - [Output 1 and destination]                                         |
| - [Output 2 and destination]                                         |
|                                                                      |
| DEPENDENCIES:                                                        |
| - [Credential 1]                                                     |
| - [Sub-workflow 1]                                                   |
| - [External service 1]                                               |
|                                                                      |
| ERROR HANDLING:                                                      |
| [How errors are handled and who is notified]                         |
|                                                                      |
| CHANGE LOG:                                                          |
| v1.0 - Initial release                                               |
| v1.1 - Added rate limiting                                           |
| v2.0 - Major refactor, new error handling                            |
|                                                                      |
+======================================================================+
```

4.2 External Documentation

Maintain External Documentation:

```
# [Workflow Name] - Technical Documentation

## Overview
[Detailed description]

## Architecture
[Flow diagram or description]

## Data Flow
| Step | Node | Input | Output | Notes |
|------|------|-------|--------|-------|
| 1 | Webhook | HTTP POST | JSON | Entry point |
| 2 | Validate | JSON | JSON/Error | Checks format |

## Configuration
| Setting | Value | Description |
|---------|-------|-------------|
| Timeout | 30s | Max wait time |
| Retries | 3 | Retry count |

## Error Handling
[How errors are handled]

## Monitoring
[How to monitor workflow health]

## Troubleshooting
[Common issues and solutions]
```

4.3 Inline Documentation

In Code Nodes:

```javascript
/**
 * Transform Order Data
 *
 * Converts raw order data from webhook into CRM-compatible format.
 *
 * @param {Object} rawOrder - The incoming order data
 * @returns {Object} - Formatted order for CRM
 *
 * Business Rules:
 * - Orders under $10 are flagged as "small"
 * - International orders get special handling
 * - Missing email triggers alert
 */

const items = $input.all();
const results = [];

// Process each order item
for (const item of items) {
  const order = item.json;

  // Extract and validate customer email
  // Required field - will throw error if missing
  const email = order.customer?.email;
  if (!email) {
    throw new Error('Customer email is required');
  }

  // Calculate order total including tax
  // Tax rate is 8.5% for US orders, 0% international
  const subtotal = order.items.reduce((sum, i) => sum + i.price, 0);
  const taxRate = order.country === 'US' ? 0.085 : 0;
  const total = subtotal * (1 + taxRate);

  results.push({
    email,
    total,
    // ... more fields
  });
}

return results;
```

5. Error Handling Standards

5.1 Error Handling Architecture

```
STANDARD ERROR HANDLING PATTERN:

     +-- [Success Path] --> [Output]
    /
[Node] ---> [Error Branch] --> [Log Error] --> [Notify] --> [Graceful Exit]
    \
     +-- [Retry Logic] ---> [Back to Node]
```

5.2 Required Error Handling

Every Workflow Must Have:

```
MINIMUM ERROR HANDLING:

1. TRY-CATCH WRAPPER
   - All external API calls
   - All code nodes with logic
   - All data transformations

2. ERROR NOTIFICATION
   - Slack/Email alert for failures
   - Include workflow name and execution ID
   - Include error message and context

3. GRACEFUL DEGRADATION
   - Fallback for non-critical failures
   - Continue with partial data when possible
   - Clear indication of degraded state

4. ERROR LOGGING
   - Log all errors to central location
   - Include timestamp, workflow, node, error
   - Searchable and analyzable
```

5.3 Error Message Format

Standard Error Notification:

```
ERROR NOTIFICATION TEMPLATE:

+--------------------------------------------------+
| WORKFLOW ERROR                                   |
+--------------------------------------------------+
| Workflow: [Workflow Name]                        |
| Execution ID: [ID]                               |
| Timestamp: [ISO 8601 format]                     |
| Node: [Node Name]                                |
|                                                  |
| ERROR:                                           |
| [Error message]                                  |
|                                                  |
| CONTEXT:                                         |
| [Relevant data that caused the error]            |
|                                                  |
| ACTION REQUIRED:                                 |
| [What needs to be done]                          |
+--------------------------------------------------+
```

5.4 Retry Logic Standards

When to Implement Retry:

```
ALWAYS RETRY:
- Network timeouts
- Rate limit responses (429)
- Temporary server errors (500, 502, 503)

NEVER RETRY:
- Authentication failures (401, 403)
- Bad request errors (400)
- Not found errors (404)
- Business logic failures

RETRY CONFIGURATION:
- Max retries: 3
- Backoff: Exponential (1s, 2s, 4s)
- Timeout per attempt: 30s
```

Retry Implementation:

```
// Standard retry configuration
const MAX_RETRIES = 3;
const INITIAL_DELAY = 1000; // 1 second

async function withRetry(operation) {
  let lastError;

  for (let attempt = 1; attempt <= MAX_RETRIES; attempt++) {
    try {
      return await operation();
    } catch (error) {
      lastError = error;

      // Don't retry client errors
      if (error.status >= 400 && error.status < 500) {
        throw error;
      }

      // Wait with exponential backoff
      if (attempt < MAX_RETRIES) {
        const delay = INITIAL_DELAY * Math.pow(2, attempt - 1);
        await new Promise(resolve => setTimeout(resolve, delay));
      }
    }
  }

  throw lastError;
}
```

5.5 Error Categories

Classify Errors by Severity:

SEVERITY	RESPONSE	NOTIFICATION
Critical	Stop workflow, immediate alert	Slack + Email + SMS
High	Stop workflow, alert within 5 min	Slack + Email
Medium	Log, continue if possible	Slack
Low	Log only	None

6. Logging Standards

6.1 Logging Architecture

```
LOGGING FLOW:

[Workflow] --> [Log Entry] --> [Log Storage] --> [Analysis]
                   |
                   v
            +-------------+
            | Log Format: |
            | - Timestamp |
            | - Workflow  |
            | - Level     |
            | - Message   |
            | - Data      |
            +-------------+
```

6.2 Log Entry Format

Standard Log Structure:

```
const logEntry = {
  timestamp: new Date().toISOString(),
  level: 'INFO', // DEBUG, INFO, WARN, ERROR
  workflow: $workflow.name,
  executionId: $execution.id,
  node: 'Node-Name',
  message: 'Descriptive message',
  data: {
    // Relevant contextual data
    // NEVER include sensitive information
  },
  duration: endTime - startTime
};
```

6.3 Log Levels

When to Use Each Level:

LEVEL	USE CASE	EXAMPLES
DEBUG	Development details	Variable values, flow paths
INFO	Normal operations	Process started, completed
WARN	Potential issues	Retry triggered, degraded mode
ERROR	Failures	API failure, validation error
FATAL	Critical failures	Cannot continue, data loss risk

6.4 What to Log

```
ALWAYS LOG:
- Workflow start and completion
- External API calls (request/response summary)
- Decision points (which branch taken)
- Error occurrences
- Performance metrics

NEVER LOG:
- Passwords or API keys
- Full credit card numbers
- Personal health information
- Social security numbers
- Full email content (summarize instead)
- Any PII in plain text
```

6.5 Log Storage Options

Recommended Approaches:

```
OPTION 1: Google Sheets (Simple)
- Good for: Small volume, manual review
- Sheet structure: Timestamp | Level | Workflow | Message | Data

OPTION 2: Airtable (Structured)
- Good for: Medium volume, filtering/views
- Benefits: Better querying, relationships

OPTION 3: Database (Scalable)
- Good for: High volume, long retention
- Options: PostgreSQL, MongoDB

OPTION 4: Dedicated Logging Service
- Good for: Enterprise, analysis
- Options: Datadog, Logtail, custom
```

7. Code Style for Code/Function Nodes

7.1 JavaScript Standards

General Code Style:

```javascript
// ============================================================
// FILE HEADER (for complex code nodes)
// Purpose: Transform and validate incoming order data
// Author: [Name]
// Last Updated: [Date]
// ============================================================

// CONSTANTS at the top
const MAX_ORDER_VALUE = 10000;
const DEFAULT_CURRENCY = 'USD';
const VALID_STATUSES = ['pending', 'processing', 'shipped'];

// Main processing logic
const items = $input.all();
const results = [];

for (const item of items) {
  const json = item.json;

  // Input validation with clear error messages
  if (!json.orderId) {
    throw new Error('Missing required field: orderId');
  }

  // Process with clear variable names
  const orderTotal = calculateOrderTotal(json.items);
  const isValidOrder = validateOrder(json);

  // Build result object
  const processedOrder = {
    id: json.orderId,
    total: orderTotal,
    currency: json.currency || DEFAULT_CURRENCY,
    isValid: isValidOrder,
    processedAt: new Date().toISOString()
  };

  results.push(processedOrder);
}

return results;

// ============================================================
// HELPER FUNCTIONS (at bottom, well-documented)
// ============================================================

/**
 * Calculate total order value including discounts
 * @param {Array} items - Order line items
 * @returns {number} - Total order value
 */
```

```
function calculateOrderTotal(items) {
  if (!items || !Array.isArray(items)) {
    return 0;
  }

  return items.reduce((total, item) => {
    const itemTotal = (item.price || 0) * (item.quantity || 1);
    const discount = item.discount || 0;
    return total + (itemTotal - discount);
  }, 0);
}

/**
 * Validate order meets business rules
 * @param {Object} order - The order to validate
 * @returns {boolean} - Whether order is valid
 */
function validateOrder(order) {
  // Must have at least one item
  if (!order.items || order.items.length === 0) {
    return false;
  }

  // Order total must be positive
  const total = calculateOrderTotal(order.items);
  if (total <= 0) {
    return false;
  }

  // Status must be valid
  if (order.status && !VALID_STATUSES.includes(order.status)) {
    return false;
  }

  return true;
}
```

7.2 Python Standards

```python
# ============================================================
# FILE HEADER
# Purpose: Process and analyze customer data
# Author: [Name]
# Last Updated: [Date]
# ============================================================

from typing import Dict, List, Optional
from datetime import datetime
import json

# CONSTANTS
MAX_BATCH_SIZE = 100
DEFAULT_REGION = "US"

def process_customer(customer: Dict) -> Dict:
    """
    Process a single customer record.

    Args:
        customer: Raw customer data dictionary

    Returns:
        Processed customer data with calculated fields

    Raises:
        ValueError: If required fields are missing
    """
    # Validate required fields
    if not customer.get("email"):
        raise ValueError("Customer email is required")

    # Process customer data
    processed = {
        "email": customer["email"].lower().strip(),
        "name": format_name(customer.get("name", "")),
        "region": customer.get("region", DEFAULT_REGION),
        "processed_at": datetime.now().isoformat(),
        "is_active": determine_active_status(customer)
    }

    return processed

def format_name(name: str) -> str:
    """Format customer name to title case."""
    return name.strip().title() if name else "Unknown"

def determine_active_status(customer: Dict) -> bool:
    """Determine if customer is active based on activity."""
    last_order = customer.get("last_order_date")
    if not last_order:
        return False

    # Active if ordered in last 90 days
    last_order_date = datetime.fromisoformat(last_order)
    days_since = (datetime.now() - last_order_date).days

    return days_since <= 90

# Main execution
items = $input.all()
results = []

for item in items:
    try:
        processed = process_customer(item.json)
        results.append(processed)
    except ValueError as e:
        # Log error but continue processing
        results.append({
            "error": str(e),
            "original": item.json
        })

return results
```

7.3 Code Quality Rules

Mandatory Practices:

```
DO:
- Use descriptive variable names
- Add comments for complex logic
- Validate all inputs
- Handle errors explicitly
- Use constants for magic values
- Keep functions small and focused
- Return early for invalid cases

DON'T:
- Use single-letter variable names (except i, j in loops)
- Leave commented-out code
- Use nested ternary operators
- Ignore error cases
- Hardcode values in multiple places
- Write functions over 50 lines
- Mix business logic with data access
```

8. Version Control Practices

8.1 Workflow Versioning

Version Number Format:

```
Major.Minor.Patch (v1.2.3)

MAJOR: Breaking changes, complete redesigns
       v1.0 -> v2.0 (new trigger, different output format)

MINOR: New features, significant improvements
       v1.0 -> v1.1 (added Slack notification)

PATCH: Bug fixes, minor tweaks
       v1.0 -> v1.0.1 (fixed typo in email)
```

8.2 Change Documentation

Changelog Format (in Sticky Note):

```
+======================================================================+
| CHANGE LOG                                                           |
+======================================================================+
|                                                                      |
| v2.1.0 (2024-01-15) - [Name]                                         |
| - Added rate limiting to prevent API overload                        |
| - Improved error messages for validation failures                    |
|                                                                      |
| v2.0.0 (2024-01-10) - [Name]                                         |
| - BREAKING: Changed webhook response format                          |
| - Added support for batch processing                                 |
| - Migrated to new CRM integration                                    |
|                                                                      |
| v1.2.0 (2024-01-05) - [Name]                                         |
| - Added Slack notifications for errors                               |
| - Fixed timezone handling bug                                        |
|                                                                      |
| v1.1.0 (2024-01-01) - [Name]                                         |
| - Added email validation                                             |
|                                                                      |
| v1.0.0 (2023-12-15) - [Name]                                         |
| - Initial release                                                    |
|                                                                      |
+======================================================================+
```

8.3 Branching Strategy

```
WORKFLOW VERSIONING APPROACH:

Production (Active):
  Acme-OrderProcessing-v2 [ACTIVE - DO NOT MODIFY]

Development (New Version):
  Acme-DEV-OrderProcessing-v3 [IN DEVELOPMENT]

Testing:
  Acme-TEST-OrderProcessing-v3 [TESTING]

Archive:
  Acme-OrderProcessing-v1 [DEPRECATED - ARCHIVED]
```

8.4 Pre-Change Checklist

```
BEFORE MODIFYING A PRODUCTION WORKFLOW:

[ ] Create duplicate for development
[ ] Name duplicate with DEV prefix
[ ] Test changes in development version
[ ] Document changes in changelog
[ ] Increment version number appropriately
[ ] Get approval for production deployment
[ ] Schedule maintenance window if needed
[ ] Have rollback plan ready
```

9. Backup and Export Procedures

9.1 Export Standards

Workflow Export Format:

```
EXPORT NAMING CONVENTION:
[WorkflowName]_[Version]_[Date]_[Environment].json

EXAMPLES:
Acme-LeadCapture-v2_2024-01-15_prod.json
Acme-OrderProcessing-v3_2024-01-15_dev.json
```

What to Export:

```
WORKFLOW EXPORT:
[ ] Main workflow JSON
[ ] Sub-workflows used
[ ] Credential requirements list (NOT values)
[ ] Environment variables list
[ ] Configuration documentation
```

9.2 Backup Schedule

Recommended Backup Cadence:

TYPE	FREQUENCY	RETENTION
Production workflows	Weekly	12 months
After major changes	Immediately	Permanent
Before deployments	Pre-deployment	3 months
Full environment	Monthly	6 months

9.3 Backup Storage

Storage Location Standards:

```
BACKUP STRUCTURE:

/backups/
|
+-- /[client-name]/
|   |
|   +-- /production/
|   |   +-- weekly/
|   |   +-- pre-deployment/
|   |
|   +-- /archived/
|   |   +-- [deprecated workflows]
|   |
|   +-- /documentation/
       +-- [workflow docs]

STORAGE OPTIONS:
- Cloud storage (Google Drive, Dropbox, S3)
- Git repository (for JSON files)
- Client's document management system
```

9.4 Recovery Procedures

Workflow Recovery Steps:

```
TO RESTORE A WORKFLOW:

1. Locate correct backup file
   - Check version matches needed state
   - Verify date is appropriate

2. Import to n8n
   - Create new workflow from import
   - Do NOT overwrite active workflow

3. Reconfigure credentials
   - Credentials are not exported
   - Re-link all credential nodes

4. Test thoroughly
   - Run with test data
   - Verify all integrations work

5. Switch over
   - Deactivate old workflow
   - Activate restored workflow
   - Monitor closely
```

10. Performance Optimization Guidelines

10.1 Performance Principles

```
+======================================================================+
|                                                                      |
|   "OPTIMIZE FOR CLARITY FIRST, THEN FOR SPEED"                       |
|                                                                      |
|   Only optimize when there's a measured performance problem.        |
|   Premature optimization leads to unmaintainable workflows.          |
|                                                                      |
+======================================================================+
```

10.2 Performance Targets

METRIC	TARGET	ACCEPTABLE	ACTION NEEDED
Total execution time	< 30s	< 60s	> 60s
Individual API call	< 10s	< 20s	> 20s
Memory usage	< 256MB	< 512MB	> 512MB
Execution queue wait	< 5s	< 15s	> 15s

10.3 Optimization Techniques

Data Handling:

```
// BAD: Processing all data when only some needed
const allCustomers = await getAllCustomers(); // 10,000 records
const activeCustomers = allCustomers.filter(c => c.active);

// GOOD: Filter at source
const activeCustomers = await getCustomers({ status: 'active' }); // 500 records
```

API Call Optimization:

```
REDUCE API CALLS:

1. BATCH OPERATIONS
   - Combine multiple creates into batch
   - Use bulk update endpoints

2. CACHING
   - Cache frequently accessed data
   - Set appropriate TTL

3. PAGINATION
   - Process large datasets in pages
   - Don't load everything at once

4. SELECTIVE FIELDS
   - Request only needed fields
   - Avoid fetching full records
```

Parallel Processing:

```
WHEN TO PARALLELIZE:

Good candidates:
- Independent API calls
- Processing separate records
- Notifications to different services

NOT parallel:
- Dependent operations
- Sequential logic
- Rate-limited APIs
```

10.4 Memory Management

```
// BAD: Keeping all data in memory
const allResults = [];
for (const batch of batches) {
  const batchResults = await processBatch(batch);
  allResults.push(...batchResults);
}
// allResults now has everything in memory

// GOOD: Process and output in chunks
for (const batch of batches) {
  const batchResults = await processBatch(batch);
  await sendToDestination(batchResults);
  // Results are written, memory freed
}
```

10.5 Performance Monitoring

What to Monitor:

```
TRACK THESE METRICS:

1. Execution duration over time
2. Success/failure rate
3. Queue wait times
4. API response times
5. Memory consumption (if available)

RED FLAGS:
- Execution time increasing over time
- More retries needed
- Frequent timeouts
- Memory errors
```

11. Pre-Deployment Checklist

11.1 Development Complete Checklist

```
BEFORE REQUESTING CODE REVIEW:

NAMING:
[ ] Workflow name follows convention
[ ] All nodes named descriptively
[ ] Variables follow naming standards
[ ] No default/placeholder names remain

DOCUMENTATION:
[ ] Workflow header documentation complete
[ ] All sections have sticky notes
[ ] Complex nodes annotated
[ ] Code comments in place

ERROR HANDLING:
[ ] All API calls have error handling
[ ] Error notifications configured
[ ] Retry logic implemented where needed
[ ] Fallbacks defined for non-critical paths

TESTING:
[ ] Unit tested each node
[ ] End-to-end test passed
[ ] Edge cases tested
[ ] Error scenarios tested

CODE QUALITY:
[ ] No hardcoded credentials
[ ] No commented-out code
[ ] No console.log in production
[ ] Functions are under 50 lines
```

11.2 Code Review Checklist

```
CODE REVIEW VERIFICATION:

FUNCTIONALITY:
[ ] Workflow achieves stated purpose
[ ] All requirements implemented
[ ] Edge cases handled
[ ] No obvious bugs

STANDARDS:
[ ] Naming conventions followed
[ ] Documentation complete
[ ] Code style consistent
[ ] Error handling appropriate

SECURITY:
[ ] No credentials in code/notes
[ ] Input validation present
[ ] No sensitive data logged
[ ] Webhook authentication if needed

PERFORMANCE:
[ ] No unnecessary API calls
[ ] Efficient data handling
[ ] Appropriate timeouts set
[ ] Memory usage reasonable

MAINTAINABILITY:
[ ] Logic is understandable
[ ] No overly complex nodes
[ ] Reusable components used
[ ] Easy to modify later
```

11.3 Pre-Production Checklist

```
BEFORE GOING LIVE:

ENVIRONMENT:
[ ] Production credentials configured
[ ] Environment variables set correctly
[ ] Correct API endpoints (not sandbox)
[ ] Timezone configured properly

TESTING:
[ ] Tested with production credentials
[ ] Test data cleaned up
[ ] All integrations verified
[ ] Performance acceptable

MONITORING:
[ ] Error notifications configured
[ ] Logging enabled
[ ] Alerts set up
[ ] Dashboard updated

BACKUP:
[ ] Current workflow backed up
[ ] Rollback plan documented
[ ] Previous version accessible

APPROVAL:
[ ] Technical review complete
[ ] Client approval received
[ ] Deployment window confirmed
[ ] Stakeholders notified
```

11.4 Post-Deployment Checklist

```
AFTER GO-LIVE:

IMMEDIATE (First Hour):
[ ] Workflow activated successfully
[ ] First execution monitored
[ ] No immediate errors
[ ] Notifications working

SHORT-TERM (First Day):
[ ] Multiple executions verified
[ ] Performance as expected
[ ] No unexpected errors
[ ] Client notified of success

ONGOING (First Week):
[ ] Daily monitoring in place
[ ] Any issues addressed
[ ] Performance baseline established
[ ] Documentation finalized
```

12. Code Review Standards

12.1 Review Process

```
CODE REVIEW WORKFLOW:

1. DEVELOPER
   - Completes development checklist
   - Creates review request
   - Provides context and testing notes

2. REVIEWER
   - Reviews against checklist
   - Tests workflow if needed
   - Provides written feedback

3. RESOLUTION
   - Developer addresses feedback
   - Reviewer verifies fixes
   - Approval granted

4. DEPLOYMENT
   - Follows pre-deployment checklist
   - Reviewer monitors initial deployment
```

12.2 Review Request Format

```
CODE REVIEW REQUEST TEMPLATE:

+=======================================================================+
| REVIEW REQUEST                                                        |
+=======================================================================+
|                                                                       |
| Workflow: [Workflow Name]                                             |
| Developer: [Name]                                                     |
| Date: [Date]                                                          |
|                                                                       |
| SUMMARY:                                                              |
| [Brief description of what this workflow does]                       |
|                                                                       |
| CHANGES:                                                              |
| - [Change 1]                                                          |
| - [Change 2]                                                          |
|                                                                       |
| TESTING DONE:                                                         |
| - [Test 1 and result]                                                 |
| - [Test 2 and result]                                                 |
|                                                                       |
| AREAS OF CONCERN:                                                     |
| - [Any areas that need extra attention]                               |
|                                                                       |
| HOW TO TEST:                                                          |
| 1. [Step 1]                                                           |
| 2. [Step 2]                                                           |
|                                                                       |
+=======================================================================+
```

12.3 Feedback Guidelines

Giving Feedback:

```
FEEDBACK PRINCIPLES:

1. BE SPECIFIC
   Bad:  "The error handling is wrong"
   Good: "Node 'HTTP-GetUser' should catch 404 errors
         and return empty result instead of failing"

2. EXPLAIN WHY
   Bad:  "Change this variable name"
   Good: "Rename 'x' to 'customerCount' for clarity,
         as single-letter names make maintenance harder"

3. PROVIDE SOLUTIONS
   Bad:  "This is too slow"
   Good: "Consider batching these API calls (10 per request)
         to reduce total execution time from 30s to ~5s"

4. PRIORITIZE
   - [CRITICAL] Must fix before deployment
   - [IMPORTANT] Should fix, significant impact
   - [SUGGESTION] Nice to have, minor improvement
   - [QUESTION] Need clarification
```

12.4 Review Checklist by Category

Functional Review:

```
FUNCTIONAL CORRECTNESS:

[ ] All requirements are implemented
[ ] Logic is correct for all cases
[ ] Edge cases are handled
[ ] Data flows correctly through workflow
[ ] Output matches expected format
[ ] Triggers work as expected
```

Security Review:

```
SECURITY VERIFICATION:

[ ] No hardcoded secrets
[ ] Credentials properly stored
[ ] Input validation present
[ ] Output sanitization if needed
[ ] Webhook authentication implemented
[ ] No sensitive data in logs
[ ] PII handling is appropriate
```

Performance Review:

```
PERFORMANCE CHECK:

[ ] No unnecessary API calls
[ ] Efficient loops/iterations
[ ] Appropriate batch sizes
[ ] Timeouts configured
[ ] Memory usage reasonable
[ ] Parallel processing where appropriate
```

Maintainability Review:

```
MAINTAINABILITY ASSESSMENT:

[ ] Code is readable and clear
[ ] Documentation is complete
[ ] Naming is consistent
[ ] No dead code or nodes
[ ] Error messages are helpful
[ ] Easy to modify for future needs
```

Quick Reference Card

```
+======================================================================+
|                    WORKFLOW STANDARDS QUICK REFERENCE                |
+======================================================================+

NAMING PATTERNS:
- Workflow:   [Client]-[Function]-[SubFunction]-[Version]
- Node:       [Action]-[Target]-[Qualifier]
- Credential: [Service]-[Client]-[Purpose]-[Environment]
- Variable:   camelCase (const, let)
- Constant:   UPPER_SNAKE_CASE

REQUIRED DOCUMENTATION:
- Workflow header with purpose, trigger, inputs, outputs
- Section labels for major areas
- Annotations for complex nodes
- Changelog for version history

ERROR HANDLING:
- All API calls wrapped in try-catch
- Error notifications configured
- Retry logic for transient failures
- Graceful degradation when possible

LOGGING:
- Log start and completion
- Log errors with context
- Never log sensitive data
- Use appropriate log levels

CODE QUALITY:
- Descriptive variable names
- Comments for complex logic
- Functions under 50 lines
- Validate all inputs

BEFORE DEPLOYMENT:
- Complete all checklists
- Get code review approval
- Test with production credentials
- Have rollback plan ready

+======================================================================+
```

Standards Compliance Verification

```
+======================================================================+
|                    STANDARDS COMPLIANCE AUDIT                        |
+======================================================================+

Workflow: _____________________________________________
Reviewer: _____________________________________________
Date: _________________________________________________

NAMING CONVENTIONS                              Score: ___/10
[ ] Workflow naming correct
[ ] Node naming correct
[ ] Credential naming correct
[ ] Variable naming correct
[ ] Tag naming correct

ORGANIZATION                                    Score: ___/10
[ ] Clear left-to-right flow
[ ] Proper node spacing
[ ] Section grouping
[ ] Sub-workflows used appropriately

DOCUMENTATION                                   Score: ___/10
[ ] Header documentation complete
[ ] Section labels present
[ ] Complex nodes annotated
[ ] External documentation available

ERROR HANDLING                                  Score: ___/10
[ ] All errors caught
[ ] Notifications configured
[ ] Retry logic implemented
[ ] Graceful degradation

LOGGING                                         Score: ___/10
[ ] Appropriate logging present
[ ] No sensitive data logged
[ ] Log levels correct
[ ] Searchable and useful

CODE QUALITY                                    Score: ___/10
[ ] Clean, readable code
[ ] Proper commenting
[ ] No dead code
[ ] Follows style guide

SECURITY                                        Score: ___/10
[ ] No hardcoded credentials
[ ] Input validation
[ ] Output sanitization
[ ] Proper authentication
```

```
PERFORMANCE                                           Score: ___/10
[ ] Efficient API usage
[ ] Appropriate batching
[ ] Memory management
[ ] Timeouts configured

TOTAL SCORE: ___/80

COMPLIANCE RATING:
- 72-80: Excellent (Ready for production)
- 64-71: Good (Minor improvements needed)
- 56-63: Acceptable (Improvements recommended)
- Below 56: Needs Work (Must improve before deployment)

NOTES:
__________________________________________________________
__________________________________________________________
__________________________________________________________

+======================================================================+
```

Related Guides:
- `02-security-implementation.md` - Security best practices
- `04-testing-qa-framework.md` - Testing procedures
- `05-handover-delivery.md` - Delivery standards
- `06-maintenance-retainer.md` - Ongoing maintenance
