

Complete n8n Automation Guide

From Beginner to Advanced: A Comprehensive Educational Handbook

Version 1.0 | February 2026

Author: n8n Automation Specialist

Table of Contents

Part I: Foundations

1. Introduction to n8n
2. Core Automation Concepts
3. Getting Started with n8n

Part II: Building Blocks

4. Nodes Deep Dive
5. Data Handling & Transformation
6. Integrations & APIs

Part III: Advanced Techniques

7. Advanced Workflow Logic
8. Error Handling & Reliability
9. Scaling & Performance

Part IV: Security & Production

10. Security & Governance
11. Deployment & Production
12. Extending n8n

Part V: Real-World Applications

13. Business Use Cases
14. n8n + AI Integration
15. Future of Automation

Part VI: Resources

16. Appendices
-

Part I: Foundations

Chapter 1: Introduction to n8n

What is n8n?

n8n (pronounced "n-eight-n") is a fair-code workflow automation platform that helps you connect any app with an API to any other app and manipulate data with little or no code[1]. Think of it as the "IFTTT on steroids" or a self-hostable alternative to Zapier that gives you complete control over your automation infrastructure.

Core Philosophy:

- **Fair-code licensed:** Source-available with a sustainable business model
- **Privacy-first:** Self-host for complete data control
- **Extendable:** Build custom nodes or use 400+ pre-built integrations
- **Developer-friendly:** Full JavaScript/Python code access when needed

Why n8n Exists

Traditional automation platforms face critical limitations:

The Vendor Lock-in Problem: Cloud-only platforms control your data, workflows, and pricing. One policy change can break your entire automation strategy.

The Code vs. No-Code Dilemma: Pure no-code tools hit walls with complex logic. Pure code solutions require heavy development time.

The Cost Scaling Issue: Task-based pricing becomes prohibitively expensive at scale. A single workflow processing 100,000 items monthly can cost thousands on traditional platforms.

n8n solves these by providing:

- Self-hosting option (own your infrastructure)
- Hybrid approach (visual + code when needed)
- Execution-based pricing (Cloud) or free unlimited executions (self-hosted)

n8n vs. Competitors

![Platform Comparison][chart:47]

n8n vs. Zapier

Aspect	n8n	Zapier
Hosting	Self-hosted or Cloud	Cloud only
Integrations	400+ nodes	8,000+ apps
Pricing Model	Execution-based (Cloud) or Free (self-hosted)	Task-based
Customization	Full code access, custom nodes	Limited code steps
Target Users	Technical teams, developers	Non-technical users
AI Capabilities	Native LangChain, full LLM control	AI integrations via apps
Complex Workflows	Native support (loops, branches, sub-workflows)	Limited complexity
Data Control	Full control (self-hosted)	Vendor-controlled

Best for n8n: Technical teams, high-volume processing, complex workflows, data sovereignty requirements, AI/LLM integration[2][3].

Best for Zapier: Quick setup, non-technical users, need many pre-built integrations, simple linear workflows.

n8n vs. Make (Integromat)

Aspect	n8n	Make
Hosting	Self-hosted or Cloud	Cloud only
Visual Builder	Node-based	Scenario-based (visual flow)
Pricing	More affordable at scale	Operation-based
Complexity	Full control with code	Visual complexity
Customization	Unlimited (self-hosted)	Limited to platform
Learning Curve	Steeper (more power)	Moderate

Best for n8n: Self-hosting needs, complex business logic, AI workflows, developer teams[3].

Best for Make: Visual workflow design preference, cloud-only acceptable, good middle ground.

Automation Concepts Explained Simply

What is Workflow Automation?

Automation replaces manual, repetitive tasks with automated processes. Instead of:

1. Manually checking email for new orders
2. Copying data to a spreadsheet
3. Sending confirmation messages
4. Updating inventory

You create a workflow that does all four steps automatically when an email arrives.

The Trigger-Action Model

Every automation follows this pattern:

TRIGGER (When X happens) → ACTIONS (Do Y and Z)

Example: When a new customer signs up (trigger), send welcome email (action 1) and add to CRM (action 2).

Event-Driven Architecture

Modern automation responds to events in real-time:

- Webhook receives data → Process immediately
- Database row changes → Update related systems
- Schedule hits → Execute batch operation

Data Flow Fundamentals

Data moves through workflows like water through pipes:

Input Data → Transform → Filter → Enrich → Output

Each node is a processing station that receives data, does something with it, and passes it to the next node.

Chapter 2: Core Automation Concepts

Understanding Nodes

What is a Node?

A node is a single step in your workflow. It represents one action, decision, or data transformation. Think of nodes as Lego blocks—each piece has a specific function, and you combine them to build complex structures.

Node Categories:

1. **Trigger Nodes** (Green lightning bolt icon)
 - Start workflow execution
 - Examples: Webhook, Schedule, Email Trigger
 - Always the first node in a workflow
2. **Action Nodes** (Blue icon)
 - Perform operations
 - Examples: HTTP Request, Database Query, Send Email
 - The "doing" part of your workflow
3. **Core Nodes** (Purple icon)
 - Built-in functionality
 - Examples: IF, Switch, Code, Set, Merge
 - Logic and data manipulation
4. **AI Nodes** (Orange brain icon)
 - AI and LLM operations
 - Examples: OpenAI, LangChain Agent, Vector Store
 - Introduced in n8n 1.0+

Triggers vs. Actions

Trigger Nodes (How Workflows Start)

Trigger Type	How it Works	Use Cases
Webhook	Receives HTTP requests	API integrations, real-time events
Schedule	Time-based execution	Daily reports, periodic cleanup
Poll Trigger	Checks for new data periodically	Email monitoring, RSS feeds
Event Trigger	Listens to external events	Database changes, file uploads
Manual Trigger	You click "Execute"	Testing, on-demand execution

Action Nodes (What Workflows Do)

- **Data Operations:** Transform, filter, aggregate data
- **Integrations:** Connect to external services (Gmail, Slack, databases)
- **Logic:** IF/THEN decisions, routing, loops
- **Output:** Save to database, send notifications, create files

Data Flow Fundamentals

![[Workflow Execution Flow]][chart:48]

Items and JSON Structure

n8n processes data as **items**—individual units of information. Each item contains:

```
{
  "json": {
    "name": "John Doe",
    "email": "john@example.com",
    "age": 30
  },
  "binary": {} // Optional: files, images
}
```

Key Concepts:

- **Item:** One record/object being processed
- **JSON:** The actual data payload
- **Binary:** File attachments (images, PDFs, etc.)
- **Batch Processing:** Multiple items flowing through the workflow

Example Data Flow:

```
Trigger Node
↓ (Outputs 3 items)
[Item 1: User A, Item 2: User B, Item 3: User C]
↓
Filter Node (Keep only premium users)
↓ (Outputs 2 items)
[Item 1: User A, Item 2: User C]
↓
Email Node (Send to each)
↓
2 emails sent
```

Executions

What is an Execution?

One complete run of your workflow from trigger to completion. Each execution:

- Has a unique ID
- Processes one or more items
- Stores execution history (configurable retention)
- Shows success/failure status

Execution Modes:

1. **Main Process** (Default)
 - Executions run in the main n8n process
 - Good for: Low to moderate volume

- Limitation: Can't scale horizontally
- 2. Queue Mode** (Production)
- Executions queued in Redis/database
 - Worker processes pick up jobs
 - Good for: High volume, reliability
 - Scales horizontally

Execution Context:

Every execution has access to metadata:

`$execution.id` // Unique execution ID
`$execution.mode` // "manual" or "production"
`$execution.resumeUrl` // For wait nodes
`$workflow.id` // Workflow identifier
`$workflow.name` // Workflow name

JSON Basics for n8n

Why JSON Matters

n8n uses JSON (JavaScript Object Notation) for all data. Understanding JSON is essential for effective automation.

JSON Structure:

```
{
  "string": "text value",
  "number": 42,
  "boolean": true,
  "array": [1, 2, 3],
  "object": {
    "nested": "value"
  },
  "null": null
}
```

Accessing JSON Data in n8n:

```
// Get a field
{{ $json.fieldName }}
```

```
// Nested access
{{ $json.useremail }}
```

```
// Array access
{{ $json.items[0] }}
```

```
// Previous node data
{{ $('Node Name').item.json.field }}
```

Common JSON Operations:

```
// Check if field exists
{{ $json.field ?? 'default' }}

// Convert to uppercase
{{ $json.name.toUpperCase() }}

// Get array length
{{ $json.items.length }}

// Filter array
{{ $json.users.filter(u => u.active) }}
```

Chapter 3: Getting Started with n8n

Installation Options

n8n offers multiple installation methods based on your needs:

1. n8n Cloud (Easiest - Recommended for Beginners)

Fully managed hosting by n8n:

- Sign up at <https://n8n.io>
- No infrastructure management
- Automatic updates
- SSL certificates included
- Pricing: Based on executions (~\$20-300/month)

Pros: Zero setup, automatic backups, professional support

Cons: Vendor dependency, monthly costs

2. Docker (Recommended for Self-Hosting)

Single command deployment:

```
docker run -it --rm
--name n8n
-p 5678:5678
-v ~/.n8n:/home/node/.n8n
n8nio/n8n
```

Docker Compose (Production-Ready):

```
version: '3.8'
```

```
services:
```

```
n8n:
```

```
image: n8nio/n8n:latest
```

```
container_name: n8n
```

```
restart: always
```

```
ports:
```

```
- "5678:5678"
```

```
environment:
```

- N8N_BASIC_AUTH_ACTIVE=true
- N8N_BASIC_AUTH_USER=admin
- N8N_BASIC_AUTH_PASSWORD=secure_password
- N8N_HOST=[yourdomain.com](#)
- WEBHOOK_URL=https://yourdomain.com/
- GENERIC_TIMEZONE=America/New_York
- N8N_ENCRYPTION_KEY=your_encryption_key_here
- EXECUTIONS_DATA_PRUNE=true
- EXECUTIONS_DATA_MAX_AGE=168 # 7 days

volumes:

- ~/.n8n:/home/node/.n8n

Pros: Isolated environment, easy updates, portable

Cons: Requires Docker knowledge

3. npm (For Developers)

Install globally

npm install n8n -g

Run n8n

n8n start

Pros: Quick setup, no containers needed

Cons: Requires Node.js, less isolated

4. Kubernetes (Enterprise)

Scalable, production-grade deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: n8n
spec:
  replicas: 3
  selector:
    matchLabels:
      app: n8n
  template:
    metadata:
      labels:
        app: n8n
    spec:
      containers:
        - name: n8n
          image: n8nio/n8n:latest
      ports:
        - containerPort: 5678
```

env:

- name: DB_TYPE

value: "postgresdb"

- name: N8N_RUNNERS_ENABLED

value: "true"

Pros: Auto-scaling, high availability, enterprise features

Cons: Complex setup, requires Kubernetes expertise

Environment Configuration

Essential Environment Variables:

Security

N8N_ENCRYPTION_KEY=random_32_char_string # CRITICAL for credential security

N8N_BASIC_AUTH_ACTIVE=true

N8N_BASIC_AUTH_USER=admin

N8N_BASIC_AUTH_PASSWORD=secure_password

Hosting

N8N_HOST=[yourdomain.com](#)

N8N_PORT=5678

N8N_PROTOCOL=https

WEBHOOK_URL=https://yourdomain.com/

Database (PostgreSQL recommended for production)

DB_TYPE=postgresdb

DB_POSTGRESDB_HOST=postgres

DB_POSTGRESDB_PORT=5432

DB_POSTGRESDB_DATABASE=n8n

DB_POSTGRESDB_USER=n8n_user

DB_POSTGRESDB_PASSWORD=db_password

Execution Settings

EXECUTIONS_PROCESS=main # or "queue" for production

EXECUTIONS_DATA_PRUNE=true

EXECUTIONS_DATA_MAX_AGE=168 # hours

Queue Mode (Redis required)

QUEUE_BULL_REDIS_HOST=redis

QUEUE_BULL_REDIS_PORT=6379

Timezone

GENERIC_TIMEZONE=America/New_York

Logging

N8N_LOG_LEVEL=info

N8N_LOG_OUTPUT=console

UI Overview

Main Interface Components:

1. **Workflow Canvas** (Center)
 - Drag-and-drop nodes
 - Visual connection lines
 - Zoom and pan controls
2. **Left Sidebar**
 - Workflows list
 - Credentials management
 - Executions history
 - Community nodes
3. **Top Bar**
 - Save workflow
 - Execute workflow (test)
 - Active/Inactive toggle (production mode)
 - Settings and help
4. **Right Panel** (When node selected)
 - Node parameters
 - Test execution
 - Expression editor
 - Output data viewer
5. **Node Panel** (Opens when clicking +)
 - Search nodes
 - Browse by category
 - Triggers vs Actions tabs

Navigation Shortcuts:

- **Ctrl/Cmd + Enter:** Execute workflow
- **Ctrl/Cmd + S:** Save workflow
- **Delete:** Remove selected node
- **Ctrl/Cmd + C/V:** Copy/paste nodes
- **Ctrl/Cmd + Z:** Undo

Creating Your First Workflow

Tutorial: "Daily Weather Email Reminder"

Goal: Send yourself weather forecast every morning at 8 AM.

Step 1: Add Schedule Trigger

1. Click the "+" button
2. Search for "Schedule Trigger"
3. Configure:
 - Mode: "Every Day"
 - Hour: 8
 - Minute: 0

Step 2: Get Weather Data

1. Add HTTP Request node
2. Configure:
 - Method: GET
 - URL: <https://wttr.in/NewYork?format=j1>
 - (Free weather API, no key needed)

Step 3: Extract Weather Info

1. Add Code node
2. Write simple extraction logic:

```
const weather = items[0].json;  
const current = weather.current_condition[0];  
const forecast = weatherweather[0];
```

```
return [{  
  json: {  
    temperature: current.temp_F + "°F",  
    condition: current.weatherDesc[0].value,  
    high: forecast.maxtempF + "°F",  
    low: forecast.mintempF + "°F",  
    city: "New York"  
  }  
}];
```

Step 4: Send Email

1. Add Gmail node (or any email service)
2. Add Gmail credentials (OAuth2)
3. Configure:
 - Operation: Send Email
 - To: your@email.com
 - Subject: Daily Weather - {{ \$json.city }}
 - Message:

Good morning!

Today's weather in {{ \$json.city }}:
Current: {{ \$json.temperature }} - {{ \$json.condition }}
High: {{ \$json.high }}
Low: {{ \$json.low }}

Have a great day!

Step 5: Test and Activate

1. Click "Execute workflow" (test)
2. Check each node's output
3. Verify email received
4. Click "Active" toggle (top right)

Congratulations! Your first automation is running.

Part II: Building Blocks

Chapter 4: Nodes Deep Dive

n8n 2.7+ includes 400+ nodes[4]. This chapter covers the 17 most essential nodes that appear in 90% of workflows.

Trigger Nodes

1. Schedule Trigger

Executes workflows on a time-based schedule.

Use Cases:

- Daily reports at 9 AM
- Weekly data cleanup
- Monthly invoice generation
- Hourly health checks

Configuration Options:

Mode: Intervals, Days of Week, Custom Cron

Timezone: Automatic detection or manual

Examples:

Every hour: 0 * * * *

Business days 9 AM: 0 9 * * 1-5

First of month: 0 0 1 * *

Every 15 minutes: */15 * * * *

Pro Tip: Use timezone awareness. "9 AM EST" is different from "9 AM UTC."

2. Webhook

Receives HTTP requests to trigger workflows.

Use Cases:

- API integrations
- Third-party service callbacks
- Form submissions
- Real-time notifications

Configuration:

HTTP Method: GET, POST, PUT, DELETE, PATCH

Path: /webhook/unique-identifier

Authentication: None, Basic, Header

Response: Return data or custom response

Example Webhook URL:

<https://your-n8n.com/webhook/order-received>

Code to call webhook:

```
fetch('https://your-n8n.com/webhook/order-received', {  
  method: 'POST',  
  headers: { 'Content-Type': 'application/json' },  
  body: JSON.stringify({  
    orderId: '12345',  
    customer: 'John Doe',  
    amount: 99.99  
  })  
});
```

Security Best Practice: Always use header authentication for production webhooks:

Header Name: X-API-Key

Header Value: your_secret_key_here

3. Email Trigger (IMAP)

Monitors email inbox for new messages.

Use Cases:

- Process order confirmations
- Extract data from automated reports
- Customer support automation
- Invoice processing

Configuration:

IMAP Host: imap.gmail.com

Port: 993

Email: your@email.com

Folder: INBOX

Poll Interval: Every 1 minute

Example Filter: Only process emails with subject containing "Order Confirmation"

Core Logic Nodes

4. IF Node

Routes data based on conditions. Most commonly used logic node.

Structure:

Input → Condition Check → True Output OR False Output

Configuration:

Condition Type:

- Boolean (true/false)
- Number (equals, greater than, less than)
- String (equals, contains, regex)
- Date/Time comparison
- Exists (field has value)

Example: Route based on order value

Condition: `{{ $json.amount }} > 100`

True → VIP Customer Workflow

False → Standard Customer Workflow

Multiple Conditions: Use "Add Condition" to create AND/OR logic.

5. Switch Node

Routes to multiple paths (like IF with 3+ options).

Use Cases:

- Order status routing (pending/shipped/delivered/cancelled)
- Priority levels (low/medium/high/urgent)
- Department routing (sales/support/billing)
- Language routing (en/es/fr/de)

Configuration:

Mode: Rules or Expression

Output: 0, 1, 2, 3... (multiple outputs)

Fallback: "Default" output if no rules match

Example: Route by Customer Type

Rule 1: `{{ $json.type }} equals "enterprise"` → Output 0

Rule 2: `{{ $json.type }} equals "business"` → Output 1

Rule 3: `{{ $json.type }} equals "individual"` → Output 2

Fallback → Output 3 (unknown type)

6. Code Node

Run custom JavaScript or Python code.

When to Use:

- Complex data transformations
- Custom business logic
- API response parsing
- Mathematical calculations
- No suitable built-in node

JavaScript Example:

```
// Access input items
for (const item of items) {
  // Transform data
  item.json.fullName = `${item.json.firstName} ${item.json.lastName}`;
  item.json.discountPrice = item.json.price * 0.9;

  // Add new fields
  item.json.processedAt = new Date().toISOString();
}

// Return modified items
return items;
```

Python Example (n8n 1.0+):

```
import json
from datetime import datetime
```

Access items

```
for item in items:
  # Transform data
  item['json']['fullName'] = f"{item['json']['firstName']} {item['json']['lastName']}"
  item['json']['processedAt'] = datetime.now().isoformat()
```

Return items

```
return items
```

Best Practices:

- Keep code concise (use sub-workflows for complex logic)
- Handle errors with try/catch
- Log important steps for debugging
- Return items array, not raw data

7. Set Node (Edit Fields)

Transform data structure without code.

Operations:

- Set field values
- Remove fields
- Rename fields
- Convert types

Use Cases:

- Clean up API responses
- Standardize field names
- Remove sensitive data
- Prepare data for next node

Example:

Keep Only:

- customerName (from [json.customername](#))
- orderTotal (from [json.total](#))
- orderDate (from [json.created_at](#))

Remove:

- internal_id
- raw_data

8. Merge Node

Combines data from multiple paths.

Modes:

1. **Append:** Stack items from both inputs
2. **Merge By Index:** Combine items at same position
3. **Merge By Key:** Join like SQL (match on field)
4. **Multiplex:** Create all combinations

Example: Enrich Customer Data

Input 1: Customer list from CRM

Input 2: Order history from database

Merge By Key: customer_id

Result: Customers with order counts attached

9. Split Out

Converts single item with array into multiple items.

Use Case:

Input (1 item):

```
{
  "company": "Acme Corp",
  "employees": [
    {"name": "Alice", "email": "alice@acme.com"},
    {"name": "Bob", "email": "bob@acme.com"}
  ]
}
```

```
]
}
```

Output (2 items):

Item 1: {"name": "Alice", "email": "alice@acme.com"}

Item 2: {"name": "Bob", "email": "bob@acme.com"}

10. Aggregate

Opposite of Split Out. Combines multiple items into one.

Use Case:

Input (3 items):

Item 1: {"sale": 100}

Item 2: {"sale": 200}

Item 3: {"sale": 150}

Output (1 item):

```
{
  "sales": [
    {"sale": 100},
    {"sale": 200},
    {"sale": 150}
  ],
  "total": 450
}
```

Integration Nodes

11. HTTP Request

The Swiss Army knife of n8n. Connects to any API.

Use Cases:

- REST API calls
- GraphQL queries
- Webhook sending
- File downloads
- Custom integrations

Configuration:

Method: GET, POST, PUT, DELETE, PATCH

URL: <https://api.example.com/endpoint>

Authentication: None, Basic Auth, OAuth2, Header, Query

Headers: Custom HTTP headers

Body: JSON, Form Data, Raw

Response Format: Autodetect, JSON, String, Binary

Example: Call OpenAI API

Method: POST
URL: <https://api.openai.com/v1/chat/completions>
Authentication: Header Auth
Header Name: Authorization
Value: Bearer YOUR_API_KEY
Headers:
Content-Type: application/json
Body (JSON):
{
 "model": "gpt-4",
 "messages": [
 {"role": "user", "content": "{{ \$json.userQuestion }}" }
]
}

Error Handling:

Retry on Fail: 3 attempts
Wait Between Tries: 1000ms (1 second)
Continue On Fail: true (optional)

12. Loop Over Items

Processes items one at a time with rate limiting.

Use Cases:

- API rate limit compliance (max 10 requests/second)
- Sequential processing
- Batching operations
- Iterative API calls

Configuration:

Batch Size: 1 (one at a time)
Pause Between Batches: 100ms
Maximum Iterations: 1000

Example: Process 1000 users, 10 per second

Loop Over Items:
Batch Size: 1
Pause: 100ms (10 items per second)

Inside Loop:
→ Update User in CRM
→ Send Notification

13. Subworkflow

Call another workflow as a reusable component.

Use Cases:

- Error notification workflow (reused across all workflows)
- Data validation logic
- Common transformations
- Modular architecture

Example: Reusable Email Error Handler

Main Workflow:

- Process Data
- IF Error Occurs
- Call Subworkflow: "Send Error Email"

Subworkflow "Send Error Email":

Input: error details

- Format Error Message
- Send to Admin Email
- Log to Database

14. Wait Node

Pauses execution for a duration or until a condition.

Use Cases:

- Wait for API processing
- Delay between retries
- Scheduled follow-ups
- Rate limiting

Modes:

1. **Duration:** Wait X seconds/minutes/hours
2. **Until:** Wait until specific date/time
3. **Resume Webhook:** Wait for external trigger

Example: Follow-up Email Sequence

Send Welcome Email

↓

Wait (2 days)

↓

Send Tutorial Email

↓

Wait (3 days)

↓

Send Feedback Request

15. Error Trigger

Catches errors from any workflow.

Use Cases:

- Global error monitoring

- Error notifications
- Failed execution retry
- Error logging

Configuration:

Trigger On: Workflow Error

Capture: Error message, stack trace, workflow name

Example: Error Notification System

Error Trigger

↓

Format Error Email:

Subject: "Workflow Failed: {{ \$json.workflow.name }}"

Body: "Error: {{ \$json.errorMessage }}"

↓

Send to Admin

↓

Log to Error Database

16. Webhook Response

Send custom response from webhook trigger.

Use Cases:

- API endpoints
- Sync responses
- Status confirmations
- Error messages to caller

Example: Order API Endpoint

Webhook (POST /create-order)

↓

Validate Order Data

↓

IF Valid:

→ Create Order in Database

→ Webhook Response: {"status": "success", "orderId": "123"}

Else:

→ Webhook Response: {"status": "error", "message": "Invalid data"}

17. Sticky Note

Documentation and comments in workflow.

Use Cases:

- Explain complex logic
- TODO notes
- Workflow documentation
- Team collaboration notes

Best Practice: Add sticky notes for:

- Workflow purpose (top of canvas)
- Complex transformations
- External dependencies
- Known issues/limitations

Chapter 5: Data Handling & Transformation

Expressions Explained from Scratch

What are Expressions?

Expressions are JavaScript snippets wrapped in `{{ }}` that allow dynamic data access and transformation[5].

Basic Syntax:

```
{{ expression here }}
```

Fundamental Expressions:

```
// Access current item data  
{{ $json.fieldName }}
```

```
// Access nested data  
{{ $json.customeremail }}
```

```
// Access array elements  
{{ $json.items[0].price }}
```

```
// Access previous node data  
{{ $('HTTP Request').item.json.userId }}
```

Built-in Variables:

Variable	Description	Example
<code>\$json</code>	Current item's JSON data	<code>{{ \$json.name }}</code>
<code>\$binary</code>	Current item's binary data	<code>{{ \$binary.data }}</code>
<code>\$itemIndex</code>	Current item's index (0-based)	<code>{{ \$itemIndex }}</code>
<code>\$now</code>	Current timestamp	<code>{{ \$now.toFormat('yyyy-MM-dd') }}</code>
<code>\$today</code>	Today at midnight	<code>{{ \$today }}</code>
<code>\$execution.id</code>	Execution ID	<code>{{ \$execution.id }}</code>
<code>\$workflow.name</code>	Workflow name	<code>{{ \$workflow.name }}</code>

Expression Modes:

Every parameter field has two modes:

1. **Fixed:** Static value (no expressions)
2. **Expression:** Dynamic value with `{{ }}`

Toggle between modes by clicking the "fx" icon.

JavaScript Basics for n8n

String Operations:

```
// Concatenation
{{ $json.firstName + ' ' + $json.lastName }}
```

```
// Template literals
{{ Hello, ${$json.name}! }}
```

```
// String methods
{{ $json.email.toLowerCase() }}
{{ $json.name.toUpperCase() }}
{{ $json.text.trim() }}
{{ $json.title.replace('old', 'new') }}
```

```
// Check if contains
{{ $json.text.includes('keyword') }}
```

```
// Extract substring
{{ $json.email.substring(0, 5) }}
```

Number Operations:

```
// Arithmetic
{{ $json.price * 1.2 }} // Add 20% tax
{{ $json.total - $json.discount }}
{{ $json.quantity * json.value) }}
```

```
// Comparison
{{ $json.age >= 18 }}
{{ $json.price < 100 }}
{{ $json.stock === 0 }}
```

Conditional Logic:

```
// Ternary operator
{{ $json.age >= 18 ? 'Adult' : 'Minor' }}

// Null coalescing
{{ $json.nickname ?? $json.firstName }}

// Logical OR (fallback)
{{ $json.phone || 'No phone provided' }}

// Logical AND
{{ $json.premium && $json.verified }}

// Complex conditionals
{{ $json.score >= 90 ? 'A' : $json.score >= 80 ? 'B' : 'C' }}
```

Date/Time with Luxon:

n8n includes Luxon for date operations.

```
// Current time
{{ $now.toISO() }}
{{ $now.toFormat('yyyy-MM-dd HH:mm:ss') }}

// Date arithmetic
{{ $now.plus({ days: 7 }) }}
{{ $now.minus({ hours: 2 }) }}

// Parse date string
{{ DateTime.fromISO($json.dateString) }}

// Format examples
{{ $now.toFormat('MM/dd/yyyy') }} // 02/03/2026
{{ $now.toFormat('MMMM d, yyyy') }} // February 3, 2026
{{ $now.toFormat('HH:mm') }} // 14:30

// Date comparison
{{ now.diff(DateTime.fromISO(json.created), 'days').days }}
```

Data Mapping and Restructuring

Transform Object Structure:

Input:

```
{
  "user": {
    "profile": {
      "name": "John Doe",
      "contact": {
        "email": "john@example.com"
      }
    }
  }
}
```

Output using Set Node:

```
{
  "userName": "John Doe",
  "userEmail": "john@example.com"
}
```

Array Transformation:

```
// Map array
{{ $json.orders.map(order => ordertotal) }}
// Result: [100, 200, 150]

// Filter array
{{ $json.products.filter(p => p.stock > 0) }}

// Reduce array (sum)
{{ $json.items.reduce((sum, item) => sum + item.price, 0) }}

// Find in array
{{ $json.users.find(u => u.id === '123') }}

// Check if any match
{{ $json.items.some(item => item.price > 100) }}

// Check if all match
{{ $json.items.every(item => item.inStock) }}
```

Common Transformation Patterns:

1. Extract Email from Text:

```
{{ $json.text.extractEmail() }}
```

2. Parse JSON String:

```
{{ JSON.parse($json.jsonString) }}
```

3. Stringify Object:

```
{{ JSON.stringify($json.dataObject) }}
```

4. Convert CSV to Array:

```
{{ $json.csvString.split(',').map(v => v.trim()) }}
```

5. Generate Random String:

```
{{ Math.random().toString(36).substring(7) }}
```

Working with Arrays, Objects, and Pagination

Array Manipulation:

```
// Sort array
```

```
{{ $json.items.sort((a, b) => a.price - b.price) }}
```

```
// Reverse array
```

```
{{ $json.items.reverse() }}
```

```
// Get unique values
```

```
{{ [...new Set($json.tags)] }}
```

```
// Slice array (first 10 items)
```

```
{{ $json.results.slice(0, 10) }}
```

```
// Concat arrays
```

```
{{ json.array1.concat(json.array2) }}
```

```
// Flatten nested arrays
```

```
{{ $json.nested.flat() }}
```

Object Operations:

```
// Get object keys
```

```
{{ Object.keys($json.userData) }}
```

```
// Get object values
```

```
{{ Object.values($json.userData) }}
```

```
// Check if property exists
```

```
{{ 'email' in $json }}
```

```
// Merge objects
```

```
{{ Object.assign({}, $json.defaults, $json.userSettings) }}
```

```
// Create object from arrays
```

```
{{ Object.fromEntries($json.pairs) }}
```

Pagination Patterns:

API Pagination (Offset-based):

```
// Loop setup
```

```
currentPage = 1
```

```
itemsPerPage = 100
```

```
// HTTP Request URL
{{ https://api.example.com/items?page={{currentPage}}&limit=100 }}

// Check if more pages
{{ $json.totalItems > (currentPage * itemsPerPage) }}
```

Cursor-based Pagination:

```
// First request
URL: https://api.example.com/items?limit=100

// Subsequent requests
URL: https://api.example.com/items?limit=100&cursor={{ \$json.nextCursor }}

// Loop until no nextCursor
Continue While: {{ $json.nextCursor !== null }}
```

Handle Large Datasets:

```
HTTP Request (Get Page 1)
↓
Split Out (array of items)
↓
Loop Over Items (batch size: 10)
↓
Process Each Item
↓
Check for Next Page
↓
IF Next Page Exists → HTTP Request (Next Page)
```

Expression Best Practices

1. Use Null Safety:

```
// Bad: Will error if email is undefined
{{ $json.user.email.toLowerCase() }}

// Good: Safe navigation
{{ $json.user?.email?.toLowerCase() ?? 'no-email@example.com' }}
```

2. Avoid Complex Logic in Expressions:

```
// Bad: Hard to read
{{ $json.users.filter(u => u.age > 18 && u.country === 'US' && u.verified).map(u => ({name:
u.name, email: u.email))).slice(0, 10) }}

// Good: Use Code node for complex operations
```

3. Debug with Console:

```
Code Node debugging:

console.log('Item data:', items[0].json);
console.log('Field value:', items[0].json.fieldName);
```

return items;

4. Format for Readability:

// Use Set node or Code node for multi-step transformations

// Keep expressions short and focused

Chapter 6: Integrations & APIs

REST API Fundamentals

What is REST API?

REST (Representational State Transfer) is an architectural style for APIs that uses HTTP methods to interact with resources.

HTTP Methods:

Method	Purpose	Example
GET	Retrieve data	Get list of customers
POST	Create new resource	Create new order
PUT	Update entire resource	Replace customer record
PATCH	Update partial resource	Update email address
DELETE	Remove resource	Delete product

API Request Structure:

METHOD /endpoint?queryParam=value HTTP/1.1

Host: api.example.com

Header-Name: Header-Value

Content-Type: application/json

{request body for POST/PUT/PATCH}

Response Structure:

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "status": "success",
  "data": {...}
}
```

Status Codes:

- **2xx Success:** 200 (OK), 201 (Created), 204 (No Content)
- **4xx Client Error:** 400 (Bad Request), 401 (Unauthorized), 404 (Not Found)

- **5xx Server Error:** 500 (Internal Error), 503 (Service Unavailable)

Authentication Methods

1. No Authentication

Public APIs with no security.

URL: <https://api.publicdata.com/weather>

2. API Key (Header)

Most common for simple APIs.

HTTP Request Configuration:

Authentication: Header Auth

Header Name: X-API-Key

Header Value: your_api_key_here

3. API Key (Query Parameter)

URL: https://api.example.com/data?api_key=your_key_here

4. Basic Authentication

Username and password (Base64 encoded).

Authentication: Basic Auth

Username: your_username

Password: your_password

n8n automatically encodes to Authorization: Basic dXNlcjpwYXNz

5. Bearer Token

Token-based authentication.

Authentication: Header Auth

Header Name: Authorization

Header Value: Bearer your_token_here

6. OAuth2

Complex but secure. Used by Google, Microsoft, etc.

OAuth2 Flow:

1. User authorizes app
2. Receive authorization code
3. Exchange code for access token
4. Use access token for API calls
5. Refresh token when expired

n8n OAuth2 Setup:

Credentials → Add Credential → OAuth2 API
Client ID: your_client_id
Client Secret: your_client_secret
Authorization URL: <https://provider.com/oauth/authorize>
Token URL: <https://provider.com/oauth/token>
Scopes: read write

7. JWT (JSON Web Tokens)

Self-contained tokens with encoded data.

```
// Generate JWT (Code node)
const jwt = require('jsonwebtoken');

const token = jwt.sign(
  { userId: 123, role: 'admin' },
  'your_secret_key',
  { expiresIn: '1h' }
);

return [{ json: { token } }];
```

Connecting SaaS Tools

Common Integration Patterns:

1. CRM Integration (Salesforce, HubSpot)

Webhook (New Lead)
↓
HTTP Request (Create Contact in CRM)
Method: POST
URL: <https://api.crm.com/contacts>
Body: {
 "email": "{{ \$json.email }}",
 "name": "{{ \$json.name }}",
 "source": "website"
}
↓
Send Confirmation Email

2. Database Operations

PostgreSQL Node:
Operation: Execute Query
Query: SELECT * FROM users WHERE created_at > NOW() - INTERVAL '24 hours'
↓
Process New Users
↓
Send Welcome Email to Each

3. File Storage (Google Drive, S3)

Google Drive Node:
Operation: Upload File
File: {{ \$binary.data }}
Parent Folder: "Reports/{{ \$now.toFormat('yyyy-MM') }}"
Name: "Report_{{ \$now.toFormat('yyyy-MM-dd') }}.pdf"

4. Communication (Slack, Email)

Slack Node:
Operation: Send Message
Channel: #notifications
Text: "New order received: \${{ \$json.total }}"
Attachments: [Order details JSON]

Error Handling and Retries

Built-in Retry Configuration:

HTTP Request Node Settings:
Retry on Fail: true
Max Tries: 3
Wait Between Tries: 1000ms (1 second)

Retry Strategies:

1. Exponential Backoff

Try 1: Immediate
Try 2: Wait 1 second
Try 3: Wait 2 seconds
Try 4: Wait 4 seconds
Try 5: Wait 8 seconds

Implementation:

```
// Code node retry logic
const maxRetries = 5;
let attempt = 0;
let success = false;

while (attempt < maxRetries && !success) {
  try {
    // API call here
    success = true;
  } catch (error) {
    attempt++;
    const waitTime = Math.pow(2, attempt) * 1000;
    await new Promise(resolve => setTimeout(resolve, waitTime));
  }
}
```

2. Circuit Breaker Pattern

Track failures over time

If failures > threshold:

- Open circuit (stop trying)
- Wait cooldown period
- Try again (half-open state)
- If success: Close circuit
- If fail: Open circuit again

3. Error Workflow Pattern

Main Workflow:

Try Operation

↓

IF Error:

- Trigger Error Workflow

Error Workflow:

Error Trigger

↓

Log Error to Database

↓

Send Notification to Admin

↓

(Optional) Retry Original Operation

Error Handling Best Practices:

1. **Set Continue on Fail** for non-critical nodes
2. **Use Error Trigger** for global error monitoring
3. **Implement retry logic** for transient failures
4. **Log errors** for debugging
5. **Set timeouts** to prevent hanging workflows
6. **Validate data** before API calls

Example: Robust API Call

HTTP Request Node:

URL: <https://api.example.com/data>

Authentication: Bearer token

Timeout: 30000ms (30 seconds)

Retry on Fail: true

Max Tries: 3

Wait Between: 2000ms

Continue on Fail: false (fail workflow if all retries fail)

Response Error Handling:

IF Status >= 400:

- Trigger Error Workflow
 - Log to Monitoring
-

Part III: Advanced Techniques

Chapter 7: Advanced Workflow Logic

Conditional Logic

Complex IF Conditions:

```
// Multiple conditions (AND)
{{ $json.age >= 18 && $json.verified === true && $json.country === 'US' }}

// Multiple conditions (OR)
{{ $json.role === 'admin' || $json.role === 'superadmin' }}

// Nested conditions
{{ json.isPremium?(json.credits > 100 ? 'VIP' : 'Premium') : 'Standard' }}

// Regular expression matching
{{ / }}
```

Switch Node for Multiple Paths:

Example: Customer Segmentation

Switch Node:

Rule 1: {{ \$json.totalSpent > 10000 }} → VIP Path
Rule 2: {{ \$json.totalSpent > 1000 }} → Premium Path
Rule 3: {{ \$json.totalSpent > 100 }} → Regular Path
Fallback → New Customer Path

Looping and Batching

Loop Over Items Configuration:

Batch Size: Number of items per iteration
Pause Between Batches: Milliseconds to wait

Use Case 1: Rate-Limited API

API allows 10 requests/second:

Loop Over Items:

Batch Size: 1

Pause: 100ms (10 items per second)

Inside Loop:

- HTTP Request to API
- Process Response

Use Case 2: Batch Processing

Process 1000 records in batches of 50:

Loop Over Items:
Batch Size: 50
Pause: 1000ms (1 second between batches)

Inside Loop:
→ Bulk Insert to Database (50 records at once)

Use Case 3: Sequential Processing with Dependencies

Each iteration depends on previous result:

Loop Over Items:
Batch Size: 1

Inside Loop:
→ Get Previous Result
→ Calculate Next Value
→ Store Result

Manual Loop Implementation (Code Node):

```
const results = [];  
  
for (const item of items) {  
  // Process each item  
  const processed = await processItem(item);  
  results.push(processed);  
  
  // Rate limiting  
  await new Promise(resolve => setTimeout(resolve, 100));  
}  
  
return results;
```

Parallel Execution

When to Use Parallel Execution:

- Independent operations (no data dependencies)
- Multiple API calls to different services
- Concurrent database queries
- File processing in parallel

Pattern 1: Split and Merge

Input Data
↓
Split to Multiple Branches
├→ Branch 1: Get Weather Data
├→ Branch 2: Get Stock Prices
└→ Branch 3: Get News Headlines
↓
Merge Results

↓
Combine All Data

Pattern 2: Fan-Out Fan-In

Trigger (New Order)
↓
Fan Out (3 parallel paths):
├→ Path 1: Update Inventory
├→ Path 2: Send Confirmation Email
└→ Path 3: Notify Shipping Department
↓
Wait for All (Merge)
↓
Mark Order as Processed

Implementation with Split Out:

HTTP Request (Get Customers)
↓ (Returns array of 100 customers)
Split Out
↓ (Now 100 separate items)
Process Each Customer (runs in parallel)
↓
Aggregate Results

Limitations:

- n8n processes items sequentially by default
- True parallel execution requires queue mode + multiple workers
- Use sub-workflows for complex parallel logic

Sub-workflows

What are Sub-workflows?

Reusable workflow components that can be called from other workflows.

Use Cases:

1. **Error Handling:** Centralized error notification
2. **Data Validation:** Reusable validation logic
3. **Notifications:** Standard email/Slack templates
4. **Data Transformation:** Common formatting rules
5. **Modular Architecture:** Break complex workflows into pieces

Example: Error Notification Sub-workflow

Sub-workflow: "Send Error Alert"

Manual Trigger (accepts input data)
↓
Extract Error Details:

- Workflow Name: {{ \$json.workflowName }}
 - Error Message: {{ \$json.error }}
 - Timestamp: {{ \$json.timestamp }}
- ↓
- Format Email:
- Subject: "⚠ Workflow Error: {{ \$json.workflowName }}"
- Body: ""
- Workflow: {{ \$json.workflowName }}
- Error: {{ \$json.error }}
- Time: {{ \$json.timestamp }}
- Execution ID: {{ \$json.executionId }}
- ""
- ↓
- Send to Admin Email
- ↓
- Log to Error Database

Main Workflow: Using Sub-workflow

Main Processing

↓

TRY:

→ Complex Operation

↓

CATCH Error:

→ Execute Workflow Node

Workflow: "Send Error Alert"

Input Data: {

"workflowName": "{{ \$workflow.name }}",

"error": "{{ \$errorMessage }}",

"timestamp": "{{ \$now.toISO() }}",

"executionId": "{{ \$execution.id }}"

}

Sub-workflow Best Practices:

1. **Input Validation:** Check required fields at start
2. **Clear Naming:** Use descriptive workflow names
3. **Documentation:** Add sticky notes explaining inputs/outputs
4. **Version Control:** Use workflow versions for changes
5. **Error Handling:** Handle errors within sub-workflow

Dynamic Workflows

What are Dynamic Workflows?

Workflows that change behavior based on configuration or runtime data.

Use Case 1: Multi-Tenant Workflows

Webhook (Customer Request)

↓

Get Customer Config from Database:

```
SELECT * FROM customer_config WHERE id = {{ $json.customerId }}
```

↓

HTTP Request (Dynamic Endpoint):

URL: {{ \$('Get Config').item.json.apiEndpoint }}

Headers: {{ \$('Get Config').item.json.headers }}

↓

Process Response

Use Case 2: Conditional Node Execution

Get Workflow Configuration

↓

IF config.enableEmailNotification:

→ Send Email

ELSE:

→ Skip

↓

IF config.enableSlackNotification:

→ Send Slack Message

Use Case 3: Dynamic Routing

Get User Preferences

↓

Switch (based on notification_channel):

Case 'email' → Send Email

Case 'sms' → Send SMS

Case 'slack' → Send Slack

Case 'webhook' → Call Custom Webhook

Configuration Management:

Store configuration in:

- Database table
- JSON file in Google Drive
- Environment variables
- n8n Variables (n8n Cloud)

Example: Database-Driven Configuration

```
CREATE TABLE workflow_configs (  
  workflow_name VARCHAR(255),  
  config_key VARCHAR(255),  
  config_value TEXT,  
  updated_at TIMESTAMP  
);
```

-- Query in workflow

```
SELECT config_value
```

```
FROM workflow_configs
```

```
WHERE workflow_name = '{{ $workflow.name }}'
```

```
AND config_key = 'api_endpoint';
```

Chapter 8: Error Handling & Reliability

Try/Catch Patterns

Error Trigger Workflow:

Error Trigger (Global)

↓

Extract Error Information:

- Workflow: {{ `$json.workflow.name` }}
- Node: {{ `$json.node.name` }}
- Error: {{ `$json.errorMessage` }}
- Stack: {{ `$json.error.stack` }}

↓

Format Notification

↓

Send to Admin (Email/Slack)

↓

Log to Error Database

Node-Level Error Handling:

HTTP Request Settings:

Continue on Fail: true

Retry on Fail: true

Max Tries: 3

After HTTP Request:

IF Node:

Condition: {{ `$('HTTP Request').item.json.error !== undefined` }}

True → Error Handling Path

False → Success Path

Code Node Try/Catch:

```
const results = [];
```

```
for (const item of items) {
```

```
  try {
```

```
    // Risky operation
```

```
    const result = await processItem(item);
```

```
    results.push({
```

```
      json: { status: 'success', data: result }
```

```
    });
```

```
  } catch (error) {
```

```
    // Handle error gracefully
```

```
    results.push({
```

```
      json: {
```

```
        status: 'error',
```

```
        error: errorMessage,
```

```
        itemData: item.json
```

```
}  
});  
}  
}
```

return results;

Execution Modes

Main Process Mode (Default)

Workflow executions run in the main n8n process

Pros:

- Simple setup
- No additional infrastructure
- Good for development

Cons:

- Limited scalability
- Long-running workflows block others
- No horizontal scaling

Queue Mode (Production Recommended)

Workflow executions queued and processed by workers

Architecture:

n8n Main Process (Web UI + Webhook Server)

↓

Queue (Redis)

↓

Worker 1 | Worker 2 | Worker 3 (Process Executions)

Configuration:

Main process

EXECUTIONS_PROCESS=queue

QUEUE_BULL_REDIS_HOST=redis

QUEUE_BULL_REDIS_PORT=6379

Worker process

docker run n8nio/n8n worker

Pros:

- Horizontal scaling (add more workers)
- Isolated execution (failures don't affect main process)

- Better resource utilization
- Handles high volume

Cons:

- Requires Redis
- More complex setup
- Slightly higher latency

Logging and Debugging

Workflow Execution History:

n8n UI → Executions

- View all past executions
- Filter by status (success/error)
- See execution time
- Inspect node outputs

Console Logging (Code Node):

```
console.log('Processing item:', item.json);
console.log('API Response:', response);
console.log('Calculated value:', total);

// Log entire item structure
console.log(JSON.stringify(items[0], null, 2));

return items;
```

Custom Logging to External System:

After Critical Step:

↓

HTTP Request (Log to External Service):

Method: POST

URL: <https://logging-service.com/log>

Body: {

```
"level": "info",
"workflow": "{{ $workflow.name }}",
"execution": "{{ $execution.id }}",
"message": "Processed {{ $itemIndex + 1 }} items",
"timestamp": "{{ $now.toISO() }}"
}
```

Error Logging Pattern:

Error Trigger

↓

Format Error Log:

```
{
"timestamp": "{{ $now.toISO() }}",
"workflow": "{{ $json.workflow.name }}"
}
```

```
"node": "{{ $json.node.name }}",  
"error": "{{ $json.errorMessage }}",  
"executionId": "{{ $json.execution.id }}",  
"stack": "{{ $json.error.stack }}"  
}
```

↓

PostgreSQL Insert:

```
INSERT INTO error_logs (timestamp, workflow, node, error, execution_id, stack)  
VALUES (...)
```

Debugging Tips:

1. **Use Test Mode:** Execute workflow manually to see each node output
2. **Check Item Data:** Expand items in node output panel
3. **Add Sticky Notes:** Document expected vs actual behavior
4. **Use Code Node Logging:** Add console.log throughout code
5. **Check Executions History:** Review past runs for patterns
6. **Validate Expressions:** Use expression editor to test syntax
7. **Enable Continue on Fail:** See which node fails without stopping workflow

Building Resilient Workflows

Resilience Checklist:

1. Input Validation

At Workflow Start:

↓

Validate Input Data:

IF Node:

Conditions:

- Required fields exist
- Data types correct
- Values in valid range

↓

TRUE → Continue

FALSE → Return Error Response

2. Timeout Configuration

HTTP Request:

Timeout: 30000ms (30 seconds)

Long-Running Operation:

Timeout: 300000ms (5 minutes)

3. Idempotency

Design workflows to produce same result if run multiple times:

Before Insert:

↓

Check if Record Exists:

```
SELECT * FROM orders WHERE order_id = '{{ $json.orderId }}'
```

↓

IF Exists:

→ Skip Insert (Idempotent)

ELSE:

→ Insert New Record

4. Dead Letter Queue

After Multiple Retries Failed:

↓

Send to Dead Letter Queue:

→ Store failed item in database

→ Flag for manual review

→ Alert admin

5. Circuit Breaker Implementation

// Code node - Circuit breaker logic

```
const failures = await getFailureCount('api-service');
```

```
if (failures > 5) {
```

```
// Circuit is open - don't attempt call
```

```
return [{
```

```
  json: {
```

```
    error: 'Circuit breaker open',
```

```
    message: 'Service temporarily unavailable'
```

```
  }
```

```
  }];
```

```
}
```

```
try {
```

```
// Attempt API call
```

```
const result = await apiCall();
```

```
await resetFailureCount('api-service');
```

```
return [{ json: result }];
```

```
} catch (error) {
```

```
  await incrementFailureCount('api-service');
```

```
  throw error;
```

```
}
```

6. Health Checks

Schedule Trigger (Every 5 minutes)

↓

Test Critical Services:

├→ Ping Database

├→ Test API Endpoint

└→ Check Disk Space

↓

IF Any Service Down:

→ Alert Operations Team

7. Graceful Degradation

Try Primary Service

↓

IF Fails:

→ Try Backup Service

↓

IF Also Fails:

→ Use Cached Data

↓

IF No Cache:

→ Return Graceful Error Message

Chapter 9: Scaling & Performance

Workflow Optimization

Performance Anti-Patterns:

1. N+1 Query Problem

✗ Bad:

Get List of 100 Users

↓

Loop Each User:

→ Query Database for User Details (100 queries!)

✓ Good:

Get List of User IDs

↓

Single Database Query:

SELECT * FROM users WHERE id IN (...) (1 query)

2. Unnecessary Data Transfer

✗ Bad:

HTTP Request:

Get ALL customer data (1000 fields)

↓

Use only email field

✓ Good:

HTTP Request:

Get only needed fields: ?fields=email,name

3. Blocking Operations

✗ Bad:

Loop 1000 Items Sequentially

(Total time: $1000 \times 2 \text{ seconds} = 33 \text{ minutes}$)

✓ Good:

Split to Parallel Batches

Process 10 batches of 100 items
(Total time: 10×2 seconds = 20 seconds)

Optimization Techniques:

1. Batch Operations

Instead of:

Loop 1000 times:

→ Insert 1 record to database

Do:

Collect all 1000 records

→ Bulk insert in single query

2. Conditional Execution

Before HTTP Request:

↓

Check Cache:

IF data exists in cache AND fresh:

→ Use cached data

ELSE:

→ Make API call

→ Update cache

3. Minimize Transformations

Instead of:

HTTP Request

→ Set Node (transform)

→ Code Node (transform)

→ Set Node (transform)

Do:

HTTP Request

→ Single Code Node (all transformations)

4. Filter Early

Get 10,000 Records

↓

Filter (keep only active: 100 records)

↓

Process Only 100 Records

Queue Mode

When to Use Queue Mode:

- Processing > 1000 executions/day
- Long-running workflows (> 1 minute)
- Need horizontal scaling
- Production environment

Setup with Docker Compose:

version: '3.8'

services:

redis:

image: redis:7

container_name: n8n_redis

postgres:

image: postgres:15

environment:

POSTGRES_DB: n8n

POSTGRES_USER: n8n

POSTGRES_PASSWORD: n8n_password

volumes:

- postgres_data:/var/lib/postgresql/data

n8n:

image: n8nio/n8n:latest

container_name: n8n_main

ports:

- "5678:5678"

environment:

- DB_TYPE=postgresdb

- DB_POSTGRESDB_HOST=postgres

- EXECUTIONS_PROCESS=queue

- QUEUE_BULL_REDIS_HOST=redis

- N8N_ENCRYPTION_KEY=your_encryption_key

depends_on:

- postgres

- redis

n8n-worker-1:

image: n8nio/n8n:latest

container_name: n8n_worker_1

command: worker

environment:

- DB_TYPE=postgresdb

- DB_POSTGRESDB_HOST=postgres

- QUEUE_BULL_REDIS_HOST=redis

- N8N_ENCRYPTION_KEY=your_encryption_key

depends_on:

- postgres

- redis

n8n-worker-2:

image: n8nio/n8n:latest

container_name: n8n_worker_2

command: worker

environment:

- DB_TYPE=postgresdb

- DB_POSTGRESDB_HOST=postgres
- QUEUE_BULL_REDIS_HOST=redis
- N8N_ENCRYPTION_KEY=your_encryption_key
depends_on:
- postgres
- redis

volumes:
postgres_data:

Scaling Workers:

Add more workers

docker-compose up -d --scale n8n-worker=5

Concurrency

Worker Concurrency Settings:

Environment variable

EXECUTIONS_CONCURRENCY_MAX=10

Each worker can process up to 10 executions simultaneously

Concurrency Considerations:

1. **CPU-bound workflows:** Set concurrency = CPU cores
2. **I/O-bound workflows** (API calls): Set concurrency = 10-50
3. **Database-heavy:** Monitor DB connection pool

Rate Limiting:

Loop Over Items:

Batch Size: 10

Pause Between Batches: 1000ms

→ Ensures max 10 requests per second

Handling Large Datasets

Strategy 1: Streaming/Pagination

Initial Request (Page 1)

↓

Process Batch

↓

Store Results

↓
Check for Next Page
↓
IF Next Page Exists:
→ Fetch Next Page
→ Repeat

Strategy 2: Split Processing

Get 100,000 Records
↓
Split into 100 Sub-workflows:
Each processes 1,000 records
↓
Aggregate Results

Strategy 3: Incremental Processing

Schedule Trigger (Daily)
↓
Get Last Processed ID from Database
↓
Query: SELECT * FROM table WHERE id > {{ lastId }} LIMIT 10000
↓
Process Records
↓
Update Last Processed ID

Memory Management:

```
// Bad: Load everything into memory
const allData = await fetchAllRecords(); // 1GB of data!

// Good: Process in chunks
for await (const chunk of fetchRecordChunks(1000)) {
  processChunk(chunk);
} // Chunk is garbage collected after processing
```

Database Optimization:

```
-- Add indexes for queries
CREATE INDEX idx_user_email ON users(email);
CREATE INDEX idx_order_date ON orders(created_at);

-- Use efficient queries
SELECT id, name, email
FROM users
WHERE active = true
LIMIT 1000 OFFSET 0;
```

Performance Monitoring:

After Each Major Step:

↓

Log Performance Metrics:

```
{
  "step": "API Call",
  "duration": {{ now.diff(execution.startedAt, 'seconds').seconds }},
  "itemsProcessed": {{ $itemIndex + 1 }},
  "timestamp": "{{ $now.toISO() }}"
}
```

Part IV: Security & Production

Chapter 10: Security & Governance

Credentials Management

n8n Credential System:

n8n stores credentials separately from workflows, encrypted at rest using N8N_ENCRYPTION_KEY.

Credential Types:

1. **Predefined:** OAuth2, API Key, Basic Auth for specific services
2. **Generic:** Header Auth, Query Auth, Custom Auth
3. **Custom:** Build your own credential type

Best Practices:

1. Always Set Encryption Key

CRITICAL: Set before first use

```
N8N_ENCRYPTION_KEY=$(openssl rand -hex 32)
```

Store securely (environment variable or secret manager)

Without encryption key: Credentials stored in plain text in database!

2. Use Workflow-Specific Credentials

✗ Don't: Share one "Production API" credential across all workflows

✓ Do: Create separate credentials:

- "Stripe - Billing Workflow"
- "Stripe - Customer Sync"
- "Stripe - Reporting"

Why? Limits blast radius if one workflow is compromised.

3. Credential Naming Convention

Format: [Service] - [Environment] - [Purpose]

Examples:

- "Salesforce - Production - Read Only"
- "AWS S3 - Staging - Upload"
- "Database - Production - Admin"

4. Regular Credential Rotation

Schedule Trigger (Monthly)

↓

Alert: "Rotate credentials for workflows X, Y, Z"

↓

Track Last Rotation Date

5. Least Privilege Principle

✗ Don't: Use admin API key for read-only operations

✓ Do: Create read-only API key for reporting workflows

6. Audit Credential Usage

PostgreSQL Query:

```
SELECT
workflow_name,
credential_name,
last_used_at,
usage_count
FROM credential_usage_log
WHERE last_used_at < NOW() - INTERVAL '90 days';
```

→ Identify unused credentials for cleanup

Environment Variables

Using Environment Variables:

Set in environment

```
export API_ENDPOINT=https://api.production.com
export MAX_RETRIES=5
export NOTIFICATION_EMAIL=admin@company.com
```

Access in n8n expressions

```
{{ $env.API_ENDPOINT }}  
{{ $env.MAX_RETRIES }}  
{{ $env.NOTIFICATION_EMAIL }}
```

Benefits:

- Separate configuration from workflow logic
- Easy environment switching (dev/staging/prod)
- Secure sensitive values
- Centralized configuration

Docker Environment Variables:

```
services:  
  n8n:  
    image: n8nio/n8n:latest  
    environment:  
      # n8n Configuration  
      - N8N_ENCRYPTION_KEY=${ENCRYPTION_KEY}
```

```
# Custom Variables  
- API_ENDPOINT=${API_ENDPOINT}  
- DATABASE_URL=${DATABASE_URL}  
- SLACK_WEBHOOK=${SLACK_WEBHOOK}
```

Kubernetes Secrets:

apiVersion: v1

kind: Secret

metadata:

name: n8n-secrets

type: Opaque

data:

api-key: base64_encoded_value

database-password: base64_encoded_value

```
apiVersion: apps/v1  
kind: Deployment  
spec:  
  template:  
    spec:  
      containers:  
        - name: n8n
```

env:
- name: API_KEY
valueFrom:
secretKeyRef:
name: n8n-secrets
key: api-key

Secrets Handling

Secrets Management Hierarchy:

1. **Best:** External Secret Manager (AWS Secrets Manager, HashiCorp Vault)
2. **Good:** Kubernetes Secrets + encryption at rest
3. **Acceptable:** Environment variables (not in code)
4. **Bad:** Hardcoded in workflows

AWS Secrets Manager Integration:

HTTP Request Node:
URL: <https://secretsmanager.us-east-1.amazonaws.com/>
Authentication: AWS Signature V4
Action: GetSecretValue
SecretId: "production/api-keys"
↓
Code Node (Parse Secret):

```
const secret = JSON.parse(items[0].json.SecretString);  
return [{ json: { apiKey: secret.apiKey } }];
```


↓
Use {{ \$json.apiKey }} in subsequent nodes

Never:

- Store secrets in workflow JSON
- Log secrets to console
- Send secrets in error messages
- Store secrets in version control

Always:

- Use n8n credentials for API keys
- Rotate secrets regularly
- Audit secret access
- Encrypt secrets at rest

Role-Based Access Control (RBAC)

n8n Cloud/Enterprise Features:

- User roles: Owner, Admin, Member, Viewer
- Workflow sharing permissions
- Credential sharing controls
- Audit logs

Self-Hosted RBAC:

- Basic: Single user (use Basic Auth)
- Advanced: Proxy with authentication (OAuth2, SAML)

Proxy Authentication Setup:

Nginx reverse proxy with OAuth2

```
location / {
  auth_request /oauth2/auth;
  proxy_pass http://n8n:5678;
}
```

```
location /oauth2/ {
  proxy_pass http://oauth2-proxy:4180;
}
```

Access Control Checklist:

- ☐ Limit who can create workflows
- ☐ Limit who can edit credentials
- ☐ Limit who can activate/deactivate workflows
- ☐ Limit who can view execution data
- ☐ Audit user actions
- ☐ Enforce strong passwords
- ☐ Enable 2FA where possible

Compliance Considerations

GDPR Compliance:

1. **Data Minimization:** Only collect necessary data
2. **Right to Erasure:** Ability to delete user data
3. **Data Portability:** Export user data
4. **Consent Management:** Track consent in workflows

Example: GDPR Delete Request Workflow

Webhook (Delete Request)

↓

Validate Request (User ID)

↓

Delete from All Systems:

└→ Delete from CRM

└→ Delete from Email Service

└→ Delete from Analytics

└→ Delete from Database

↓

Log Deletion (Audit Trail)

↓

Confirm to User

SOC 2 Compliance:

- Enable audit logging
- Implement access controls
- Encrypt data in transit and at rest
- Regular security reviews
- Incident response procedures

HIPAA Compliance (Healthcare):

- Use encryption for PHI (Protected Health Information)
- Implement access controls
- Audit all PHI access
- Business Associate Agreements (BAAs) with vendors
- Use compliant hosting (n8n self-hosted on HIPAA-compliant infrastructure)

Audit Logging:

After Each Workflow Execution:

↓

Log Audit Event:

```
{
  "timestamp": "{{ $now.toISO() }}",
  "user": "{{ $execution.user }}",
  "workflow": "{{ $workflow.name }}",
  "action": "executed",
  "executionId": "{{ $execution.id }}",
  "status": "{{ $execution.status }}",
  "dataAccessed": ["customer_email", "payment_info"]
}
```

↓

Store in Audit Database

Chapter 11: Deployment & Production

Docker & Kubernetes Deployment

Production Docker Compose:

version: '3.8'

services:

postgres:

image: postgres:15

restart: always

environment:

POSTGRES_DB: n8n

POSTGRES_USER: \${DB_USER}

POSTGRES_PASSWORD: \${DB_PASSWORD}

volumes:

- postgres_data:/var/lib/postgresql/data

healthcheck:

test: ["CMD-SHELL", "pg_isready -U \${DB_USER}"]

interval: 10s

timeout: 5s

retries: 5

redis:

image: redis:7

restart: always

healthcheck:

test: ["CMD", "redis-cli", "ping"]

interval: 10s

timeout: 3s

retries: 3

n8n:

image: n8nio/n8n:latest

restart: always

ports:

- "5678:5678"

environment:

- DB_TYPE=postgresdb

- DB_POSTGRESDB_HOST=postgres

- DB_POSTGRESDB_DATABASE=n8n

- DB_POSTGRESDB_USER= $DB_USER - DB_POSTGRESDB_PASSWORD =$
{DB_PASSWORD}

- EXECUTIONS_PROCESS=queue

- QUEUE_BULL_REDIS_HOST=redis

- N8N_ENCRYPTION_KEY= $ENCRYPTION_KEY - N8N_HOST =$ {DOMAIN}

- N8N_PROTOCOL=https

- WEBHOOK_URL=https://{DOMAIN}/

- N8N_LOG_LEVEL=info

- EXECUTIONS_DATA_PRUNE=true

- EXECUTIONS_DATA_MAX_AGE=168

volumes:

- n8n_data:/home/node/.n8n

depends_on:

postgres:

condition: service_healthy

redis:

condition: service_healthy

n8n-worker:

image: n8nio/n8n:latest

restart: always

command: worker

environment:

- DB_TYPE=postgresdb

- DB_POSTGRESDB_HOST=postgres

- DB_POSTGRESDB_DATABASE=n8n

- DB_POSTGRESDB_USER= $DB_USER - DB_POSTGRESDB_PASSWORD =$
{DB_PASSWORD}

- QUEUE_BULL_REDIS_HOST=redis

- N8N_ENCRYPTION_KEY=\${ENCRYPTION_KEY}

```
depends_on:
  postgres:
    condition: service_healthy
  redis:
    condition: service_healthy
deploy:
  replicas: 3

volumes:
  postgres_data:
  n8n_data:
```

Kubernetes Production Deployment:

Namespace

```
apiVersion: v1
kind: Namespace
metadata:
  name: n8n-production
```

PostgreSQL StatefulSet

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
  namespace: n8n-production
spec:
  serviceName: postgres
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:15
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_DB
              value: "n8n"
            - name: POSTGRES_USER
              valueFrom:
```

```
secretKeyRef:
name: n8n-secrets
key: db-user
- name: POSTGRES_PASSWORD
valueFrom:
secretKeyRef:
name: n8n-secrets
key: db-password
volumeMounts:
- name: postgres-storage
mountPath: /var/lib/postgresql/data
volumeClaimTemplates:
  • metadata:
    name: postgres-storage
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 50Gi
```

n8n Main Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
name: n8n-main
namespace: n8n-production
spec:
replicas: 1
selector:
matchLabels:
app: n8n-main
template:
metadata:
labels:
app: n8n-main
spec:
containers:
- name: n8n
image: n8nio/n8n:latest
ports:
- containerPort: 5678
env:
- name: DB_TYPE
value: "postgresdb"
- name: DB_POSTGRESDB_HOST
value: "postgres"
- name: N8N_ENCRYPTION_KEY
valueFrom:
```

secretKeyRef:
name: n8n-secrets
key: encryption-key
- name: EXECUTIONS_PROCESS
value: "queue"
- name: N8N_RUNNERS_ENABLED
value: "true"
resources:
requests:
memory: "512Mi"
cpu: "500m"
limits:
memory: "2Gi"
cpu: "2000m"

n8n Worker Deployment

apiVersion: apps/v1
kind: Deployment
metadata:
name: n8n-worker
namespace: n8n-production
spec:
replicas: 5
selector:
matchLabels:
app: n8n-worker
template:
metadata:
labels:
app: n8n-worker
spec:
containers:
- name: n8n-worker
image: n8nio/n8n:latest
command: ["n8n", "worker"]
env:
- name: DB_TYPE
value: "postgresdb"
- name: N8N_ENCRYPTION_KEY
valueFrom:
secretKeyRef:
name: n8n-secrets
key: encryption-key
resources:
requests:
memory: "512Mi"
cpu: "500m"
limits:

memory: "2Gi"
cpu: "2000m"

Horizontal Pod Autoscaler

apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
 name: n8n-worker-hpa
 namespace: n8n-production
spec:
 scaleTargetRef:
 apiVersion: apps/v1
 kind: Deployment
 name: n8n-worker
 minReplicas: 3
 maxReplicas: 20
 metrics:

- type: Resource
 resource:
 name: cpu
 target:
 type: Utilization
 averageUtilization: 70

CI/CD for Workflows

Version Control Strategy:

Export workflow to JSON

n8n export:workflow --id=workflow_id --output=/workflows/

Commit to Git

```
git add workflows/  
git commit -m "Add customer onboarding workflow"  
git push origin main
```

GitHub Actions Workflow:

name: Deploy n8n Workflows

on:
 push:
 branches: [main]
 paths:
 - 'workflows/**'

```
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
```

```
- name: Import Workflows
  run: |
    for workflow in workflows/*.json; do
      curl -X POST \
        https://n8n.company.com/api/v1/workflows \
        -H "X-N8N-API-KEY: ${ secrets.N8N_API_KEY }" \
        -H "Content-Type: application/json" \
        -d @$workflow
    done
```

Workflow Testing:

workflow-test.yml

```
name: Test Workflows

on: [pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
```

```
- name: Start n8n Test Instance
  run: docker-compose -f docker-compose.test.yml up -d

- name: Wait for n8n
  run: sleep 30

- name: Import Test Workflows
  run: |
    for workflow in workflows/*.json; do
      n8n import:workflow --input=$workflow
    done
```

```
- name: Execute Test Workflows
run: |
  n8n execute --id=test-workflow-1
  n8n execute --id=test-workflow-2

- name: Verify Results
run: ./scripts/verify-test-results.sh
```

Backup & Restore

Database Backup:

PostgreSQL backup script

```
#!/bin/bash
```

```
BACKUP_DIR="/backups/n8n"
TIMESTAMP={TIMESTAMP}.sql"
```

Create backup

```
docker exec n8n_postgres pg_dump -U n8n_user n8n > BACKUP_DIR/{BACKUP_FILE}
```

Compress

```
gzip BACKUP_DIR/{BACKUP_FILE}
```

Upload to S3

```
aws s3 cp BACKUP_DIR/{BACKUP_FILE}.gz s3://n8n-backups/
```

Delete local backup older than 7 days

```
find {BACKUP_DIR} -name "*.sql.gz" -mtime +7 -delete
```

Automated Backup Workflow:

Schedule Trigger (Daily 2 AM)

↓

Execute PostgreSQL Backup:

```
docker exec n8n_postgres pg_dump -U n8n_user n8n
```

↓

Compress Backup

↓

Upload to S3/Google Drive:

File: "n8n_backup_{{ \$now.toFormat('yyyyMMdd') }}.sql.gz"

↓

Verify Backup Integrity

↓

IF Success:

→ Log Success

ELSE:

→ Alert Admin

↓

Delete Old Backups (> 30 days)

Restore Process:

Download backup from S3

```
aws s3 cp s3://n8n-backups/n8n_backup_20260203.sql.gz .
```

Decompress

```
gunzip n8n_backup_20260203.sql.gz
```

Stop n8n

```
docker-compose stop n8n n8n-worker
```

Restore database

```
docker exec -i n8n_postgres psql -U n8n_user n8n < n8n_backup_20260203.sql
```

Start n8n

```
docker-compose up -d
```

Monitoring

Health Check Workflow:

Schedule Trigger (Every 5 minutes)

↓

Check n8n API:

HTTP Request: GET <https://n8n.company.com/healthz>

↓

Check Database:

PostgreSQL: SELECT 1

↓

Check Redis:

HTTP Request: redis PING

↓

IF Any Check Fails:

- Increment Failure Counter
- IF Counter > 3:
- Alert: "n8n system unhealthy"

ELSE:

- Reset Failure Counter
- Log: "Health check passed"

Execution Monitoring:

Schedule Trigger (Every hour)

↓

Query Execution Stats:

SELECT

DATE_TRUNC('hour', finished_at) as hour,

COUNT(*) as total,

SUM(CASE WHEN mode = 'error' THEN 1 ELSE 0 END) as errors,

AVG(execution_time_ms) as avg_duration

FROM executions

WHERE finished_at > NOW() - INTERVAL '1 hour'

GROUP BY hour

↓

Calculate Error Rate:

{{ (\$json.errors / \$json.total) * 100 }}%

↓

IF Error Rate > 5%:

- Alert: "High error rate detected"

Prometheus Metrics (Custom):

// Code node - Export metrics to Prometheus

const metrics = {

workflow_executions_total: items.length,

workflow_errors_total: items.filter(i => i.json.error).length,

workflow_duration_seconds: items[0].json.duration / 1000

};

// Push to Prometheus Pushgateway

await fetch('http://pushgateway:9091/metrics/job/n8n', {

method: 'POST',

body: Object.entries(metrics)

.map(([k, v]) => `\${k} \${v}`)

.join("\n")

});

return items;

Grafana Dashboard Queries:

Execution rate

```
rate(n8n_workflow_executions_total[5m])
```

Error rate

```
rate(n8n_workflow_errors_total[5m]) / rate(n8n_workflow_executions_total[5m])
```

Average execution duration

```
avg(n8n_workflow_duration_seconds)
```

Queue length

```
n8n_queue_length
```

Chapter 12: Extending n8n

Creating Custom Nodes

When to Create Custom Nodes:

- Frequently used integration not in n8n
- Company-specific internal APIs
- Complex reusable logic
- Community contribution

Node Development Setup:

Clone n8n repository

```
git clone https://github.com/n8n-io/n8n.git  
cd n8n
```

Install dependencies

```
npm install
```

Create new node

```
npm run node-dev new
```

Node name: MyCustomNode

Node description: Integrates with My Service

Basic Node Structure:

```
import {
  IExecuteFunctions,
  INodeExecutionData,
  INodeType,
  INodeTypeDescription,
} from 'n8n-workflow';

export class MyCustomNode implements INodeType {
  description: INodeTypeDescription = {
    displayName: 'My Custom Node',
    name: 'myCustomNode',
    group: ['transform'],
    version: 1,
    description: 'Integrates with My Service',
    defaults: {
      name: 'My Custom Node',
    },
    inputs: ['main'],
    outputs: ['main'],
    credentials: [
      {
        name: 'myServiceApi',
        required: true,
      },
    ],
    properties: [
      {
        displayName: 'Operation',
        name: 'operation',
        type: 'options',
        options: [
          {
            name: 'Get Data',
            value: 'getData',
          },
          {
            name: 'Create Item',
            value: 'createItem',
          },
        ],
        default: 'getData',
      },
    ],
  };
}
```

```
},  
],  
};
```

```
async execute(this: IExecuteFunctions): Promise<INodeExecutionData[][]> {  
  const items = this.getInputData();  
  const returnData: INodeExecutionData[] = [];  
  const operation = this.getNodeParameter('operation', 0) as string;
```

```
  for (let i = 0; i < items.length; i++) {  
    if (operation === 'getData') {  
      // Implementation  
      const data = await this.helpers.request({  
        method: 'GET',  
        url: 'https://api.myservice.com/data',  
      });  
  
      returnData.push({ json: data });  
    }  
  }  
  
  return [returnData];
```

```
}  
}
```

Custom Credential Type:

```
import {  
  ICredentialType,  
  INodeProperties,  
} from 'n8n-workflow';  
  
export class MyServiceApi implements ICredentialType {  
  name = 'myServiceApi';  
  displayName = 'My Service API';  
  properties: INodeProperties[] = [  
    {  
      displayName: 'API Key',  
      name: 'apiKey',  
      type: 'string',  
      default: '',  
    },  
    {  
      displayName: 'API Secret',  
      name: 'apiSecret',
```

```
type: 'string',
typeOptions: {
password: true,
},
default: "",
},
];
}
```

Testing Custom Node:

Link node for development

npm run link

Start n8n in development mode

npm run dev

Test node in UI

Create workflow, add custom node, test

Node Development Best Practices

1. Error Handling:

```
try {
const response = await this.helpers.request(options);
return [this.helpers.returnJsonArray(response)];
} catch (error) {
if (this.continueOnFail()) {
return [this.helpers.returnJsonArray({ error: errorMessage })];
}
throw error;
}
```

2. Pagination Support:

```
let hasMore = true;
let page = 1;
const allData: any[] = [];
```

```
while (hasMore) {
const response = await this.helpers.request({
url: https://api.example.com/data?page=${page},
});
```

```

allData.push(...response.items);
hasMore = response.hasMore;
page++;
}

return [this.helpers.returnJsonArray(allData)];

```

3. Binary Data Handling:

```

// Download file
const response = await this.helpers.request({
url: fileUrl,
encoding: null, // Important for binary data
});

items[0].binary = {
data: {
data: response.toString('base64'),
mimeType: 'image/png',
fileName: 'downloaded-image.png',
},
};

```

Community Nodes

Installing Community Nodes:

n8n UI → Settings → Community Nodes → Install
Package Name: n8n-nodes-package-name

Popular Community Nodes:

- n8n-nodes-langchain (AI/LLM integrations)
- n8n-nodes-chromadb (Vector database)
- n8n-nodes-text-analysis (NLP operations)
- n8n-nodes-date-fns (Advanced date operations)

Publishing Community Node:

Package structure

```

my-n8n-nodes/
├── package.json
├── credentials/
│   └── MyServiceApi.credentials.ts
├── nodes/
│   ├── MyCustomNode/
│   │   ├── MyCustomNode.node.ts
│   │   └── MyCustomNode.node.json
└── README.md

```

Publish to npm

npm publish

package.json:

```
{
  "name": "n8n-nodes-my-service",
  "version": "1.0.0",
  "description": "n8n node for My Service",
  "keywords": ["n8n-community-node-package"],
  "n8n": {
    "credentials": [
      "credentials/MyServiceApi.credentials.ts"
    ],
    "nodes": [
      "nodes/MyCustomNode/MyCustomNode.node.ts"
    ]
  }
}
```

Contributing to n8n

Contribution Process:

1. **Fork Repository:** Fork n8n on GitHub
2. **Create Branch:** git checkout -b feature/my-new-node
3. **Develop:** Build node following guidelines
4. **Test:** Write unit tests
5. **Document:** Add documentation
6. **Pull Request:** Submit PR with description

Code Quality Standards:

- TypeScript strict mode
- ESLint compliance
- Unit test coverage > 80%
- Documentation for all parameters
- Error handling implementation

Part V: Real-World Applications

Chapter 13: Business Use Cases

CRM Automation

Use Case: Automated Lead Qualification

Webhook (New Lead from Website)

↓

Enrich Lead Data:

→ HTTP Request (Clearbit API): Get company info

→ HTTP Request ([Hunterio](#)): Verify email

↓

Calculate Lead Score:

Code Node:

score = 0

if (company.employees > 50) score += 20

if (company.revenue > 1M) score += 30

if (email.verified) score += 10

if (job_title includes "Director|VP|C-Level") score += 40

↓

Route by Score:

Switch Node:

High Score (> 70) → Sales Rep Assignment

Medium Score (40-70) → Nurture Campaign

Low Score (< 40) → Generic Newsletter

↓

Create in CRM:

Salesforce/HubSpot: Create Contact + Lead Score

↓

Notify Sales Team (High Score Only):

Slack: "👏 High-quality lead: {{ \$json.name }} from {{ \$json.company }}"

ROI: Saves 10+ hours/week on manual lead qualification.

Marketing Automation

Use Case: Multi-Channel Campaign Management

Schedule Trigger (Monday 9 AM)

↓

Get Campaign Audience:

Database Query: SELECT * FROM customers

WHERE segment = 'engaged'

AND last_purchase > NOW() - INTERVAL '30 days'

↓

Split by Preference:

Switch Node (contact_preference):

'email' → Email Campaign Path

'sms' → SMS Campaign Path

'push' → Push Notification Path

↓

Email Path:

→ Personalize Content (Code Node)

→ Send via SendGrid

→ Track Open/Click Events

↓

SMS Path:

→ Format Message (140 chars)

→ Send via Twilio

→ Track Delivery Status

↓

Push Path:

→ Send via Firebase

→ Track Engagement

↓

Aggregate Results:

Merge All Paths

↓

Update Campaign Dashboard:

→ Store Metrics in Database

→ Send Summary to Marketing Team

Data Pipelines

Use Case: Multi-Source Data Warehouse Sync

Schedule Trigger (Every hour)

↓

Parallel Data Collection:

└→ Branch 1: Salesforce (Get new/updated records)

└→ Branch 2: Google Analytics (Get session data)

└→ Branch 3: Stripe (Get transaction data)

└→ Branch 4: Zendesk (Get support tickets)

↓

Transform Data:

For Each Source:

→ Standardize Schema

→ Handle Missing Fields

→ Convert Data Types

→ Add Metadata (source, timestamp)

↓

Data Quality Checks:

→ Validate Required Fields

→ Check Data Types

→ Detect Duplicates

→ Flag Anomalies

↓

Load to Warehouse:

→ PostgreSQL: UPSERT into fact tables

→ Update Dimension Tables

→ Refresh Materialized Views

↓

Trigger Analytics:

→ dbt Cloud: Run transformation jobs

→ Update BI Dashboards

↓

Monitor & Alert:

IF Errors > Threshold:

→ Alert Data Team

ELSE:

→ Log Success Metrics

AI & LLM Workflows

Use Case: Intelligent Customer Support

Email Trigger (New Support Email)

↓

Extract Email Content:

→ Parse Email Body

→ Extract Attachments

→ Identify Customer (email lookup)

↓

Classify Issue Type:

OpenAI Node:

Prompt: "Classify this support request into:

[Technical, Billing, Sales, Feature Request, Bug Report]

Email: {{ \$json.body }}

Model: gpt-4

↓

Route by Classification:

Switch Node:

'Technical' → Technical Support Path

'Billing' → Billing Department

'Bug Report' → Development Team

↓

Generate Initial Response:

OpenAI Node:

Prompt: "Draft a professional support response for:

Issue Type: {{ \$json.classification }}

Customer: {{ \$json.customerName }}

Issue: {{ \$json.body }}

Include:

- Acknowledgment

- Next steps

- Expected timeline"

↓

Search Knowledge Base:

Vector Database (Pinecone):

→ Embed question

→ Find similar resolved tickets

→ Extract solutions

↓

Combine AI Response + KB Articles:

Code Node:

```
response = aiDraft
if (kbArticles.length > 0) {
  response += "\n\nRelated articles:\n"
  response += kbArticles.map(a => a.title).join("\n")
}
```

↓

Send Response:

Gmail/Outlook: Send email

↓

Create Ticket:

Zendesk/Freshdesk: Create ticket with classification

↓

IF High Priority:

→ Slack: Notify support team

OCR & Document Processing

Use Case: Invoice Processing Automation

Email Trigger (Invoices sent to invoices@company.com)

↓

Extract PDF Attachment:

→ Get binary data from email

↓

OCR Processing:

HTTP Request (Mindee/Tesseract API):

Upload: {{ \$binary.data }}

Document Type: Invoice

↓

Extract Fields:

Code Node (Parse OCR result):

```
{
  "invoiceNumber": extracted,
  "date": extracted,
  "vendor": extracted,
  "total": extracted,
  "lineItems": [...]
}
```

↓

Validate Data:

→ Check required fields exist

→ Validate date format

→ Verify amounts (total = sum of line items)

↓

IF Validation Passes:

→ Create in Accounting System (QuickBooks/Xero)

→ Store PDF in Google Drive (organized by vendor/month)

→ Send Approval Request (if amount > \$1000)

ELSE:

- Flag for Manual Review
- Notify Accounting Team

↓

Update Invoice Tracking:

Database: INSERT invoice record with status

Chapter 14: n8n + AI Integration

n8n version 2.0+ includes native AI capabilities with LangChain integration[17].

Integrating OpenAI / LLMs

Available LLM Nodes:

- OpenAI Chat Model
- Anthropic Claude
- Google Gemini
- Azure OpenAI
- Open Source Models (Ollama, LM Studio)

Basic LLM Workflow:

Manual Trigger (Input: User Question)

↓

OpenAI Chat Model Node:

Model: gpt-4

Messages:

System: "You are a helpful assistant."

User: "{{ \$json.question }}"

Temperature: 0.7

Max Tokens: 500

↓

Format Response:

{{ \$json.choices[0].message.content }}

Advanced LLM Configuration:

```
// Code node - Custom OpenAI call
const response = await fetch('https://api.openai.com/v1/chat/completions', {
  method: 'POST',
  headers: {
    'Authorization': Bearer ${process.env.OPENAI_API_KEY},
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    model: 'gpt-4',
    messages: [
      {
        role: 'system',
        content: 'You are a data analyst.'
      },
```

```
{
  role: 'user',
  content: items[0].json.query
}
],
temperature: 0.3,
max_tokens: 1000,
response_format: { type: 'json_object' } // JSON mode
})
});

return [{ json: await response.json() }];
```

Prompt Chaining

Multi-Step AI Processing:

User Input

↓

Step 1: Analyze Intent

OpenAI:

Prompt: "Analyze the user's intent. Return JSON with:
{intent: string, entities: array, sentiment: string}"

Input: "{{ \$json.userMessage }}"

↓

Step 2: Route by Intent

Switch ({{ \$json.intent }}):

'question' → Answer Path

'complaint' → Escalation Path

'request' → Fulfillment Path

↓

Step 3: Generate Context-Aware Response

OpenAI:

Prompt: "Previous analysis: {{ \$('Step 1').item.json }}
Generate appropriate response for {{ \$json.intent }}"

↓

Step 4: Refine Response

OpenAI:

Prompt: "Review and improve this response:
{{ \$('Step 3').item.json.response }}"

Make it:

- More professional
- Concise (under 200 words)
- Action-oriented"

↓

Send Final Response

Tool Calling with n8n

LangChain Agent with Tools:

AI Agent Node:

Agent Type: OpenAI Functions

Model: gpt-4

Tools:

- Tool 1: Search Knowledge Base

- Tool 2: Get Customer Data

- Tool 3: Create Support Ticket

System Prompt: "You are a customer support agent.

Use available tools to help customers."

Tool Definitions:

Tool 1: Search Knowledge Base

HTTP Request Node:

URL: <https://api.knowledge-base.com/search>

Query: {{ \$json.query }}

Tool 2: Get Customer Data

Database Node:

Query: SELECT * FROM customers WHERE email = '{{ \$json.email }}'

Tool 3: Create Support Ticket

Zendesk Node:

Operation: Create Ticket

Subject: {{ \$json.subject }}

Description: {{ \$json.description }}

Example Execution Flow:

User: "I can't access my account and my subscription ends today."

Agent Reasoning:

1. Calls Tool 2 (Get Customer Data) with email
2. Sees subscription status
3. Calls Tool 1 (Search KB) for "account access issues"
4. Combines information
5. Calls Tool 3 (Create Ticket) for urgent issue
6. Returns response: "I've found your account. Your subscription is active until midnight. I've created an urgent ticket for the access issue. Here are steps to try..."

Building AI Agents Using Workflows

Multi-Agent System:

Webhook (Customer Query)

↓

Orchestrator Agent:

OpenAI:

Prompt: "You are an orchestrator. Analyze this query and decide which specialist agent should handle it:

- Sales Agent
- Support Agent
- Technical Agent"

Query: "{{ \$json.query }}"

↓

Route to Specialist:

Switch ({{ \$json.assignedAgent }}):

'sales' → Sales Agent Path

'support' → Support Agent Path

'technical' → Technical Agent Path

↓

Sales Agent Path:

Sub-workflow: Sales Agent

→ Has access to: Product catalog, Pricing, CRM

→ Generates sales-focused responses

↓

Support Agent Path:

Sub-workflow: Support Agent

→ Has access to: Knowledge base, Ticket system

→ Generates support-focused responses

↓

Technical Agent Path:

Sub-workflow: Technical Agent

→ Has access to: Documentation, API logs, System status

→ Generates technical responses

↓

Quality Check:

OpenAI:

Prompt: "Review this agent response. Check for:

- Accuracy
- Tone
- Completeness

Rate 1-10 and suggest improvements."

↓

IF Quality Score < 7:

→ Regenerate with improvements

ELSE:

→ Send to customer

RAG (Retrieval Augmented Generation):

User Question

↓

Embed Question:

OpenAI Embeddings:

Input: "{{ \$json.question }}"

Model: text-embedding-ada-002

↓

Search Vector Database:

Pinecone/ChromaDB:

Query: {{ \$json.embedding }}

Top K: 5

↓

Retrieve Relevant Documents:

Get top 5 most similar chunks

↓

Generate Answer:

OpenAI:

Prompt: "Answer the question based ONLY on this context:

Context: {{ \$json.retrievedDocs }}

Question: {{ \$json.question }}

If the context doesn't contain the answer, say
'I don't have enough information to answer that.'"

↓

Return Answer with Sources:

```
{  
  "answer": "{{ $json.response }}",  
  "sources": {{ $json.retrievedDocs.map(d => d.source) }}  
}
```

Chapter 15: Future of Automation with n8n

Trends in Workflow Automation

1. AI-Native Automation

Future workflows will be designed around AI capabilities from the ground up:

- Natural language workflow creation
- Self-optimizing workflows
- Predictive automation (act before events occur)
- Autonomous decision-making agents

2. Low-Code + Full-Code Convergence

The line between no-code and code-first is blurring:

- Visual interfaces for 80% of work
- Drop into code for the remaining 20%
- Best of both worlds

3. Event-Driven Everything

Move from schedule-based to pure event-driven:

- Real-time data streaming
- Instant reactions to changes
- Event meshes connecting systems

4. Composable Architecture

Workflows as reusable components:

- Workflow marketplaces
- Plug-and-play automation blocks
- Community-driven libraries

AI-Driven Orchestration

n8n AI Builder (Beta)

Natural language to workflow:

User: "Every time someone fills out the contact form, check if they're a company with 50+ employees, and if so, alert the sales team on Slack."

AI Builder generates:

Webhook Trigger

- HTTP Request (Clearbit - company enrichment)
- IF (employees > 50)
- Slack (Send to #sales channel)

Self-Healing Workflows:

Error Detected

↓

AI Analyzes Error:

- Error type
- Frequency
- Impact

↓

AI Suggests Fix:

"This API endpoint moved. Update URL to: <https://new-api.com>"

↓

AI Applies Fix (with approval):

- Update HTTP Request node
- Test execution
- If successful, mark as resolved

Workflow Optimization AI:

Schedule (Weekly)

↓

Analyze Workflow Performance:

- Execution times
- Error rates

- Resource usage
 - ↓
AI Identifies Bottlenecks:
"HTTP Request to API X takes 45 seconds on average.
Recommendation: Add caching layer."
 - ↓
AI Generates Optimized Version:
 - Adds Redis cache node
 - Adjusts batch sizes
 - Optimizes queries
 - ↓
Present to User:
"I can improve this workflow's speed by 60%.
Review changes and approve?"

Event-Driven Architectures

Event Mesh Pattern:

Event Sources:

- └→ Database Change Events (CDC)
- └→ API Webhooks
- └→ Message Queue (Kafka/RabbitMQ)
- └→ IoT Device Events

↓

Event Router (n8n):

- Filters events
- Transforms payload
- Routes to subscribers

↓

Event Consumers:

- └→ Update Dashboard
- └→ Trigger Workflows
- └→ Send Notifications
- └→ Store in Data Lake

Real-Time Data Streaming:

Kafka Consumer Node

↓

Process Stream in Real-Time:

- Window: 1 minute batches
- Aggregate: Count, Average, Sum
- Detect Anomalies

↓

React to Patterns:

IF spike detected:

- Auto-scale infrastructure
- Alert operations team

The n8n Roadmap (2026+)

Planned Features:

1. **Native Python Support:** Write Python code nodes natively
 2. **Visual Debugger:** Step-through workflow execution
 3. **Workflow Versioning:** Git-like version control in UI
 4. **Marketplace:** Buy/sell workflow templates
 5. **Advanced RBAC:** Granular permissions per workflow
 6. **Multi-Region Deployments:** Global workflow execution
 7. **Workflow Analytics:** Built-in performance insights
 8. **AI Co-Pilot:** AI assistant for workflow building
-

Part VI: Resources

Chapter 16: Appendices

Glossary

Aggregate: Node that combines multiple items into a single item

Authentication: Process of verifying identity to access protected resources

Binary Data: Non-text data like images, PDFs, audio files

Circuit Breaker: Pattern that prevents repeated calls to failing services

Credentials: Stored authentication information (API keys, passwords)

Execution: One complete run of a workflow from trigger to end

Expression: Dynamic value using `{{ }}` syntax with JavaScript

IF Node: Conditional logic node that routes data based on true/false conditions

Item: Single unit of data flowing through workflow (contains json and binary)

JSON: JavaScript Object Notation - data format used throughout n8n

Loop: Process items repeatedly, often with rate limiting

Merge Node: Combines data from multiple workflow branches

Node: Single step in a workflow (trigger, action, or logic)

OAuth2: Secure authentication protocol used by Google, Microsoft, etc.

Pagination: Retrieving large datasets in smaller chunks/pages

Queue Mode: Production execution mode using Redis for job queuing

REST API: Standard way to interact with web services over HTTP

Sub-workflow: Reusable workflow called from other workflows

Switch Node: Routes data to multiple paths based on conditions

Trigger: Node that starts workflow execution (webhook, schedule, etc.)

Webhook: HTTP endpoint that receives data to trigger workflows

Worker: Process that executes queued workflow jobs

Common Patterns

Pattern 1: Webhook → Process → Response

Webhook (POST)

↓

Validate Input

↓

Process Data

↓

Webhook Response (success/error)

Pattern 2: Schedule → Extract → Transform → Load

Schedule Trigger (Daily)

↓

Extract (Get data from source)

↓

Transform (Clean and format)

↓

Load (Save to destination)

Pattern 3: Error Handling

Main Process

↓

Set "Continue on Fail" = true

↓

IF Error:

→ Error Workflow (alert + log)

Pattern 4: Fan-Out Fan-In

Single Input

↓

Split to Multiple Parallel Paths

├→ Process A

├→ Process B

└→ Process C

↓

Merge Results

Pattern 5: Polling with State

Schedule Trigger

↓

Get Last Processed ID
↓
Query: WHERE id > lastId
↓
Process New Records
↓
Update Last Processed ID

Example Workflows

Example 1: Automated Invoice Collection

Email Trigger (invoices@company.com)
↓
Extract PDF Attachment
↓
OCR (Extract invoice data)
↓
Validate Data Quality
↓
Create in Accounting System
↓
Store PDF in Google Drive
↓
Send Slack Notification

Example 2: Customer Onboarding

Webhook (New Signup)
↓
Create User in Auth0
↓
Add to CRM (Salesforce)
↓
Create Stripe Customer
↓
Add to Email List (Mailchimp)
↓
Send Welcome Email Series:
→ Day 0: Welcome
→ Day 2: Tutorial
→ Day 7: Feedback Request

Example 3: Social Media Monitor

Schedule Trigger (Every 15 min)
↓
Search Twitter API (brand mentions)
↓
Sentiment Analysis (OpenAI)
↓
Route by Sentiment:

Positive → Save to "Testimonials"
Neutral → Archive
Negative → Alert Customer Support
↓
Store in Database

Troubleshooting Guide

Issue: "Workflow execution fails silently"

Causes:

- Error occurred but "Continue on Fail" is enabled
- Execution timed out
- Webhook trigger misconfigured

Solutions:

- Check Executions history for error details
- Review node settings for "Continue on Fail"
- Increase timeout values
- Enable Error Trigger workflow

Issue: "Expression returns undefined"

Causes:

- Field doesn't exist in data
- Previous node failed
- Incorrect expression syntax

Solutions:

- Use null coalescing: `{{ $json.field ?? 'default' }}`
- Check previous node output in UI
- Use expression editor to validate syntax
- Add `console.log` in Code node

Issue: "API rate limit exceeded"

Causes:

- Too many requests in short time
- No rate limiting in workflow

Solutions:

- Add Loop Over Items node with pause
- Implement exponential backoff
- Use batch operations where possible
- Cache API responses

Issue: "Webhook not triggering workflow"

Causes:

- Incorrect webhook URL
- Authentication mismatch
- Network/firewall blocking
- Workflow not activated

Solutions:

- Verify webhook URL is correct
- Check authentication settings match sender
- Test webhook with curl/Postman
- Ensure workflow is "Active"
- Check n8n logs for incoming requests

Issue: "High memory usage / crashes"

Causes:

- Processing too much data at once
- Memory leak in Code node
- No pagination for large datasets

Solutions:

- Implement pagination
- Process in smaller batches
- Clear variables in loops
- Use queue mode with separate workers

Further Reading

Official Documentation:

- n8n Docs: <https://docs.n8n.io>
- n8n Community Forum: <https://community.n8n.io>
- n8n GitHub: <https://github.com/n8n-io/n8n>
- n8n YouTube Channel: n8n tutorials and demos

Learning Resources:

- n8n Templates: 3000+ ready-to-use workflows
- n8n Academy: Video courses (official)
- n8n Blog: Best practices and use cases
- n8n Discord: Real-time community help

Related Technologies:

- LangChain Documentation: <https://python.langchain.com>
- Docker Documentation: <https://docs.docker.com>
- Kubernetes Documentation: <https://kubernetes.io/docs>
- PostgreSQL Documentation: <https://www.postgresql.org/docs>

Books:

- *API Design Patterns* by JJ Geewax
- *Building Microservices* by Sam Newman

- *Designing Data-Intensive Applications* by Martin Kleppmann

Courses:

- n8n Fundamentals (official)
 - API Integration Best Practices
 - Kubernetes for Application Developers
 - Building AI Agents with LangChain
-

Conclusion

n8n represents the convergence of no-code accessibility and full-code power. Whether you're automating simple tasks or building sophisticated AI-powered systems, n8n provides the flexibility and control to bring your ideas to life.

Key Takeaways:

1. **Start Simple:** Begin with basic workflows, gradually add complexity
2. **Leverage Community:** 400+ nodes, thousands of templates, active community
3. **Think Modular:** Build reusable sub-workflows and patterns
4. **Embrace AI:** Integrate LLMs for intelligent automation
5. **Scale Thoughtfully:** Use queue mode, workers, and optimization techniques
6. **Secure by Default:** Encryption, credentials management, RBAC
7. **Monitor Everything:** Health checks, error handling, performance metrics

Next Steps:

- Install n8n (Cloud or self-hosted)
- Build 3-5 simple workflows to learn fundamentals
- Explore templates for your use cases
- Join community forum for help and inspiration
- Contribute back: share workflows, build custom nodes

The future of work is automated. With n8n, you're equipped to build that future.

References

- [1] n8n Documentation. (2026). *Welcome to n8n Docs*. <https://docs.n8n.io>
- [2] Digidop. (2026, January 25). *n8n vs Make vs Zapier [2026 Comparison]*. <https://www.digidop.com/blog/n8n-vs-make-vs-zapier>
- [3] Hatchworks. (2025, November 23). *n8n vs Zapier: The Definitive 2026 Automation Face-Off*. <https://hatchworks.com/blog/ai-agents/n8n-vs-zapier/>
- [4] Releasebot. (2026, February 1). *n8n Release Notes - February 2026 Latest Updates*. <https://releasebot.io/updates/n8n>
- [5] n8n Docs. (2017, March 12). *Expressions*. <https://docs.n8n.io/code/expressions/>
- [6] n8n Arena. (2025, June 5). *n8n Expressions Cheat-Sheet*. <https://n8narena.com/guides/n8n-expression-cheatsheet/>

[7] n8n.news. (2025, May 17). *Mastering n8n Error Handling: Best Practices for Workflow Retries*. <https://n8n.news/mastering-n8n-error-handling-best-practices-for-workflow-retries/>

[8] n8n Community. (2025, July 22). *Master List of Every n8n Node*. <https://community.n8n.io/t/master-list-of-every-n8n-node/155146>

[9] AI Fire. (2025, July 31). *Build Custom AI Agents In N8N With LangChain's Code Node*. <https://www.aifire.co/p/build-custom-ai-agents-in-n8n-with-langchain-s-code-node>

[10] Osher. (2025, November 2). *How to Host n8n with Docker*. <https://osher.com.au/blog/how-to-host-n8n-with-docker/>