MÉTHODE ÉPROUVÉE : Créer des Workflows n8n avec Cursor

100% Fiable | Self-Hosted | De la Conception au Publish

🎯 OBJECTIF

Créer une méthode **fiable à 100%** pour générer des workflows n8n via Cursor, sans erreurs, de la conception jusqu'au test et à la publication.

✅ PRÉREQUIS OBLIGATOIRES

1. Infrastructure n8n Self-Hosted

- n8n déployé en Docker sur VPS (DigitalOcean, OVH, Hetzner recommandés)
- Version la plus récente (>= 1.45.0 en 2026)
- Accès API configuré avec clés AUTH (Bearer Token)
- PostgreSQL en backend (pour la stabilité)

Commande Docker optimale :
```
docker run -d
--name n8n
-p 5678:5678
-e DB_TYPE=postgres
-e DB_POSTGRESDB_HOST=postgres
-e DB_POSTGRESDB_USER=n8n
-e DB_POSTGRESDB_PASSWORD=secure_password
n8nio/n8n:latest
```

2. Cursor IDE Configuré

- Cursor 0.35+ avec MCP (Model Context Protocol) activé
- `.cursor/mcp.json` préconfiguré
- Connexion API n8n validée

3. n8n-MCP Server Actif

- Installation : `npm install -g n8n-mcp`
- Configuration endpoint local
- Bearer token n8n généré dans l'interface

🔧 CONFIGURATION CURSOR + n8n-MCP

Étape 1 : Générer une clé API n8n

1. Dans n8n UI → **Paramètres** → **API Tokens**
2. Créer nouveau token Cursor-Integration
3. Copier le Bearer token complet

Étape 2 : Configurer `.cursor/mcp.json`

Créer/modifier le fichier `.cursor/mcp.json` à la racine du projet :

```
{
"mcpServers": {
"n8n": {
"command": "npx",
"args": [
"-y",
"n8n-mcp",
"--url",
"http://localhost:5678",
"--token",
"YOUR_N8N_API_TOKEN_HERE"
]
}
}
}
```

Étape 3 : Créer `.cursorerules` ou `.cursor/rules.md`

N8N Workflow Generation Rules

WORKFLOW DESIGN PRINCIPLES

1. Always start with a clear trigger (webhook, cron, listener)
2. Use descriptive node names with prefixes: [FETCH_], [PARSE_], [SEND_], [CONDITION_]
3. Include error handling on every external API call
4. Add logging nodes (set statement) at critical decision points
5. Use IF nodes for branching logic (not complex code nodes initially)

DATA VALIDATION

- Validate input data structure at first processing node
- Add try/catch in all code nodes
- Include fallback values for missing fields
- Use type checking in JavaScript nodes

API INTEGRATION PATTERNS

- Always set appropriate headers (Content-Type, Authorization)
- Include timeout settings (default 30s)
- Add retry logic for external APIs (3 retries with exponential backoff)
- Log request/response for debugging

TESTING PROTOCOL

- Test each node individually with Execute Step
- Use sample data that mirrors production data
- Check credential validity before workflow activation
- Monitor first 5 executions in real-time

NODE NAMING CONVENTION

[ACTION]*[SERVICE]*[DESCRIPTION]

- [FETCH]_GitHub_GetRepositories
- [PARSE]_JSON_ExtractEmails
- [SEND]_Slack_NotifyChannel
- [CONDITION]_CheckDataValidity
- [ERROR]_HandleAPIFailure

Étape 4 : Redémarrer Cursor

Cmd+Shift+P → "Developer: Reload Window"

Vérifier que n8n-MCP est actif :
Cmd+Shift+P → "MCP Servers" → n8n doit être listé

WORKFLOW DESIGN TEMPLATE (Avant Cursor)

Phase 1 : Conception sur Papier/Notion

Toujours faire ça en premier ⚠

WORKFLOW: [Nom Explicite]

TRIGGER:

- Type: [Webhook/Cron/Listener]
- Frequency: [If scheduled]

NODES SEQUENCE:

1. TRIGGER_ReceiveWebhook
   ↓
2. PARSE_ExtractPayload
   ↓
3. CONDITION_ValidateData
   ├─ YES → 4. FETCH_ExternalAPI

└─ NO → ERROR_LogValidationFailure
↓
4. TRANSFORM_MappingData
↓
5. SEND_SendNotification
↓
6. LOG_ExecutionSuccess

ERROR HANDLING:

- All API nodes → Error Workflow on Failure
- Validation failure → Send alert Slack/Email
- Timeout after 30s → Retry 3x then fail

WORKFLOW GENERATION AVEC CURSOR

PROMPT TEMPLATE OPTIMAL

Toujours utiliser ce format dans Cursor :

@n8n Create a new n8n workflow with the following specifications:

WORKFLOW NAME: [ExactName]
DESCRIPTION: [What it does]

TRIGGER:

- Type: [Webhook/Cron/Listener/etc]
- Frequency: [If scheduled]

WORKFLOW LOGIC:

1. [Input validation]
2. [API call with error handling]
3. [Data transformation]
4. [Send output]

NODES REQUIRED:

- Webhook Trigger
- Set node for logging
- HTTP Request node with retry logic
- IF/Decision node for branching
- [Service] Integration node
- Error handling workflow

SECURITY/VALIDATION:

- Validate all input fields
- Add timeout 30s on all external calls
- Implement 3-retry logic on failures
- Log execution with timestamps

OUTPUT FORMAT:

- Create workflow directly in n8n instance (not JSON)
- Activate the workflow
- Test with provided sample data
- Return execution status and logs

TESTING DATA:
{
"input_field": "test_value",
"api_key": "test_key"
}

EXEMPLE CONCRET : Workflow Webhook → Slack

@n8n Create a workflow named "Webhook_to_Slack_Notifier"

TRIGGER: HTTP Webhook (POST)

LOGIC:

1. Receive webhook JSON payload
2. Extract message, channel, severity
3. Validate message is not empty
4. Format message with timestamp
5. Send to Slack channel
6. Log success or error

VALIDATION:

- message required, min 5 chars
- channel required, format #channelname
- severity optional (info/warning/error)

ERROR HANDLING:

- If Slack API fails → Retry 3x
- If validation fails → Return 400 error
- Log all attempts with timestamps

TEST WITH:
{
"message": "Test notification",
"channel": "#automation",
"severity": "info"
}

✔ PROTOCOL DE VÉRIFICATION (PRE-PUBLICATION)

Checklist Pre-Deployment

- [ ] **Structure validée**
  - [ ] Workflow a un trigger unique
  - [ ] Tous les nodes ont des noms descriptifs
  - [ ] Pas de nodes déconnectés
  - [ ] Pas de nœuds vides/invalides
- [ ] **Credentials & API**
  - [ ] Toutes les credentials configurées
  - [ ] Bearer tokens stockés en variables d'env
  - [ ] Pas de secrets en dur dans le code
  - [ ] Timeouts configurés (30-60s)
- [ ] **Gestion d'erreurs**
  - [ ] Chaque API node a error workflow
  - [ ] Try/catch sur code nodes
  - [ ] Fallback values pour données manquantes
  - [ ] Logs à chaque étape critique
- [ ] **Tests unitaires**
  - [ ] Execute Step sur chaque node individuellement
  - [ ] Valider output avec sample data
  - [ ] Tester branchement IF/ELSE
  - [ ] Vérifier format données en sortie

TESTING PROTOCOL (PRE-ACTIVATION)

Phase 1 : Test Unitaire (Node par Node)

Pour chaque node :

1. Clic droit → "Execute step"
2. Vérifier output dans onglet Output
3. Chercher errors en rouge
4. Valider structure JSON/données

Exemple Test Node API :
```
// Dans Set node après API call
{
"status": $response.status,
"headers": $response.headers,
"body": $response.body,
"timestamp": $now
}
```

Phase 2 : Test Full Workflow

1. Clic "Execute Workflow"
2. Attendre completion (vert = succès)
3. Ouvrir onglet "Executions"
4. Vérifier dernière exécution
5. Cliquer execution → "View Details"
6. Checker chaque node output
7. Valider output final vs attendu

Phase 3 : Test Avec Données Réelles

1. Préparer 5-10 exemples réalistes
2. Envoyer via webhook (curl/Postman)
3. Observer exécutions en real-time
4. Valider output final
5. Checker logs pour warnings

Commande Curl Test :
curl -X POST http://localhost:5678/webhook/test-webhook
-H "Content-Type: application/json"
-d '{
"message": "Test message",
"channel": "#automation"
}'

Phase 4 : Monitoring Pré-Production

Avant ACTIVATION définitive :

- Garder workflow DÉSACTIVÉ 24h
- Monitorer les workflows existants
- Vérifier stabilité n8n (CPU/Memory)
- Checker logs pour erreurs système

ACTIVATION & PUBLICATION

Étape 1 : Activation dans n8n UI

1. Ouvrir workflow
2. Haut droite : Toggle **ON** (devient bleu)
3. Notification "Workflow activated successfully"
4. N8n commence à écouter les triggers

Étape 2 : Configuration Monitoring

Aller à Executions :

- Activer notifications (email/Slack) sur erreurs
- Set retention policy: 30 jours
- Enable logs: tous les nodes

Étape 3 : Setup Alertes

Recommandé : Créer error workflow qui notifie si workflow principal échoue

Error Workflow Nodes:

1. TRIGGER_ExecutionError
2. FORMAT_ErrorMessage (+ stack trace)
3. SEND_SlackNotification (channel: #errors)
4. LOG_ErrorDatabase

Étape 4 : Documentation

Créer README.md dans le repo n8n :

Workflow: [Nom]

Description

[What it does]

Trigger

[How it's triggered]

Dependencies

- Service A (API key required)
- Service B (OAuth)

Error Handling

- On failure: Retries 3x
- If all fail: Notifies Slack #errors

Monitoring

- Check Executions every hour first 48h
- Monitor error logs
- Document any issues

Rollback Plan

- Keep previous version exported
- Can disable workflow anytime

🔄 CYCLE DE CORRECTION (Si Erreurs)

Si le workflow échoue

1. **Localiser l'erreur**
   - Aller Executions → dernière exécution
   - Node en rouge = point de failure
   - Cliquer node → "View Details" pour stack trace
2. **Analyser cause**
   Types erreurs courants :
   - 401: Credential/token invalide
   - 400: Malformed request/invalid data
   - 500: Service API down
   - Timeout: Network/server slow
   - Type Error: JavaScript node code bug
3. **Corriger dans Cursor**
   @n8n Fix the failing node in workflow "X"
   Node name: [NomNode]
   Error: [ErrorMessage]
   Current code: [Coller code]
   Should:
   - [Requirement 1]
   - [Requirement 2]

   Test with: [Sample data]
4. **Re-déployer**
   - Modifier dans Cursor
   - Copier la fix dans n8n UI
   - Re-tester avec Execute Step
   - Ré-activer si workflow étais désactivé

📋 BEST PRACTICES (PRODUCTION)

Naming Convention Stricte

[TYPE]*[SERVICE]*[ACTION]_[DESCRIPTION]

Examples:

- FETCH_GitHub_GetRepos_ByOwner
- PARSE_JSON_ExtractEmails_FromText
- SEND_Slack_Alert_WorkflowError
- CONDITION_Validate_EmailFormat
- TRANSFORM_Map_FieldNames
- ERROR_Retry_APIFailure
- LOG_Store_ExecutionDetails

Code Node Best Practices

```
// ✅ GOOD
try {
const data = $json.body;
if (!data || !data.email) {
return {
error: true,
message: "Email field required",
timestamp: new Date().toISOString()
};
}
return { email: data.email, processed: true };
} catch (error) {
return { error: true, message: error.message };
}

// ❌ BAD
const email = $json.email; // No validation
return email; // No error handling
```

Credentials Management

NEVER:

- Hardcode API keys in nodes
- Store tokens in workflow JSON
- Use personal API keys in prod

DO:

- Create credential in n8n UI
- Reference by name in nodes
- Use env variables for sensitive data
- Rotate tokens every 90 days

Logging Strategy

Every critical node should have:

```
Set node after it:
{
"nodeExecuted": "FETCH_API_Data",
"status": "success",
"timestamp": $now,
"dataCount": $json.length,
"duration_ms": $json.duration
}
```

Then save to database or Slack for audit trail

TROUBLESHOOTING RAPIDE

Problème	Cause	Solution
401 Unauthorized	Token expiré/invalide	Régénérer credential dans n8n
Timeout après 30s	Appel API lent	Augmenter timeout à 60s dans HTTP node
Data structure error	Format JSON inattendu	Ajouter Set node pour valider/mapper
Workflow ne trigger pas	Webhook URL mauvaise	Vérifier URL dans Executions tab
Memory leak	Boucles infinies/pas de limite	Ajouter limites dans loops (max items: 100)
Credential not found	Credential supprimée	Recréer dans Credentials tab
MCP connection lost	n8n-mcp service down	Redémarrer service: `systemctl restart n8n-mcp`

CHECKLIST FINALE PRE-PUBLICATION

CONCEPTION

- [ ] Schéma workflow dessiné sur papier
- [ ] Tous les nodes identifiés
- [ ] Error handling path défini
- [ ] Test data prepared

CURSOR GENERATION

- [ ] Prompt structuré et clair
- [ ] MCP connection verified
- [ ] Workflow créé dans n8n (pas en JSON local)
- [ ] Credentials configurées

TESTING

- [ ] Tous les nodes testés individuellement
- [ ] Full workflow test réussi
- [ ] Tests avec data réelles OK
- [ ] Erreurs gérées correctement

- [ ] Logs affichent les données attendues

PRE-ACTIVATION

- [ ] Aucune erreur non gérée
- [ ] Monitoring configuré
- [ ] Documentation rédigée
- [ ] Error workflow prêt
- [ ] Backup de la version précédente

ACTIVATION

- [ ] Toggle ON sur workflow
- [ ] Première exécution observée
- [ ] Output validé vs attendu
- [ ] Logs monitored 24h
- [ ] No regressions on other workflows

RESSOURCES & RÉFÉRENCES

Official Documentation

- **n8n API Docs**: https://docs.n8n.io/api/
- **n8n Nodes Library**: https://docs.n8n.io/integrations/
- **Model Context Protocol**: https://modelcontextprotocol.io/

Outils Recommandés

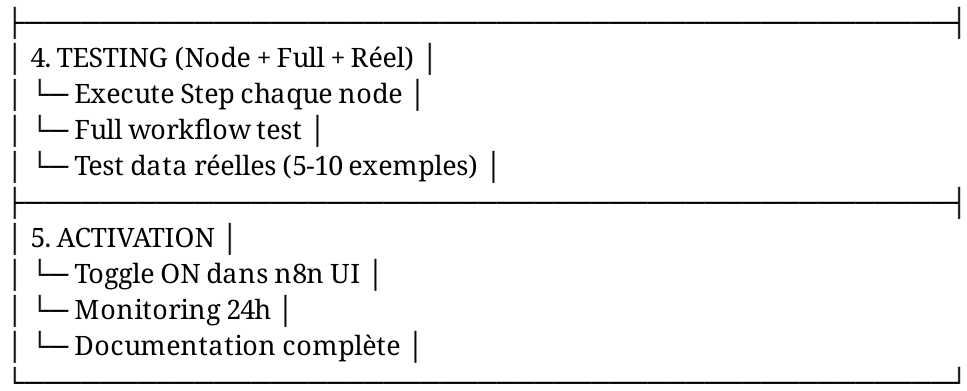
- **n8n-MCP**: https://github.com/n8n-io/n8n-mcp
- **Cursor IDE**: https://www.cursor.com
- **n8n Self-Hosted AI Starter Kit**: https://github.com/n8n-io/self-hosted-ai-starter-kit

Community & Support

- **n8n Forum**: https://community.n8n.io
- **n8n Discord**: https://discord.gg/n8n
- **GitHub Issues**: https://github.com/n8n-io/n8n/issues

RÉSUMÉ DE LA MÉTHODE

```
┌─────────────────────────────────────────────────────────────┐
│ 1. CONCEPTION                                               │
│ └─ Dessiner workflow sur papier                             │
├─────────────────────────────────────────────────────────────┤
│ 2. SETUP CURSOR + n8n-MCP                                   │
│ └─ Configurer .cursor/mcp.json                              │
│ └─ Générer API token n8n                                    │
├─────────────────────────────────────────────────────────────┤
│ 3. GENERATION AVEC CURSOR                                   │
│ └─ Prompt structuré en @n8n                                 │
│ └─ Laisser Cursor créer dans n8n (pas JSON)                 │
```



⚠ PIÈGES À ÉVITER

✕ **Générer JSON en local et copier-coller** → Erreurs format
✓ Laisser Cursor créer directement dans n8n via MCP

✕ **Oublier error handling** → Workflows silencieux qui échouent
✓ Ajouter try/catch et error workflows

✕ **Ne pas tester avant activation** → Production en chaos
✓ Tester 100% avant d'activer

✕ **Hardcoder credentials** → Fuite de sécurité
✓ Utiliser credential store n8n

✕ **Workflow monolithique** → Impossible à déboguer
✓ Découper en petits workflows réutilisables

Version: 1.0 | **Date**: Janvier 2026 | **Status**: Production Ready ✓