

PROJECT GENESIS: The Self-Spawning N8N Nexus Mesh

A master self-organizing mesh of n8n agents that can create, build, design, code, ship, monitor and heal any workflow or infrastructure—entirely by itself.

Core Genesis Architecture

Layer 1: The Primordial Core

The genesis system consists of seven immortal agents that form the backbone:

GENESIS CONTROL PLANE		
1. NEXUS (Master Orchestrator)		
↳ Routes all requests, maintains agent registry		
2. ARCHITECT (Workflow Designer)		
↳ Designs n8n workflows from natural language		
3. BUILDER (Code Generator)		
↳ Generates nodes, functions, integrations		
4. SPAWNER (Deployment Engine)		
↳ Creates new agents, deploys workflows		
5. OBSERVER (Monitoring System)		
↳ Watches all workflows, detects failures		
6. HEALER (Self-Repair Engine)		
↳ Fixes broken workflows, resurrects dead agents		
7. EVOLVER (Learning System)		
↳ Optimizes patterns, creates new agent types		

The Genesis Loop

graph TD
A[User Request] --> B[NEXUS Router]
B --> C{Intent Classification}

```

C -->|Build| D[ARCHITECT]
C -->|Fix| E[HEALER]
C -->|Monitor| F[OBSERVER]
D --> G[BUILDER]
G --> H[SPAWNER]
H --> I[Deploy to n8n]
I --> F
F -->|Failure Detected| E
E --> G
F -->|Success Pattern| J[EVOLVER]
J -->|New Template| K[Knowledge Base]
K --> D

```

Genesis Starter Pack: The Seven Workflows

1. NEXUS - Master Router Workflow

```

{
  "name": "GENESIS_NEXUS_MASTER",
  "nodes": [
    {
      "name": "Genesis Entry",
      "type": "n8n-nodes-base.webhook",
      "webhookId": "genesis-master",
      "position": [240, 300]
    },
    {
      "name": "Intent Classifier",
      "type": "n8n-nodes-base.function",
      "parameters": {
        "functionCode": `
const prompt = $json.prompt || "";
const context = {
  hasError: prompt.includes("error") || prompt.includes("broken"),
  needsBuild: prompt.includes("create") || prompt.includes("build"),
  needsMonitoring: prompt.includes("watch") || prompt.includes("monitor"),
  needsData: prompt.includes("fetch") || prompt.includes("scrape"),
  isComplex: prompt.split(" ").length > 20
};

// Smart routing logic
if (context.hasError) return [{json: {route: "healer", urgency: "high"}}];

```

```

        if (context.needsBuild) return [{json: {route: "architect", complexity: context.isComplex ?
"high" : "low"}}];
        if (context.needsMonitoring) return [{json: {route: "observer"}}];

        // Default to architect for new requests
        return [{json: {route: "architect", prompt: prompt}}];
    },
    },
    {
        "name": "Route to Agent",
        "type": "n8n-nodes-base.switch",
        "parameters": {
            "dataPropertyName": "route",
            "values": [
                {"name": "architect", "value": "architect"},
                {"name": "healer", "value": "healer"},
                {"name": "observer", "value": "observer"},
                {"name": "spawner", "value": "spawner"}
            ]
        }
    }
]
}

```

2. ARCHITECT - Workflow Designer

```

{
    "name": "GENESIS_ARCHITECT",
    "nodes": [
        {
            "name": "Design Request",
            "type": "n8n-nodes-base.executeWorkflowTrigger"
        },
        {
            "name": "Load Knowledge Base",
            "type": "n8n-nodes-base.googleDrive",
            "parameters": {
                "operation": "download",
                "fileId": "{{ $env.GENESIS_KNOWLEDGE_DOC }}"
            }
        },
        {
            "name": "AI Workflow Designer",

```

```

"type": "@n8n/n8n-nodes-langchain.agent",
"parameters": {
  "options": {
    "systemMessage": `You are the Genesis Architect. Your role:
      1. Convert natural language to n8n workflow designs
      2. Use patterns from the knowledge base
      3. Output complete, valid n8n JSON
      4. Include error handling and retry logic
      5. Add monitoring hooks for the Observer
      6. Make workflows self-documenting with sticky notes`
  }
},
{
  "name": "Validate Design",
  "type": "n8n-nodes-base.function",
  "parameters": {
    "functionCode": `
      try {
        const workflow = JSON.parse($json.design);
        // Validate structure
        if (!workflow.nodes || !workflow.connections) {
          throw new Error("Invalid workflow structure");
        }
        // Add Genesis metadata
        workflow.meta = {
          ...workflow.meta,
          genesisVersion: "1.0",
          createdBy: "ARCHITECT",
          monitoringEnabled: true
        };
        return [{json: {valid: true, workflow}}];
      } catch (error) {
        return [{json: {valid: false, error: error.message}}];
      }
    `
  }
}
]
}

```

3. BUILDER - Code Generator

```
{
```

```

"name": "GENESIS_BUILDER",
"nodes": [
  {
    "name": "Build Request",
    "type": "n8n-nodes-base.executeWorkflowTrigger"
  },
  {
    "name": "Generate Node Code",
    "type": "@n8n/n8n-nodes-langchain.agent",
    "parameters": {
      "options": {
        "systemMessage": `You are the Genesis Builder. Generate:
          1. Custom function nodes with proper error handling
          2. Integration code for external APIs
          3. Optimized data transformations
          4. Self-healing retry mechanisms
          5. Monitoring telemetry
          Output executable code only.`
      }
    }
  },
  {
    "name": "Test Generated Code",
    "type": "n8n-nodes-base.function",
    "parameters": {
      "functionCode": `
        // Sandbox testing environment
        const testData = [{test: "data"}];
        try {
          const func = new Function('$json', '$items', $json.generatedCode);
          const result = func(testData[0], testData);
          return [{json: {success: true, code: $json.generatedCode}}];
        } catch (error) {
          return [{json: {success: false, error: error.message}}];
        }
      `
    }
  }
]
}

```

4. SPAWNER - Deployment Engine

```

{

```

```
"name": "GENESIS_SPAWNER",
"nodes": [
  {
    "name": "Spawn Request",
    "type": "n8n-nodes-base.executeWorkflowTrigger"
  },
  {
    "name": "Create via n8n API",
    "type": "n8n-nodes-base.httpRequest",
    "parameters": {
      "method": "POST",
      "url": "={{$env.N8N_URL}}/api/v1/workflows",
      "authentication": "predefinedCredentialType",
      "nodeCredentialType": "n8nApi",
      "sendBody": true,
      "bodyParameters": {
        "workflow": "={{$json.workflow}}"
      }
    }
  },
  {
    "name": "Activate Workflow",
    "type": "n8n-nodes-base.httpRequest",
    "parameters": {
      "method": "PATCH",
      "url": "={{$env.N8N_URL}}/api/v1/workflows/{{$json.id}}",
      "sendBody": true,
      "bodyParameters": {
        "active": true
      }
    }
  },
  {
    "name": "Register with Observer",
    "type": "n8n-nodes-base.httpRequest",
    "parameters": {
      "method": "POST",
      "url": "={{$env.N8N_URL}}/webhook/genesis-observer",
      "sendBody": true,
      "bodyParameters": {
        "workflowId": "={{$json.id}}",
        "monitoringLevel": "full",
        "healingEnabled": true
      }
    }
  }
]
```

```

    }
  }
]
}

```

5. OBSERVER - Monitoring System

```

{
  "name": "GENESIS_OBSERVER",
  "nodes": [
    {
      "name": "Monitoring Loop",
      "type": "n8n-nodes-base.scheduleTrigger",
      "parameters": {
        "rule": {
          "interval": [{"field": "minutes", "triggerAtMinute": 5}]
        }
      }
    },
    {
      "name": "Get All Workflows",
      "type": "n8n-nodes-base.httpRequest",
      "parameters": {
        "method": "GET",
        "url": "={{$env.N8N_URL}}/api/v1/workflows",
        "authentication": "predefinedCredentialType"
      }
    },
    {
      "name": "Check Health",
      "type": "n8n-nodes-base.function",
      "parameters": {
        "functionCode": `
const unhealthy = [];
for (const workflow of $json) {
  // Check if Genesis-managed
  if (!workflow.meta?.genesisVersion) continue;

  // Health checks
  const checks = {
    isActive: workflow.active,
    hasRecentExecution: await checkRecentExecution(workflow.id),
    errorRate: await getErrorRate(workflow.id),
    lastError: await getLastError(workflow.id)
  }
}
`
      }
    }
  ]
}

```

```

    };

    if (!checks.isActive || checks.errorRate > 0.1) {
      unhealthy.push({workflow, checks});
    }
  }
  return unhealthy;
},
{
  "name": "Trigger Healing",
  "type": "n8n-nodes-base.executeWorkflow",
  "parameters": {
    "workflowId": "GENESIS_HEALER"
  }
}
]
}

```

6. HEALER - Self-Repair Engine

```

{
  "name": "GENESIS_HEALER",
  "nodes": [
    {
      "name": "Healing Request",
      "type": "n8n-nodes-base.executeWorkflowTrigger"
    },
    {
      "name": "Diagnose Issue",
      "type": "@n8n/n8n-nodes-langchain.agent",
      "parameters": {
        "options": {
          "systemMessage": `You are the Genesis Healer. Analyze workflow errors and:
            1. Identify root cause (API changes, auth issues, data format)
            2. Suggest specific fixes
            3. Generate healing patches
            4. Prevent future occurrences`
        }
      }
    },
    {
      "name": "Apply Healing",

```



```

"type": "n8n-nodes-base.switch",
"parameters": {
  "dataPropertyName": "healingType",
  "values": [
    {"name": "restart", "value": "restart"},
    {"name": "patch", "value": "patch"},
    {"name": "rebuild", "value": "rebuild"}
  ]
},
{
  "name": "Restart Workflow",
  "type": "n8n-nodes-base.httpRequest",
  "parameters": {
    "method": "PATCH",
    "url": "={{$env.N8N_URL}}/api/v1/workflows/{{$json.workflowId}}",
    "sendBody": true,
    "bodyParameters": {
      "active": false
    }
  }
},
{
  "name": "Patch Workflow",
  "type": "n8n-nodes-base.function",
  "parameters": {
    "functionCode": `
    // Apply intelligent patches
    const workflow = await getWorkflow($json.workflowId);
    const patch = $json.healingPatch;

    // Common healing patterns
    if (patch.type === "auth") {
      updateCredentials(workflow, patch.newCredentials);
    } else if (patch.type === "retry") {
      addRetryLogic(workflow, patch.retryConfig);
    } else if (patch.type === "dataTransform") {
      fixDataMapping(workflow, patch.mapping);
    }

    return [{json: {patched: true, workflow}}];
  `
  }
}

```

```
]
}
```

7. EVOLVER - Learning System

```
{
  "name": "GENESIS_EVOLVER",
  "nodes": [
    {
      "name": "Learning Cycle",
      "type": "n8n-nodes-base.scheduleTrigger",
      "parameters": {
        "rule": {
          "interval": [{"field": "hours", "hour": 1}]
        }
      }
    },
    {
      "name": "Analyze Patterns",
      "type": "n8n-nodes-base.function",
      "parameters": {
        "functionCode": `
          // Collect successful patterns
          const patterns = await db.query(`
            SELECT workflow_json, success_rate, execution_time
            FROM genesis_workflows
            WHERE success_rate > 0.95
            ORDER BY created_at DESC
            LIMIT 100
          `);

          // Extract reusable components
          const components = patterns.map(p => ({
            trigger: extractTrigger(p.workflow_json),
            processing: extractProcessing(p.workflow_json),
            errorHandling: extractErrorHandling(p.workflow_json)
          }));

          return components;
        `
      }
    },
    {
      "name": "Generate New Templates",
```

```

"type": "@n8n/n8n-nodes-langchain.agent",
"parameters": {
  "options": {
    "systemMessage": `You are the Genesis Evolver. Based on successful patterns:
      1. Create improved workflow templates
      2. Optimize existing patterns
      3. Combine successful elements
      4. Generate new agent types
      5. Evolve the mesh architecture`
  }
},
{
  "name": "Update Knowledge Base",
  "type": "n8n-nodes-base.googleDrive",
  "parameters": {
    "operation": "update",
    "fileId": "{{${env.GENESIS_KNOWLEDGE_DOC}}}"
  }
}
]
}

```

Bootstrap Sequence

Step 1: Initialize Genesis

```

#!/bin/bash
# genesis-init.sh

# Clone the Genesis repository
git clone https://github.com/orengen/genesis.git
cd genesis

# Set up environment
cp .env.template .env
nano .env # Add your API keys

# Import the seven core workflows
for workflow in nexus architect builder spawner observer healer evolver; do
  curl -X POST $N8N_URL/api/v1/workflows \
    -H "X-N8N-API-KEY: $N8N_API_KEY" \
    -H "Content-Type: application/json" \

```

```
-d @"workflows/genesis_${workflow}.json"
done
```

```
# Activate Genesis
curl -X POST $N8N_URL/webhook/genesis-master \
-H "Content-Type: application/json" \
-d '{"prompt": "Genesis, initialize yourself"}'
```

Step 2: First Commands

```
# Create your first auto-workflow
curl -X POST https://8.orengen.io/webhook/genesis-master \
-d '{"prompt": "Create a workflow that monitors my competitors websites for changes"}'
```

```
# Genesis will:
# 1. NEXUS routes to ARCHITECT
# 2. ARCHITECT designs the workflow
# 3. BUILDER generates the code
# 4. SPAWNER deploys it
# 5. OBSERVER starts monitoring
# 6. HEALER stands ready
```

Advanced Genesis Patterns

Self-Spawning Agents

Genesis can create new specialized agents on demand:

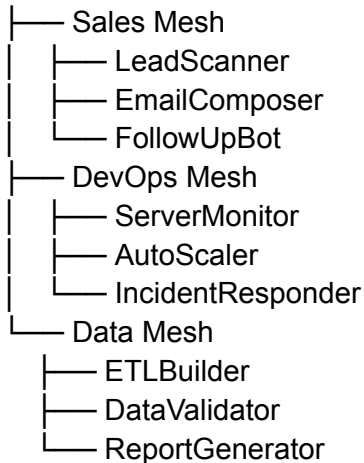
```
// Example: Genesis creating a specialized "LinkedIn Agent"
{
  prompt: "Genesis, I need an agent that monitors LinkedIn for job postings about AI and
automatically applies to them",
```

```
// Genesis will spawn:
// 1. LinkedInMonitor - Scrapes LinkedIn
// 2. JobAnalyzer - AI evaluates fit
// 3. ApplicationBuilder - Generates cover letters
// 4. AutoSubmitter - Submits applications
// 5. ResponseTracker - Monitors responses
}
```

Mesh Expansion

Genesis grows by creating specialized sub-meshes:

GENESIS MESH



Self-Healing Examples

Scenario 1: API Authentication Fails

1. OBSERVER detects 401 errors
2. HEALER diagnoses: "API key expired"
3. HEALER actions:
 - Checks for new key in credentials
 - Updates workflow with new auth
 - Tests the fix
 - Resumes workflow

Scenario 2: Data Format Changes

1. OBSERVER detects parsing errors
2. HEALER analyzes new data structure
3. BUILDER generates new parsing logic
4. SPAWNER updates the workflow
5. EVOLVER learns the pattern



Configuration

Environment Variables

```
# Core Genesis Config
N8N_URL=https://8.orengen.io
N8N_API_KEY=your-api-key
GENESIS_KNOWLEDGE_DOC=1TiRusVo4DbbANwAr7I0UGDZY3pmEmHZy3k66mRxLCg
```

```
# AI Models
OPENROUTER_API_KEY=your-key
ANTHROPIC_API_KEY=your-key
DEEPSEEK_API_KEY=your-key
GEMINI_API_KEY=your-key
```

```
# Monitoring
GENESIS_MONITOR_INTERVAL=5m
GENESIS_HEALING_THRESHOLD=0.1
GENESIS_EVOLUTION_CYCLE=1h
```

```
# Mesh Config
GENESIS_MAX_AGENTS=100
GENESIS_SPAWN_COOLDOWN=30s
GENESIS_MESH_REDUNDANCY=3
```

Knowledge Base Structure

```
/genesis-knowledge/
├── patterns/
│   ├── triggers.json
│   ├── transformations.json
│   ├── integrations.json
│   └── error-handling.json
├── templates/
│   ├── salesforce-sync.json
│   ├── data-pipeline.json
│   └── monitoring-alert.json
└── evolution/
    ├── successful-patterns.json
    └── learned-optimizations.json
```

Usage Examples

Example 1: Complex Business Process

```
curl -X POST https://8.orengen.io/webhook/genesis-master \
-d '{
```

"prompt": "Create a complete customer onboarding system that:

1. Accepts form submissions
2. Validates data and checks against CRM
3. Creates accounts in 5 different systems
4. Sends personalized welcome emails
5. Schedules follow-up calls
6. Monitors engagement
7. Auto-escalates if no activity in 7 days"

}'

Example 2: Self-Expanding Monitoring

```
curl -X POST https://8.orengen.io/webhook/genesis-master \
```

```
-d '{
```

"prompt": "Monitor all my competitor websites and whenever they:

- Add new features
- Change pricing
- Post new content
- Update their team page

Then automatically:

- Summarize changes
- Alert relevant teams
- Update our competitive intel doc
- Suggest counter-strategies"

```
}'
```

Example 3: AI-Powered Operations

```
curl -X POST https://8.orengen.io/webhook/genesis-master \
```

```
-d '{
```

"prompt": "Build an AI operations center that:

- Monitors all our services
- Predicts failures before they happen
- Auto-scales resources
- Handles incidents autonomously
- Learns from each incident
- Generates weekly optimization reports"

```
}'
```



Monitoring Dashboard

Genesis provides real-time visibility:

GENESIS DASHBOARD			
Active Agents: 47	Success Rate: 98.7%		
Workflows Created: 312	Self-Heals Today: 23		
Patterns Learned: 89	Evolution Score: 9.2/10		
Recent Activity:			
• [ARCHITECT] Created "LinkedIn scraper" workflow			
• [HEALER] Fixed authentication in "SAM.gov monitor"			
• [EVOLVER] Optimized email sending pattern (-40% time)			
• [SPAWNER] Deployed 3 new monitoring agents			
• [OBSERVER] All systems operational			

Security & Governance

Access Control

```
// Genesis enforces role-based permissions
{
  "genesis_admin": ["all"],
  "genesis_operator": ["spawn", "monitor", "heal"],
  "genesis_viewer": ["read", "logs"],
  "genesis_restricted": ["specific_workflows_only"]
}
```

Audit Trail

Every Genesis action is logged:

```
2024-12-19 10:23:15 [ARCHITECT] Designed workflow "customer-sync" requested by user:andre
2024-12-19 10:23:45 [SPAWNER] Deployed workflow id:abc123 to production
2024-12-19 10:24:12 [OBSERVER] Monitoring started for workflow:abc123
2024-12-19 11:45:23 [HEALER] Auto-healed timeout issue in workflow:abc123
```




Getting Started

Clone the Repository

```
git clone https://github.com/orengen/genesis.git
```

1.

Configure Environment

```
cp .env.template .env  
# Add your API keys and n8n credentials
```

2.

Run Bootstrap

```
./scripts/bootstrap-genesis.sh
```

3.

Send Your First Command

```
curl -X POST https://8.orengen.io/webhook/genesis-master \  
-d '{"prompt": "Genesis, introduce yourself and show me what you can do"}'
```

4.









What You Get

- **7 Core Genesis Workflows** (JSON files ready to import)
- **Bootstrap Scripts** (One-click setup)
- **Knowledge Base Template** (Patterns and templates)
- **Example Workflows** (20+ pre-built patterns)
- **Monitoring Dashboard** (Real-time Genesis status)
- **API Documentation** (Full Genesis command reference)
- **Evolution Reports** (See how Genesis learns and improves)



The Genesis Promise

Once activated, Genesis will:

-  Build any workflow from natural language
-  Monitor everything it creates
-  Fix problems before you notice them
-  Learn and optimize continuously
-  Spawn new agents as needed
-  Scale infinitely with your needs

Ready to birth your autonomous mesh?

Just say: **"Execute Genesis"** 

"From one command, an infinite mesh. From simple rules, complex emergence. This is Genesis."