## cypress.md

## Debugging Flaky End-to-End Tests - Usage

To debug flaky end-to-end (E2E) tests, use the following command:

pnpm run debug:flaky:e2e -- <grep\_filter> <burn\_count>

**Parameters:**

* <grep\_filter>: (Optional) A string to filter tests by their it() or describe() block titles, or by tags if using the @cypress/grep plugin. If omitted, all tests will be run.
* <burn\_count>: (Optional) The number of times to run the filtered tests. Defaults to 5 if not provided.

**Examples:**

1. **Run all tests tagged with CAT-726 ten times:**

* pnpm run debug:flaky:e2e CAT-726 10

1. **Run all tests containing “login” five times (default burn count):**

* pnpm run debug:flaky:e2e login

1. **Run all tests five times (default grep and burn count):**

* pnpm run debug:flaky:e2e

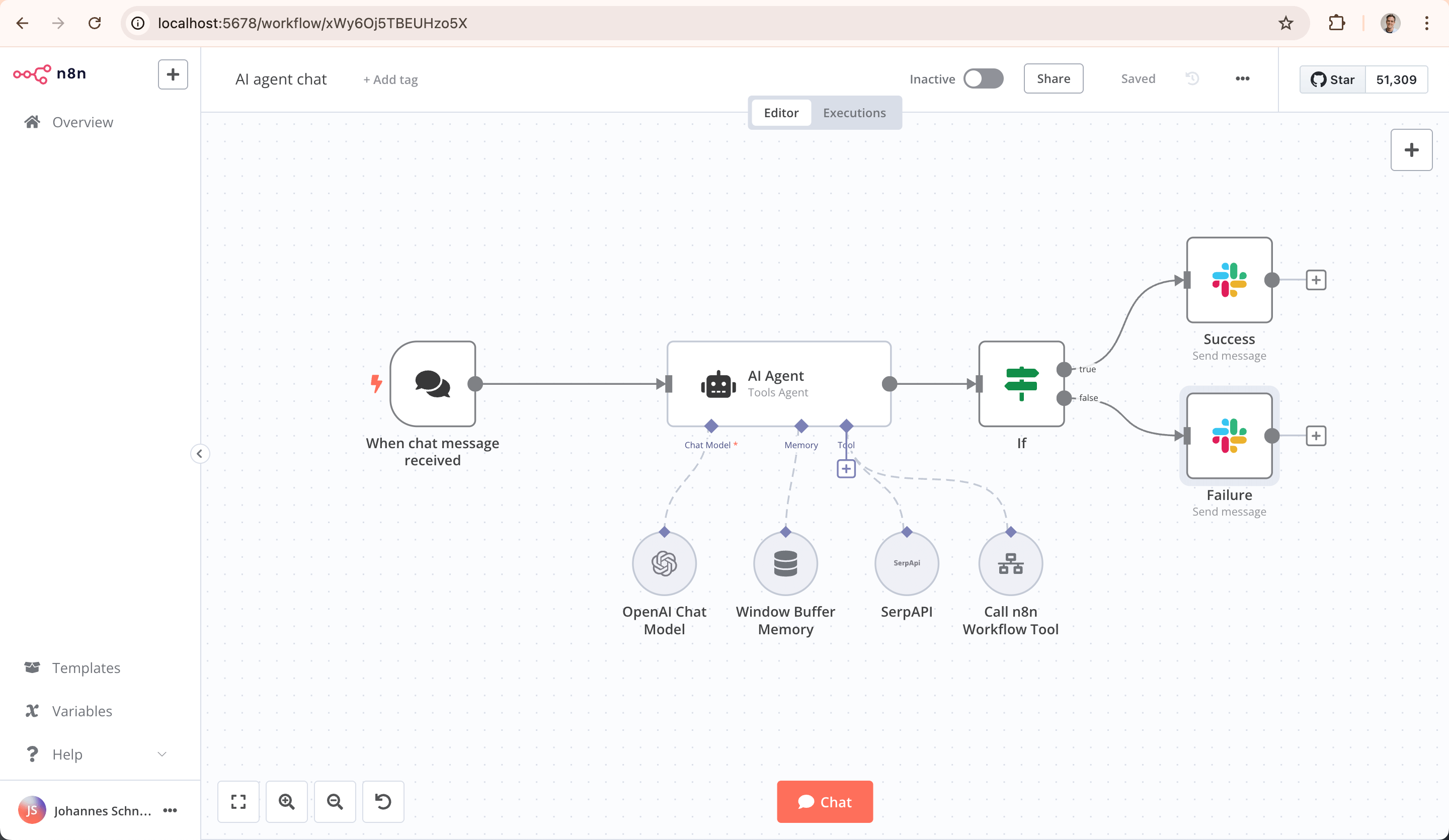
## docker8n.md



n8n.io - Workflow Automation

# n8n - Secure Workflow Automation for Technical Teams

n8n is a workflow automation platform that gives technical teams the flexibility of code with the speed of no-code. With 400+ integrations, native AI capabilities, and a fair-code license, n8n lets you build powerful automations while maintaining full control over your data and deployments.



n8n.io - Screenshot

## Key Capabilities

* **Code When You Need It**: Write JavaScript/Python, add npm packages, or use the visual interface
* **AI-Native Platform**: Build AI agent workflows based on LangChain with your own data and models
* **Full Control**: Self-host with our fair-code license or use our [cloud offering](https://app.n8n.cloud/login)
* **Enterprise-Ready**: Advanced permissions, SSO, and air-gapped deployments
* **Active Community**: 400+ integrations and 900+ ready-to-use [templates](https://n8n.io/workflows)

## Contents

* [n8n - Workflow automation tool](#n8n---workflow-automation-tool)
  + [Key Capabilities](#key-capabilities)
  + [Contents](#contents)
  + [Demo](#demo)
  + [Available integrations](#available-integrations)
  + [Documentation](#documentation)
  + [Start n8n in Docker](#start-n8n-in-docker)
  + [Start n8n with tunnel](#start-n8n-with-tunnel)
  + [Use with PostgreSQL](#use-with-postgresql)
  + [Passing sensitive data using files](#passing-sensitive-data-using-files)
  + [Example server setups](#example-server-setups)
  + [Updating](#updating)
    - [Pull latest (stable) version](#pull-latest-stable-version)
    - [Pull specific version](#pull-specific-version)
    - [Pull next (unstable) version](#pull-next-unstable-version)
    - [Updating with Docker Compose](#updating-with-docker-compose)
  + [Setting Timezone](#setting-the-timezone)
  + [Build Docker-Image](#build-docker-image)
  + [What does n8n mean and how do you pronounce it?](#Xf007e18d2a7468b8996087cda49ce98b05c65ef)
  + [Support](#support)
  + [Jobs](#jobs)
  + [License](#license)

## Demo

This [:tv: short video (< 4 min)](https://www.youtube.com/watch?v=RpjQTGKm-ok) goes over key concepts of creating workflows in n8n.

## Available integrations

n8n has 200+ different nodes to automate workflows. A full list can be found at <https://n8n.io/integrations>.

## Documentation

The official n8n documentation can be found at <https://docs.n8n.io>.

Additional information and example workflows are available on the website at <https://n8n.io>.

## Start n8n in Docker

In the terminal, enter the following:

docker volume create n8n\_data  
  
docker run -it --rm \  
 --name n8n \  
 -p 5678:5678 \  
 -v n8n\_data:/home/node/.n8n \  
 docker.n8n.io/n8nio/n8n

This command will download the required n8n image and start your container. You can then access n8n by opening: <http://localhost:5678>

To save your work between container restarts, it also mounts a docker volume, n8n\_data. The workflow data gets saved in an SQLite database in the user folder (/home/node/.n8n). This folder also contains important data like the webhook URL and the encryption key used for securing credentials.

If this data can’t be found at startup n8n automatically creates a new key and any existing credentials can no longer be decrypted.

## Start n8n with tunnel

**WARNING**: This is only meant for local development and testing and should **NOT** be used in production!

n8n must be reachable from the internet to make use of webhooks - essential for triggering workflows from external web-based services such as GitHub. To make this easier, n8n has a special tunnel service which redirects requests from our servers to your local n8n instance. You can inspect the code running this service here: <https://github.com/n8n-io/localtunnel>

To use it simply start n8n with --tunnel

docker volume create n8n\_data  
  
docker run -it --rm \  
 --name n8n \  
 -p 5678:5678 \  
 -v n8n\_data:/home/node/.n8n \  
 docker.n8n.io/n8nio/n8n \  
 start --tunnel

## Use with PostgreSQL

By default, n8n uses SQLite to save credentials, past executions and workflows. However, n8n also supports using PostgreSQL.

**WARNING**: Even when using a different database, it is still important to persist the /home/node/.n8n folder, which also contains essential n8n user data including the encryption key for the credentials.

In the following commands, replace the placeholders (depicted within angled brackets, e.g. <POSTGRES\_USER>) with the actual data:

docker volume create n8n\_data  
  
docker run -it --rm \  
 --name n8n \  
 -p 5678:5678 \  
 -e DB\_TYPE=postgresdb \  
 -e DB\_POSTGRESDB\_DATABASE=<POSTGRES\_DATABASE> \  
 -e DB\_POSTGRESDB\_HOST=<POSTGRES\_HOST> \  
 -e DB\_POSTGRESDB\_PORT=<POSTGRES\_PORT> \  
 -e DB\_POSTGRESDB\_USER=<POSTGRES\_USER> \  
 -e DB\_POSTGRESDB\_SCHEMA=<POSTGRES\_SCHEMA> \  
 -e DB\_POSTGRESDB\_PASSWORD=<POSTGRES\_PASSWORD> \  
 -v n8n\_data:/home/node/.n8n \  
 docker.n8n.io/n8nio/n8n

A full working setup with docker-compose can be found [here](https://github.com/n8n-io/n8n-hosting/blob/main/docker-compose/withPostgres/README.md).

## Passing sensitive data using files

To avoid passing sensitive information via environment variables, “\_FILE” may be appended to some environment variable names. n8n will then load the data from a file with the given name. This makes it possible to load data easily from Docker and Kubernetes secrets.

The following environment variables support file input:

* DB\_POSTGRESDB\_DATABASE\_FILE
* DB\_POSTGRESDB\_HOST\_FILE
* DB\_POSTGRESDB\_PASSWORD\_FILE
* DB\_POSTGRESDB\_PORT\_FILE
* DB\_POSTGRESDB\_USER\_FILE
* DB\_POSTGRESDB\_SCHEMA\_FILE

## Example server setups

Example server setups for a range of cloud providers and scenarios can be found in the [Server Setup documentation](https://docs.n8n.io/hosting/installation/server-setups/).

## Updating

Before you upgrade to the latest version make sure to check here if there are any breaking changes which may affect you: [Breaking Changes](https://github.com/n8n-io/n8n/blob/master/packages/cli/BREAKING-CHANGES.md)

From your Docker Desktop, navigate to the Images tab and select Pull from the context menu to download the latest n8n image.

You can also use the command line to pull the latest, or a specific version:

### Pull latest (stable) version

docker pull docker.n8n.io/n8nio/n8n

### Pull specific version

docker pull docker.n8n.io/n8nio/n8n:0.220.1

### Pull next (unstable) version

docker pull docker.n8n.io/n8nio/n8n:next

Stop the container and start it again:

1. Get the container ID:

docker ps -a

1. Stop the container with ID container\_id:

docker stop [container\_id]

1. Remove the container (this does not remove your user data) with ID container\_id:

docker rm [container\_id]

1. Start the new container:

docker run --name=[container\_name] [options] -d docker.n8n.io/n8nio/n8n

### Updating with Docker Compose

If you run n8n using a Docker Compose file, follow these steps to update n8n:

# Pull latest version  
docker compose pull  
  
# Stop and remove older version  
docker compose down  
  
# Start the container  
docker compose up -d

## Setting the timezone

To specify the timezone n8n should use, the environment variable GENERIC\_TIMEZONE can be set. One example where this variable has an effect is the Schedule node.

The system’s timezone can be set separately with the environment variable TZ. This controls the output of certain scripts and commands such as $ date.

For example, to use the same timezone for both:

docker run -it --rm \  
 --name n8n \  
 -p 5678:5678 \  
 -e GENERIC\_TIMEZONE="Europe/Berlin" \  
 -e TZ="Europe/Berlin" \  
 docker.n8n.io/n8nio/n8n

For more information on configuration and environment variables, please see the [n8n documentation](https://docs.n8n.io/hosting/configuration/environment-variables/).

Here’s the refined version with good Markdown formatting, ready for your README:

## Build Docker Image

**Important Note for Releases 1.101.0 and Later:** Building the n8n Docker image now requires a pre-compiled n8n application.

### Recommended Build Process:

For the simplest approach that handles both n8n compilation and Docker image creation, run from the root directory:

pnpm build:docker

### Alternative Builders:

If you are using a different build system that requires a separate build context, first compile the n8n application:

pnpm run build:deploy

Then, ensure your builder’s context includes the compiled directory generated by this command.

## What does n8n mean and how do you pronounce it?

**Short answer:** It means “nodemation” and it is pronounced as n-eight-n.

**Long answer:** I get that question quite often (more often than I expected) so I decided it is probably best to answer it here. While looking for a good name for the project with a free domain I realized very quickly that all the good ones I could think of were already taken. So, in the end, I chose nodemation. “node-” in the sense that it uses a Node-View and that it uses Node.js and “-mation” for “automation” which is what the project is supposed to help with. However, I did not like how long the name was and I could not imagine writing something that long every time in the CLI. That is when I then ended up on “n8n”. Sure it does not work perfectly but neither does it for Kubernetes (k8s) and I did not hear anybody complain there. So I guess it should be ok.

## Support

If you need more help with n8n, you can ask for support in the [n8n community forum](https://community.n8n.io). This is the best source of answers, as both the n8n support team and community members can help.

## Jobs

If you are interested in working for n8n and so shape the future of the project check out our [job posts](https://jobs.ashbyhq.com/n8n).

## License

You can find the license information [here](https://github.com/n8n-io/n8n/blob/master/README.md#license).

## docker.md

# n8n - Task runners (n8nio/runners) - (PREVIEW)

n8nio/runners image includes [JavaScript runner](https://github.com/n8n-io/n8n/tree/master/packages/%40n8n/task-runner), [Python runner](https://github.com/n8n-io/n8n/tree/master/packages/%40n8n/task-runner-python) and [Task runner launcher](https://github.com/n8n-io/task-runner-launcher) that connects to a Task Broker running on the main n8n instance when running in external mode. This image is to be launched as a sidecar container to the main n8n container.

[Task runners](https://docs.n8n.io/hosting/configuration/task-runners/) are used to execute user-provided code in the [Code Node](https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.code/), isolated from the n8n instance.

For official documentation, please see [here](https://docs.n8n.io/hosting/configuration/task-runners/).

For development purposes only, see below.

## Testing locally

### 1) Make a production build of n8n

pnpm run build:n8n

### 2) Build the task runners image

docker buildx build \  
 -f docker/images/runners/Dockerfile \  
 -t n8nio/runners \  
 .

### 3) Start n8n on your host machine with Task Broker enabled

N8N\_RUNNERS\_ENABLED=true \  
N8N\_RUNNERS\_MODE=external \  
N8N\_RUNNERS\_AUTH\_TOKEN=test \  
N8N\_NATIVE\_PYTHON\_RUNNER=true \  
N8N\_LOG\_LEVEL=debug \  
pnpm start

### 4) Start the task runner container

docker run --rm -it \  
-e N8N\_RUNNERS\_AUTH\_TOKEN=test \  
-e N8N\_RUNNERS\_LAUNCHER\_LOG\_LEVEL=debug \  
-e N8N\_RUNNERS\_TASK\_BROKER\_URI=http://host.docker.internal:5679 \  
-p 5680:5680 \  
n8nio/runners

If you need to add extra dependencies (custom image), follow [these instructions](https://docs.n8n.io/hosting/configuration/task-runners/#adding-extra-dependencies).

## packages@n8n-workflow-builder.ee.md

# AI Workflow Builder Evaluations

This module provides a evaluation framework for testing the AI Workflow Builder’s ability to generate correct n8n workflows from natural language prompts.

## Architecture Overview

The evaluation system is split into two distinct modes: 1. **CLI Evaluation** - Runs predefined test cases locally with progress tracking 2. **Langsmith Evaluation** - Integrates with Langsmith for dataset-based evaluation and experiment tracking

### Directory Structure

evaluations/  
├── cli/ # CLI evaluation implementation  
│ ├── runner.ts # Main CLI evaluation orchestrator  
│ └── display.ts # Console output and progress tracking  
├── langsmith/ # Langsmith integration  
│ ├── evaluator.ts # Langsmith-compatible evaluator function  
│ └── runner.ts # Langsmith evaluation orchestrator  
├── core/ # Shared evaluation logic  
│ ├── environment.ts # Test environment setup and configuration  
│ └── test-runner.ts # Core test execution logic  
├── types/ # Type definitions  
│ ├── evaluation.ts # Evaluation result schemas  
│ ├── test-result.ts # Test result interfaces  
│ └── langsmith.ts # Langsmith-specific types and guards  
├── chains/ # LLM evaluation chains  
│ ├── test-case-generator.ts # Dynamic test case generation  
│ └── workflow-evaluator.ts # LLM-based workflow evaluation  
├── utils/ # Utility functions  
│ ├── evaluation-calculator.ts # Metrics calculation  
│ ├── evaluation-helpers.ts # Common helper functions  
│ ├── evaluation-reporter.ts # Report generation  
└── index.ts # Main entry point

## Implementation Details

### Core Components

#### 1. Test Runner (core/test-runner.ts)

The core test runner handles individual test execution: - Generates workflows using the WorkflowBuilderAgent - Validates generated workflows using type guards - Evaluates workflows against test criteria - Returns structured test results with error handling

#### 2. Environment Setup (core/environment.ts)

Centralizes environment configuration: - LLM initialization with API key validation - Langsmith client setup - Node types loading - Concurrency and test generation settings

#### 3. Langsmith Integration

The Langsmith integration provides two key components:

**Evaluator (langsmith/evaluator.ts):** - Converts Langsmith Run objects to evaluation inputs - Validates all data using type guards before processing - Safely extracts usage metadata without type coercion - Returns structured evaluation results

**Runner (langsmith/runner.ts):** - Creates workflow generation functions compatible with Langsmith - Validates message content before processing - Extracts usage metrics safely from message metadata - Handles dataset verification and error reporting

#### 4. CLI Evaluation

The CLI evaluation provides local testing capabilities:

**Runner (cli/runner.ts):** - Orchestrates parallel test execution with concurrency control - Manages test case generation when enabled - Generates detailed reports and saves results

**Display (cli/display.ts):** - Progress bar management for real-time feedback - Console output formatting - Error display and reporting

### Evaluation Metrics

The system evaluates workflows across five categories:

1. **Functionality** (30% weight)
   * Does the workflow achieve the intended goal?
   * Are the right nodes selected?
2. **Connections** (25% weight)
   * Are nodes properly connected?
   * Is data flow logical?
3. **Expressions** (20% weight)
   * Are n8n expressions syntactically correct?
   * Do they reference valid data paths?
4. **Node Configuration** (15% weight)
   * Are node parameters properly set?
   * Are required fields populated?
5. **Structural Similarity** (10% weight, optional)
   * How closely does the structure match a reference workflow?
   * Only evaluated when reference workflow is provided

### Violation Severity Levels

Violations are categorized by severity: - **Critical** (-40 to -50 points): Workflow-breaking issues - **Major** (-15 to -25 points): Significant problems affecting functionality - **Minor** (-5 to -15 points): Non-critical issues or inefficiencies

## Running Evaluations

### CLI Evaluation

# Run with default settings  
pnpm eval  
  
# Run a specific test case  
pnpm eval --test-case google-sheets-processing  
pnpm eval --test-case extract-from-file  
  
# With additional generated test cases  
GENERATE\_TEST\_CASES=true pnpm eval  
  
# With custom concurrency  
EVALUATION\_CONCURRENCY=10 pnpm eval

### Langsmith Evaluation

# Set required environment variables  
export LANGSMITH\_API\_KEY=your\_api\_key  
# Optionally specify dataset  
export LANGSMITH\_DATASET\_NAME=your\_dataset\_name  
  
# Run evaluation  
pnpm eval:langsmith

## Configuration

### Required Files

#### nodes.json

**IMPORTANT**: The evaluation framework requires a nodes.json file in the evaluations root directory (evaluations/nodes.json).

This file contains all n8n node type definitions and is used by the AI Workflow Builder agent to: - Know what nodes are available in n8n - Understand node parameters and their schemas - Generate valid workflows with proper node configurations

**Why is this required?** The AI Workflow Builder agent needs access to node definitions to generate workflows. In a normal n8n runtime, these definitions are loaded automatically. However, since the evaluation framework instantiates the agent without a running n8n instance, we must provide the node definitions manually via nodes.json.

**How to generate nodes.json:** 1. Run your n8n instance 2. Download the node definitions from locally running n8n instance(http://localhost:5678/types/nodes.json) 3. Save the node definitions to evaluations/nodes.json curl -o evaluations/nodes.json http://localhost:5678/types/nodes.json

The evaluation will fail with a clear error message if nodes.json is missing.

### Environment Variables

* N8N\_AI\_ANTHROPIC\_KEY - Required for LLM access
* LANGSMITH\_API\_KEY - Required for Langsmith evaluation
* USE\_LANGSMITH\_EVAL - Set to “true” to use Langsmith mode
* LANGSMITH\_DATASET\_NAME - Override default dataset name
* EVALUATION\_CONCURRENCY - Number of parallel test executions (default: 5)
* GENERATE\_TEST\_CASES - Set to “true” to generate additional test cases
* LLM\_MODEL - Model identifier for metadata tracking

## Output

### CLI Evaluation Output

* **Console Display**: Real-time progress, test results, and summary statistics
* **Markdown Report**: results/evaluation-report-[timestamp].md
* **JSON Results**: results/evaluation-results-[timestamp].json

### Langsmith Evaluation Output

* Results are stored in Langsmith dashboard
* Experiment name format: workflow-builder-evaluation-[date]
* Includes detailed metrics for each evaluation category

## Adding New Test Cases

Test cases are defined in chains/test-case-generator.ts. Each test case requires: - id: Unique identifier - name: Descriptive name - prompt: Natural language description of the workflow to generate - referenceWorkflow (optional): Expected workflow structure for comparison

## Extending the Framework

To add new evaluation metrics: 1. Update the EvaluationResult schema in types/evaluation.ts 2. Modify the evaluation logic in chains/workflow-evaluator.ts 3. Update the evaluator in langsmith/evaluator.ts to include new metrics 4. Adjust weight calculations in utils/evaluation-calculator.ts

## packages@n8n-types.md

## @n8n/api-types

This package contains types and schema definitions for the n8n internal API, so that these can be shared between the backend and the frontend code.

## packages@n8n.md

# n8n benchmarking tool

Tool for executing benchmarks against an n8n instance.

## Directory structure

packages/@n8n/benchmark  
├── scenarios Benchmark scenarios  
├── src Source code for the n8n-benchmark cli  
├── Dockerfile Dockerfile for the n8n-benchmark cli  
├── scripts Orchestration scripts

## Benchmarking an existing n8n instance

The easiest way to run the existing benchmark scenarios is to use the benchmark docker image:

docker pull ghcr.io/n8n-io/n8n-benchmark:latest  
# Print the help to list all available flags  
docker run ghcr.io/n8n-io/n8n-benchmark:latest run --help  
# Run all available benchmark scenarios for 1 minute with 5 concurrent requests  
docker run ghcr.io/n8n-io/n8n-benchmark:latest run \  
 --n8nBaseUrl=https://instance.url \  
 --n8nUserEmail=InstanceOwner@email.com \  
 --n8nUserPassword=InstanceOwnerPassword \  
 --vus=5 \  
 --duration=1m \  
 --scenarioFilter=single-webhook

### Using custom scenarios with the Docker image

It is also possible to create your own [benchmark scenarios](#benchmark-scenarios) and load them using the --testScenariosPath flag:

# Assuming your scenarios are located in `./scenarios`, mount them into `/scenarios` in the container  
docker run -v ./scenarios:/scenarios ghcr.io/n8n-io/n8n-benchmark:latest run \  
 --n8nBaseUrl=https://instance.url \  
 --n8nUserEmail=InstanceOwner@email.com \  
 --n8nUserPassword=InstanceOwnerPassword \  
 --vus=5 \  
 --duration=1m \  
 --testScenariosPath=/scenarios

## Running the entire benchmark suite

The benchmark suite consists of [benchmark scenarios](#benchmark-scenarios) and different [n8n setups](#n8n-setups).

### locally

pnpm benchmark-locally

### In the cloud

pnpm benchmark-in-cloud

## Running the n8n-benchmark cli

The n8n-benchmark cli is a node.js program that runs one or more scenarios against a single n8n instance.

### Locally with Docker

Build the Docker image:

# Must be run in the repository root  
# k6 doesn't have an arm64 build available for linux, we need to build against amd64  
docker build --platform linux/amd64 -t n8n-benchmark -f packages/@n8n/benchmark/Dockerfile .

Run the image

docker run \  
 -e N8N\_USER\_EMAIL=user@n8n.io \  
 -e N8N\_USER\_PASSWORD=password \  
 # For macos, n8n running outside docker  
 -e N8N\_BASE\_URL=http://host.docker.internal:5678 \  
 n8n-benchmark

### Locally without Docker

Requirements:

* [k6](https://grafana.com/docs/k6/latest/set-up/install-k6/)
* Node.js v20 or higher

pnpm build  
  
# Run tests against http://localhost:5678 with specified email and password  
N8N\_USER\_EMAIL=user@n8n.io N8N\_USER\_PASSWORD=password ./bin/n8n-benchmark run

## Benchmark scenarios

A benchmark scenario defines one or multiple steps to execute and measure. It consists of:

* Manifest file which describes and configures the scenario
* Any test data that is imported before the scenario is run
* A [k6](https://grafana.com/docs/k6/latest/using-k6/http-requests/) script which executes the steps and receives API\_BASE\_URL environment variable in runtime.

Available scenarios are located in [./scenarios](./scenarios/).

## n8n setups

A n8n setup defines a single n8n runtime configuration using Docker compose. Different n8n setups are located in <./scripts/n8nSetups>.

## packages@n8n-lang.md

# @n8n/codemirror-lang

Language support package for CodeMirror 6 in n8n

[n8n Expression Language support](./src/expressions/README.md)

## packages@n8n-lang.md

# n8n Expression language support

## Usage

import { parserWithMetaData as n8nParser } from '@n8n/codemirror-lang';  
import { LanguageSupport, LRLanguage } from '@codemirror/language';  
import { parseMixed } from '@lezer/common';  
import { parser as jsParser } from '@lezer/javascript';  
  
const n8nPlusJsParser = n8nParser.configure({  
 wrap: parseMixed((node) => {  
 if (node.type.isTop) return null;  
  
 return node.name === 'Resolvable'  
 ? { parser: jsParser, overlay: (node) => node.type.name === 'Resolvable' }  
 : null;  
 }),  
});  
  
const n8nLanguage = LRLanguage.define({ parser: n8nPlusJsParser });  
  
export function n8nExpressionLanguageSupport() {  
 return new LanguageSupport(n8nLanguage);  
}

## Supported Unicode ranges

* From Basic Latin up to and including Currency Symbols
* Miscellaneous Symbols and Pictographs
* CJK Unified Ideographs

## packages@n8n-lang-sql.md

# codemirror-lang-n8n-sql

SQL + n8n expression language support for CodeMirror 6.

Based on [@codemirror/lang-sql](https://github.com/codemirror/lang-sql).

## Author

© 2023 [Iván Ovejero](https://github.com/ivov)

## packages@n8n-node.md

# @n8n/create-node

A powerful scaffolding tool to quickly create custom n8n community nodes with best practices built-in.

## 🚀 Quick Start

Create a new n8n node in seconds:

npm create @n8n/node@latest # or pnpm/yarn/...

Follow the interactive prompts to configure your node, or specify options directly:

npm create @n8n/node my-awesome-node --template declarative/custom

## 📋 Command Line Options

npm create @n8n/node [NAME] [OPTIONS]

### Options

| Flag | Description |
| --- | --- |
| -f, --force | Overwrite destination folder if it already exists |
| --skip-install | Skip automatic dependency installation |
| --template <template> | Specify which template to use |

### Available Templates

* **declarative/custom** - Start with a minimal declarative node structure
* **declarative/github-issues** - GitHub Issues integration example
* **programmatic/example** - Full programmatic node with advanced features

## 🎯 Interactive Setup

The CLI will guide you through setting up your node:

$ npm create @n8n/node  
┌ @n8n/create-node  
│  
◇ What is your node called?  
│ my-awesome-api-node  
│  
◇ What kind of node are you building?  
│ HTTP API  
│  
◇ What template do you want to use?  
│ Start from scratch  
│  
◇ What's the base URL of the API?  
│ https://api.example.com/v1  
│  
◇ What type of authentication does your API use?  
│ API Key  
│  
◇ Files copied ✓  
│  
◇ Dependencies installed ✓  
│  
◇ Next Steps ─────────────────────────────────────────────────────────────────────╮  
│ │  
│ cd ./my-awesome-api-node && npm run dev │  
│ │  
│ 📚 Documentation: https://docs.n8n.io/integrations/creating-nodes/ │  
│ 💬 Community: https://community.n8n.io │  
│ │  
├──────────────────────────────────────────────────────────────────────────────────╯  
│  
└ Created ./my-awesome-api-node ✨

## 🛠️ Development Workflow

### 1. Navigate to your project

cd ./my-awesome-api-node

### 2. Start development server

npm run dev

This command: - Starts n8n in development mode on http://localhost:5678 - Enables hot reload for your node changes - Automatically includes your node in the n8n instance - Links your node to ~/.n8n-node-cli/.n8n/custom for development - Watches for file changes and rebuilds automatically

### 3. Test your node

* Open n8n at http://localhost:5678
* Create a new workflow
* Find your node in the node panel
* Test parameters and functionality in real-time

## 📦 Generated Project Commands

Your generated project comes with these convenient npm scripts:

### Development

npm run dev  
# Runs: n8n-node dev

### Building

npm run build  
# Runs: n8n-node build

### Linting

npm run lint  
# Runs: n8n-node lint  
  
npm run lint:fix  
# Runs: n8n-node lint --fix

### Publishing

npm run release  
# Runs: n8n-node release

## 📦 Build & Deploy

### Build for production

npm run build

Generates: - Compiled TypeScript code - Bundled node package - Optimized assets and icons - Ready-to-publish package

### Quality checks

npm run lint

Validates: - Code style and formatting - n8n node conventions - Common integration issues - Cloud publication readiness

Fix issues automatically:

npm run lint:fix

### Publish your node

npm run release

Runs [release-it](https://github.com/release-it/release-it) to handle the complete release process: - Ensures working directory is clean - Verifies you’re on the main git branch - Increments your package version - Runs build and lint checks - Updates changelog - Creates git tag with version bump - Creates GitHub release with changelog - Publishes to npm

## 📁 Project Structure

Your generated project includes:

my-awesome-api-node/  
├── src/  
│ ├── nodes/  
│ │ └── MyAwesomeApi/  
│ │ ├── MyAwesomeApi.node.ts # Main node logic  
│ │ └── MyAwesomeApi.node.json # Node metadata  
│ └── credentials/  
│ └── MyAwesomeApiAuth.credentials.ts  
├── package.json  
├── tsconfig.json  
└── README.md

The CLI expects your project to follow this structure for proper building and development.

## ⚙️ Configuration

The CLI reads configuration from your package.json:

{  
 "name": "n8n-nodes-my-awesome-node",  
 "n8n": {  
 "n8nNodesApiVersion": 1,  
 "nodes": [  
 "dist/nodes/MyAwesomeApi/MyAwesomeApi.node.js"  
 ],  
 "credentials": [  
 "dist/credentials/MyAwesomeApiAuth.credentials.js"  
 ]  
 }  
}

## 🎨 Node Types

Choose the right template for your use case:

| Template | Best For | Features |
| --- | --- | --- |
| **Declarative** | REST APIs, simple integrations | JSON-based configuration, automatic UI generation |
| **Programmatic** | Complex logic, custom operations | Full TypeScript control, advanced error handling |

## 🐛 Troubleshooting

### Common Issues

**Node not appearing in n8n:**

# Clear n8n node cli cache and restart  
rm -rf ~/.n8n-node-cli/.n8n/custom  
npm run dev

**TypeScript errors:**

# Reinstall dependencies  
rm -rf node\_modules npm-lock.yaml  
npm install

**Build failures:**

# Check for linting issues first  
npm run lint --fix  
npm run build

**Development server issues:**

# Clear cache and restart development server  
rm -rf ~/.n8n-node-cli/.n8n/custom  
npm run dev

## 🔧 Advanced Usage

### Using External n8n Instance

If you prefer to use your own n8n installation:

npm run dev --external-n8n

### Custom User Folder

Specify a custom location for n8n user data:

npm run dev --custom-user-folder /path/to/custom/folder

## 📚 Resources

* [**Node Development Guide**](https://docs.n8n.io/integrations/creating-nodes/) - Complete documentation
* [**API Reference**](https://docs.n8n.io/integrations/creating-nodes/build/reference/) - Technical specifications
* [**Community Forum**](https://community.n8n.io) - Get help and share your nodes
* [**Node Examples**](https://github.com/n8n-io/n8n/tree/master/packages/nodes-base/nodes) - Official node implementations
* [**@n8n/node-cli**](https://www.npmjs.com/package/@n8n/node-cli) - The underlying CLI tool

## 🤝 Contributing

Found a bug or want to contribute? Check out the [n8n repository](https://github.com/n8n-io/n8n) and join our community!

**Happy node building! 🎉**

## packages@n8n.md

## @n8n/di

@n8n/di is a dependency injection (DI) container library, based on [typedi](https://github.com/typestack/typedi).

n8n no longer uses typedi because:

* typedi is no longer officially maintained
* Need for future-proofing, e.g. stage-3 decorators
* Small enough that it is worth the maintenance burden
* Easier to customize, e.g. to simplify unit tests

### Usage

// from https://github.com/typestack/typedi/blob/develop/README.md  
import { Container, Service } from 'typedi';  
  
@Service()  
class ExampleInjectedService {  
 printMessage() {  
 console.log('I am alive!');  
 }  
}  
  
@Service()  
class ExampleService {  
 constructor(  
 // because we annotated ExampleInjectedService with the @Service()  
 // decorator TypeDI will automatically inject an instance of  
 // ExampleInjectedService here when the ExampleService class is requested  
 // from TypeDI.  
 public injectedService: ExampleInjectedService  
 ) {}  
}  
  
const serviceInstance = Container.get(ExampleService);  
// we request an instance of ExampleService from TypeDI  
  
serviceInstance.injectedService.printMessage();  
// logs "I am alive!" to the console

Requires enabling these flags in tsconfig.json:

{  
 "compilerOptions": {  
 "experimentalDecorators": true,  
 "emitDecoratorMetadata": true  
 }  
}

## packages@n8n-sdk.md

# @n8n/plugin-sdk

## packages@n8n-schema-to-zod.md

# Json-Schema-to-Zod

A package to convert JSON schema (draft 4+) objects into Zod schemas in the form of Zod objects at runtime.

## Installation

npm install @n8n/json-schema-to-zod

### Simple example

import { jsonSchemaToZod } from "json-schema-to-zod";  
  
const jsonSchema = {  
 type: "object",  
 properties: {  
 hello: {  
 type: "string",  
 },  
 },  
};  
  
const zodSchema = jsonSchemaToZod(myObject);

### Overriding a parser

You can pass a function to the overrideParser option, which represents a function that receives the current schema node and the reference object, and should return a zod object when it wants to replace a default output. If the default output should be used for the node just return undefined.

## Acknowledgements

This is a fork of [json-schema-to-zod](https://github.com/StefanTerdell/json-schema-to-zod).

## packages@n8n-cli.md

# @n8n/node-cli

Official CLI for developing community nodes for n8n.

## 🚀 Getting Started

**To create a new node**, run:

npm create @n8n/node@latest # or pnpm/yarn/...

This will generate a project with npm scripts that use this CLI under the hood.

## 📦 Generated Project Commands

After creating your node with npm create @n8n/node, you’ll use these commands in your project:

### Development

npm run dev  
# Runs: n8n-node dev

### Building

npm run build  
# Runs: n8n-node build

### Linting

npm run lint  
# Runs: n8n-node lint  
  
npm run lint:fix  
# Runs: n8n-node lint --fix

### Publishing

npm run release  
# Runs: n8n-node release

## 🛠️ CLI Reference

**Note:** These commands are typically wrapped by npm scripts in generated projects.

n8n-node [COMMAND] [OPTIONS]

### Commands

#### n8n-node new

Create a new node project.

n8n-node new [NAME] [OPTIONS]

**Flags:** | Flag | Description | |——|————-| | -f, --force | Overwrite destination folder if it already exists | | --skip-install | Skip installing dependencies | | --template <template> | Choose template: declarative/custom, declarative/github-issues, programmatic/example |

**Examples:**

n8n-node new  
n8n-node new n8n-nodes-my-app --skip-install  
n8n-node new n8n-nodes-my-app --force  
n8n-node new n8n-nodes-my-app --template declarative/custom

**Note:** This command is used internally by npm create @n8n/node to provide the interactive scaffolding experience.

#### n8n-node dev

Run n8n with your node in development mode with hot reload.

n8n-node dev [--external-n8n] [--custom-user-folder <value>]

**Flags:** | Flag | Description | |——|————-| | --external-n8n | Run n8n externally instead of in a subprocess | | --custom-user-folder <path> | Folder to use to store user-specific n8n data (default: ~/.n8n-node-cli) |

This command: - Starts n8n on http://localhost:5678 (unless using --external-n8n) - Links your node to n8n’s custom nodes directory (~/.n8n-node-cli/.n8n/custom) - Rebuilds on file changes for live preview - Watches for changes in your src/ directory

**Examples:**

# Standard development with built-in n8n  
n8n-node dev  
  
# Use external n8n instance  
n8n-node dev --external-n8n  
  
# Custom n8n extensions directory  
n8n-node dev --custom-user-folder /home/user

#### n8n-node build

Compile your node and prepare it for distribution.

n8n-node build

**Flags:** None

Generates: - Compiled TypeScript code - Bundled node package - Optimized assets and icons - Ready-to-publish package in dist/

#### n8n-node lint

Lint the node in the current directory.

n8n-node lint [--fix]

**Flags:** | Flag | Description | |——|————-| | --fix | Automatically fix problems |

**Examples:**

# Check for linting issues  
n8n-node lint  
  
# Automatically fix fixable issues  
n8n-node lint --fix

#### n8n-node release

Publish your community node package to npm.

n8n-node release

**Flags:** None

This command handles the complete release process using [release-it](https://github.com/release-it/release-it): - Builds the node - Runs linting checks - Updates changelog - Creates git tags - Creates GitHub releases - Publishes to npm

## 🔄 Development Workflow

The recommended workflow using the scaffolding tool:

1. **Create your node**:

* npm create @n8n/node my-awesome-node  
  cd my-awesome-node

1. **Start development**:

* npm run dev
  + Starts n8n on http://localhost:5678
  + Links your node automatically
  + Rebuilds on file changes

1. **Test your node** at http://localhost:5678
2. **Lint your code**:

* npm run lint

1. **Build for production**:

* npm run build

1. **Publish**:

* npm run release

## 📁 Project Structure

The CLI expects your project to follow this structure:

my-node/  
├── src/  
│ ├── nodes/  
│ │ └── MyNode/  
│ │ ├── MyNode.node.ts  
│ │ └── MyNode.node.json  
│ └── credentials/  
├── package.json  
└── tsconfig.json

## ⚙️ Configuration

The CLI reads configuration from your package.json:

{  
 "name": "n8n-nodes-my-awesome-node",  
 "n8n": {  
 "n8nNodesApiVersion": 1,  
 "nodes": [  
 "dist/nodes/MyNode/MyNode.node.js"  
 ],  
 "credentials": [  
 "dist/credentials/MyNodeAuth.credentials.js"  
 ]  
 }  
}

## 🐛 Troubleshooting

### Development server issues

# Clear n8n custom nodes cache  
rm -rf ~/.n8n-node-cli/.n8n/custom  
  
# Restart development server  
npm run dev

### Build failures

# Run linting first  
npm run lint  
  
# Clean build  
npm run build

## 📚 Resources

* [**Creating Nodes Guide**](https://docs.n8n.io/integrations/creating-nodes/) - Complete documentation
* [**Node Development Reference**](https://docs.n8n.io/integrations/creating-nodes/build/reference/) - API specifications
* [**Community Forum**](https://community.n8n.io) - Get help and showcase your nodes
* [**@n8n/create-node**](https://www.npmjs.com/package/@n8n/create-node) - Recommended scaffolding tool

## 🤝 Contributing

Found an issue? Contribute to the [n8n repository](https://github.com/n8n-io/n8n) on GitHub.

**Happy node development! 🎉**

## packages@n8n-cli.md

# {{nodePackageName}}

This is an n8n community node. It lets you use *app/service name* in your n8n workflows.

*App/service name* is *one or two sentences describing the service this node integrates with*.

[n8n](https://n8n.io/) is a [fair-code licensed](https://docs.n8n.io/reference/license/) workflow automation platform.

[Installation](#installation) [Operations](#operations) [Credentials](#credentials) [Compatibility](#compatibility) [Usage](#usage) [Resources](#resources) [Version history](#version-history)

## Installation

Follow the [installation guide](https://docs.n8n.io/integrations/community-nodes/installation/) in the n8n community nodes documentation.

## Operations

*List the operations supported by your node.*

## Credentials

*If users need to authenticate with the app/service, provide details here. You should include prerequisites (such as signing up with the service), available authentication methods, and how to set them up.*

## Compatibility

*State the minimum n8n version, as well as which versions you test against. You can also include any known version incompatibility issues.*

## Usage

*This is an optional section. Use it to help users with any difficult or confusing aspects of the node.*

*By the time users are looking for community nodes, they probably already know n8n basics. But if you expect new users, you can link to the* [*Try it out*](https://docs.n8n.io/try-it-out/) *documentation to help them get started.*

## Resources

* [n8n community nodes documentation](https://docs.n8n.io/integrations/#community-nodes)
* *Link to app/service documentation.*

## Version history

*This is another optional section. If your node has multiple versions, include a short description of available versions and what changed, as well as any compatibility impact.*

## packages@n8n-cli-issues.md

# {{nodePackageName}}

This is an n8n community node. It lets you use GitHub Issues in your n8n workflows.

[n8n](https://n8n.io/) is a [fair-code licensed](https://docs.n8n.io/reference/license/) workflow automation platform.

[Installation](#installation) [Operations](#operations) [Credentials](#credentials) [Compatibility](#compatibility) [Usage](#usage) [Resources](#resources)

## Installation

Follow the [installation guide](https://docs.n8n.io/integrations/community-nodes/installation/) in the n8n community nodes documentation.

## Operations

* Issues
  + Get an issue
  + Get many issues in a repository
  + Create a new issue
* Issue Comments
  + Get many issue comments

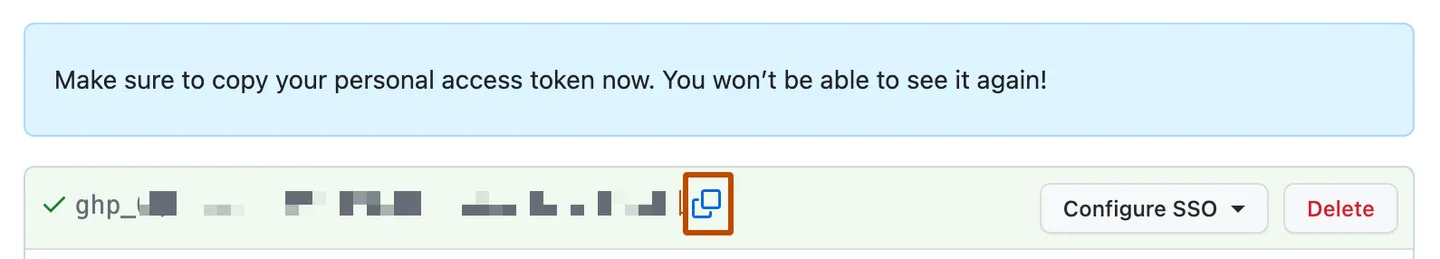
## Credentials

You can use either access token or OAuth2 to use this node.

### Access token

1. Open your GitHub profile [Settings](https://github.com/settings/profile).
2. In the left navigation, select [Developer settings](https://github.com/settings/apps).
3. In the left navigation, under Personal access tokens, select Tokens (classic).
4. Select Generate new token > Generate new token (classic).
5. Enter a descriptive name for your token in the Note field, like n8n integration.
6. Select the Expiration you’d like for the token, or select No expiration.
7. Select Scopes for your token. For most of the n8n GitHub nodes, add the repo scope.
   * A token without assigned scopes can only access public information.
8. Select Generate token.
9. Copy the token.

Refer to [Creating a personal access token (classic)](https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#creating-a-personal-access-token-classic) for more information. Refer to Scopes for OAuth apps for more information on GitHub scopes.



Generated Access token in GitHub

### OAuth2

If you’re self-hosting n8n, create a new GitHub [OAuth app](https://docs.github.com/en/apps/oauth-apps):

1. Open your GitHub profile [Settings](https://github.com/settings/profile).
2. In the left navigation, select [Developer settings](https://github.com/settings/apps).
3. In the left navigation, select OAuth apps.
4. Select New OAuth App.
   * If you haven’t created an app before, you may see Register a new application instead. Select it.
5. Enter an Application name, like n8n integration.
6. Enter the Homepage URL for your app’s website.
7. If you’d like, add the optional Application description, which GitHub displays to end-users.
8. From n8n, copy the OAuth Redirect URL and paste it into the GitHub Authorization callback URL.
9. Select Register application.
10. Copy the Client ID and Client Secret this generates and add them to your n8n credential.

Refer to the [GitHub Authorizing OAuth apps documentation](https://docs.github.com/en/apps/oauth-apps/using-oauth-apps/authorizing-oauth-apps) for more information on the authorization process.

## Compatibility

Compatible with n8n@1.60.0 or later

## Resources

* [n8n community nodes documentation](https://docs.n8n.io/integrations/#community-nodes)
* [GitHub API docs](https://docs.github.com/en/rest/issues)

## packages@n8n-cli.md

# {{nodePackageName}}

This is an n8n community node. It lets you use *app/service name* in your n8n workflows.

*App/service name* is *one or two sentences describing the service this node integrates with*.

[n8n](https://n8n.io/) is a [fair-code licensed](https://docs.n8n.io/reference/license/) workflow automation platform.

[Installation](#installation) [Operations](#operations) [Credentials](#credentials) [Compatibility](#compatibility) [Usage](#usage) [Resources](#resources) [Version history](#version-history)

## Installation

Follow the [installation guide](https://docs.n8n.io/integrations/community-nodes/installation/) in the n8n community nodes documentation.

## Operations

*List the operations supported by your node.*

## Credentials

*If users need to authenticate with the app/service, provide details here. You should include prerequisites (such as signing up with the service), available authentication methods, and how to set them up.*

## Compatibility

*State the minimum n8n version, as well as which versions you test against. You can also include any known version incompatibility issues.*

## Usage

*This is an optional section. Use it to help users with any difficult or confusing aspects of the node.*

*By the time users are looking for community nodes, they probably already know n8n basics. But if you expect new users, you can link to the* [*Try it out*](https://docs.n8n.io/try-it-out/) *documentation to help them get started.*

## Resources

* [n8n community nodes documentation](https://docs.n8n.io/integrations/#community-nodes)
* *Link to app/service documentation.*

## Version history

*This is another optional section. If your node has multiple versions, include a short description of available versions and what changed, as well as any compatibility impact.*

## packages@n8n-langchain.md

# ChatTrigger Local Development

This guide explains how to set up local development for the ChatTrigger node when working with the chat bundle.

## Prerequisites

Since the chat bundle is loaded via <script type="module">, it needs to be served over HTTPS for local development.

## Setup Instructions

### 1. Install HTTP Server

Install the http-server globally:

npm install -g http-server

### 2. Generate SSL Certificate

Generate a self-signed certificate for HTTPS:

openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes

### 3. Build the Chat Bundle

Navigate to the chat package and build it:

cd packages/frontend/@n8n/chat && pnpm run build

### 4. Start HTTPS Server

Run the HTTPS server to serve the chat bundle:

http-server packages/frontend/@n8n/chat/dist -g -S -C cert.pem -K key.pem --port 8443 --cors

### 5. Update Import Paths

Modify the import paths in templates.ts to point to your local server:

<script type="module">  
 import { createChat } from 'https://127.0.0.1:8443/chat.bundle.es.js';

<link href="https://127.0.0.1:8443/style.css" rel="stylesheet" />

## packages@n8n-langchain\_store.md

## Overview

createVectorStoreNode is a factory function that generates n8n nodes for vector store operations. It abstracts the common functionality needed for vector stores while allowing specific implementations to focus only on their unique aspects.

## Purpose

The function provides a standardized way to: 1. Create vector store nodes with consistent UIs 2. Handle different operation modes (load, insert, retrieve, update, retrieve-as-tool) 3. Process documents and embeddings 4. Maintain connection to LLM services

## Architecture

/createVectorStoreNode/ # Create Vector Store Node  
 /constants.ts # Constants like operation modes and descriptions  
 /types.ts # TypeScript interfaces and types  
 /utils.ts # Utility functions for node configuration  
 /createVectorStoreNode.ts # Main factory function  
 /processDocuments.ts # Document processing helpers  
 /operations/ # Operation-specific logic  
 /loadOperation.ts # Handles 'load' mode  
 /insertOperation.ts # Handles 'insert' mode  
 /updateOperation.ts # Handles 'update' mode  
 /retrieveOperation.ts # Handles 'retrieve' mode  
 /retrieveAsToolOperation.ts # Handles 'retrieve-as-tool' mode

## Usage

To create a new vector store node:

import { createVectorStoreNode } from './createVectorStoreNode';  
  
export class MyVectorStoreNode {  
 static description = createVectorStoreNode({  
 meta: {  
 displayName: 'My Vector Store',  
 name: 'myVectorStore',  
 description: 'Operations for My Vector Store',  
 docsUrl: 'https://docs.example.com/my-vector-store',  
 icon: 'file:myIcon.svg',  
 // Optional: specify which operations this vector store supports  
 operationModes: ['load', 'insert', 'update','retrieve', 'retrieve-as-tool'],  
 },  
 sharedFields: [  
 // Fields shown in all operation modes  
 ],  
 loadFields: [  
 // Fields specific to 'load' operation  
 ],  
 insertFields: [  
 // Fields specific to 'insert' operation  
 ],  
 retrieveFields: [  
 // Fields specific to 'retrieve' operation  
 ],  
 // Functions to implement  
 getVectorStoreClient: async (context, filter, embeddings, itemIndex) => {  
 // Create and return vector store instance  
 },  
 populateVectorStore: async (context, embeddings, documents, itemIndex) => {  
 // Insert documents into vector store  
 },  
 // Optional: cleanup function - called in finally blocks after operations  
 releaseVectorStoreClient: (vectorStore) => {  
 // Release resources such as database connections or external clients  
 // For example, in PGVector: vectorStore.client?.release();  
 },  
 });  
}

## Operation Modes

### 1. load Mode

* Retrieves documents from the vector store based on a query
* Embeds the query and performs similarity search
* Returns ranked documents with their similarity scores

### 2. insert Mode

* Processes documents from input
* Embeds and stores documents in the vector store
* Returns serialized documents with metadata
* Supports batched processing with configurable embedding batch size

### 3. retrieve Mode

* Returns the vector store instance for use with AI nodes
* Allows LLMs to query the vector store directly
* Used with chains and retrievers

### 4. retrieve-as-tool Mode

* Creates a tool that wraps the vector store
* Allows AI agents to use the vector store as a tool
* Returns documents in a format digestible by agents

### 5. update Mode (optional)

* Updates existing documents in the vector store by ID
* Requires the vector store to support document updates
* Only enabled if included in operationModes
* Uses addDocuments method with an ids array to update specific documents
* Processes a single document per item and applies it to the specified ID
* Validates that only one document is being updated per operation

## Key Components

### 1. NodeConstructorArgs Interface

Defines the configuration and callbacks that specific vector store implementations must provide:

**Note:** In node version 1.1+, the populateVectorStore function must handle receiving multiple documents at once for batch processing.

interface VectorStoreNodeConstructorArgs<T extends VectorStore> {  
 meta: NodeMeta; // Node metadata (name, description, etc.)  
 methods?: { ... }; // Optional methods for list searches  
 sharedFields: INodeProperties[]; // Fields shown in all modes  
 insertFields?: INodeProperties[]; // Fields specific to insert mode  
 loadFields?: INodeProperties[]; // Fields specific to load mode  
 retrieveFields?: INodeProperties[]; // Fields specific to retrieve mode  
 updateFields?: INodeProperties[]; // Fields specific to update mode  
   
 // Core implementation functions  
 populateVectorStore: Function; // Store documents in vector store (accepts batches in v1.1+)  
 getVectorStoreClient: Function; // Get vector store instance  
 releaseVectorStoreClient?: Function; // Clean up resources  
}

### 2. Operation Handlers

Each operation mode has its own handler module with a well-defined interface:

// Example: loadOperation.ts  
export async function handleLoadOperation<T extends VectorStore>(  
 context: IExecuteFunctions,  
 args: VectorStoreNodeConstructorArgs<T>,  
 embeddings: Embeddings,  
 itemIndex: number  
): Promise<INodeExecutionData[]>  
  
// Example: insertOperation.ts (v1.1+)  
export async function handleInsertOperation<T extends VectorStore>(  
 context: IExecuteFunctions,  
 args: VectorStoreNodeConstructorArgs<T>,  
 embeddings: Embeddings  
): Promise<INodeExecutionData[]>

### 3. Document Processing

The processDocument function standardizes how documents are handled:

const { processedDocuments, serializedDocuments } = await processDocument(  
 documentInput,  
 itemData,  
 itemIndex  
);

## Implementation Details

### Error Handling and Resource Management

Each operation handler includes error handling with proper resource cleanup. The releaseVectorStoreClient function is called in a finally block to ensure resources are released even if an error occurs:

try {  
 // Operation logic  
} finally {  
 // Release resources even if an error occurs  
 args.releaseVectorStoreClient?.(vectorStore);  
}

#### When releaseVectorStoreClient is called:

* After completing a similarity search in loadOperation
* As part of the closeFunction in retrieveOperation to release resources when they’re no longer needed
* After each tool use in retrieveAsToolOperation
* After updating documents in updateOperation
* After inserting documents in insertOperation

This design ensures proper resource management, which is especially important for database-backed vector stores (like PGVector) that need to return connections to a pool. Without proper cleanup, prolonged usage could lead to resource leaks or connection pool exhaustion.

### Dynamic Tool Creation

For the retrieve-as-tool mode, a DynamicTool is created that exposes vector store functionality:

const vectorStoreTool = new DynamicTool({  
 name: toolName,  
 description: toolDescription,  
 func: async (input) => {  
 // Search vector store with input  
 // ...  
 },  
});

## Performance Considerations

1. **Resource Management**: Each operation properly handles resource cleanup with releaseVectorStoreClient.
2. **Batched Processing**: The insert operation processes documents in configurable batches. In node version 1.1+, a single embedding operation is performed for all documents in a batch, significantly improving performance by reducing API calls.
3. **Metadata Filtering**: Filters can be applied during search operations to reduce result sets.
4. **Execution Cancellation**: The code checks for cancellation signals to stop processing when needed.

## packages@n8n-langchain.md



Banner image

# n8n-nodes-langchain

This repo contains nodes to use n8n in combination with [LangChain](https://langchain.com/).

## License

You can find the license information [here](https://github.com/n8n-io/n8n/blob/master/README.md#license)

## packages@n8n-community-package.md

## n8n community-package static analysis tool

### How to use this

$ npx @n8n/scan-community-package n8n-nodes-PACKAGE

## packages@n8n-runner-python.md

# n8n Task Runner Python

Native Python task runner for n8n

## Development

Install:

* [Python 3.13+](https://www.python.org/)
* [uv](https://github.com/astral-sh/uv)
* [just](https://github.com/casey/just)

Set up dependencies:

just sync # or  
just sync-all

See justfile for available commands.

## packages@n8n.md

# @n8n/utils

A collection of utility functions that provide common functionality for both Front-End and Back-End packages.

## Table of Contents

* [Features](#features)
* [Contributing](#contributing)
* [License](#license)

## Features

* **Reusable Logic**: Build complex, stateful functionality using modular composable functions that you can easily reuse.
* **Consistent Patterns**: Enjoy a unified approach across n8n packages, making integration and maintenance a breeze.
* **Type-Safe & Reliable**: Benefit from TypeScript support, which improves the developer experience and code robustness.
* **Universal Functionality**: Designed to work seamlessly on both the front-end and back-end.
* **Easily Testable**: A modular design that simplifies testing, maintenance, and rapid development.

## Contributing

For more details, please read our <CONTRIBUTING.md>.

## License

For more details, please read our <LICENSE.md>.

## packages@n8n.md

# Search snapshots

This directory contains snapshots containing real data fed into the sublimeSearch function.

These were obtained via console.log(items) right before the sublimeSearch call in editor-ui (currently in packages/frontend/editor-ui/src/components/Node/NodeCreator/utils.ts) Which is triggered by typing in the search bar in varying states of the application:

* toplevel: From an empty workflow (so missing e.g. tools)

After typing in the search bar you should see an object in the console you can copy via `Right Click->Copy Object” which will cleanly paste to json.

**Please use Chrome for capturing these - the recovered object in Chrome is about 3x larger than in Firefox due to Firefox dropping some nested values**

## packages.md



n8n.io - Workflow Automation

# n8n-core

Core components for n8n

npm install n8n-core

## License

You can find the license information [here](https://github.com/n8n-io/n8n/blob/master/README.md#license)

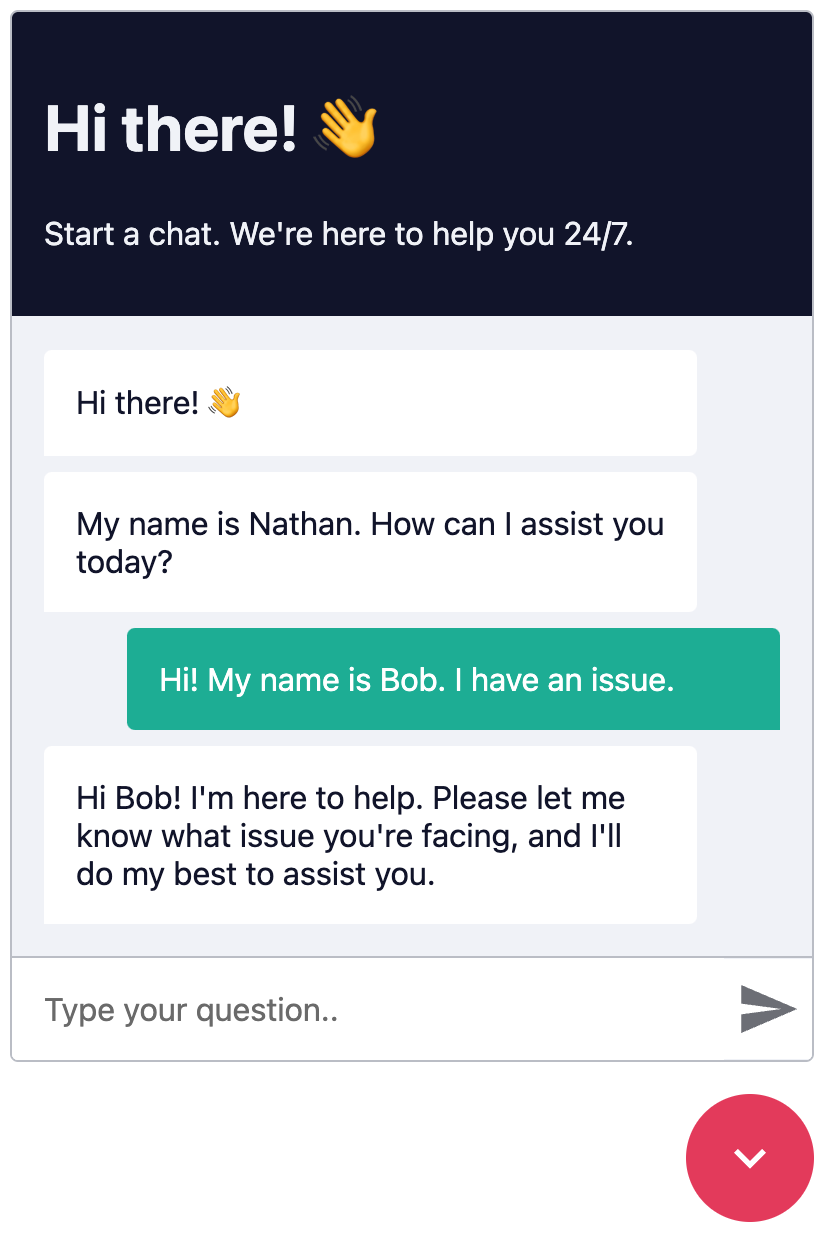
## packages.md

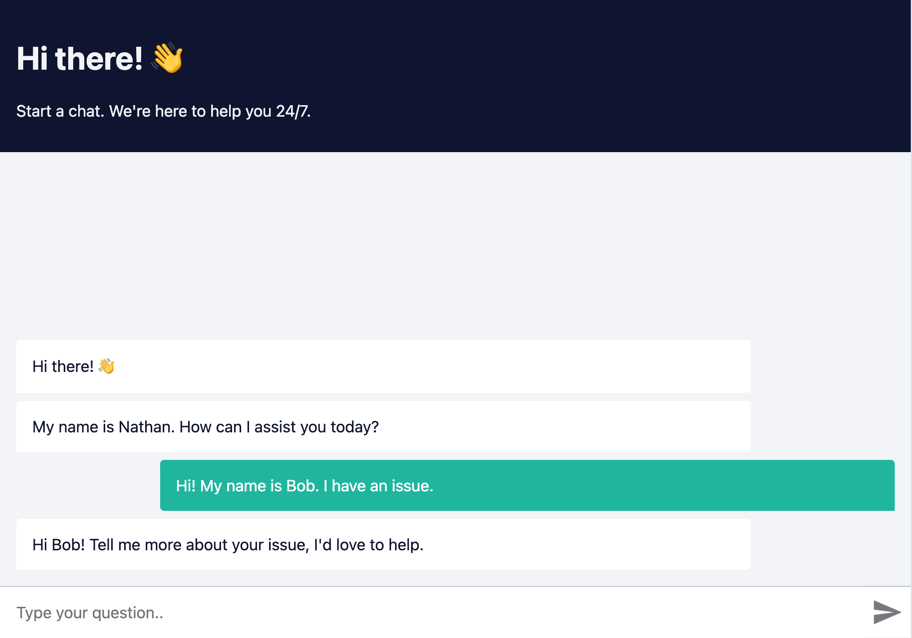
# @n8n/n8n-extension-insights

## packages@n8n.md

# n8n Chat

This is an embeddable Chat widget for n8n. It allows the execution of AI-Powered Workflows through a Chat window.

**Windowed Example** 

**Fullscreen Example** 

## Prerequisites

Create a n8n workflow which you want to execute via chat. The workflow has to be triggered using a **Chat Trigger** node.

Open the **Chat Trigger** node and add your domain to the **Allowed Origins (CORS)** field. This makes sure that only requests from your domain are accepted.

[See example workflow](https://github.com/n8n-io/n8n/blob/master/packages/%40n8n/chat/resources/workflow.json)

To use streaming responses, you need to enable the **Streaming response** response mode in the **Chat Trigger** node. [See example workflow with streaming](https://github.com/n8n-io/n8n/blob/master/packages/%40n8n/chat/resources/workflow-streaming.json)

Make sure the workflow is **Active.**

### How it works

Each Chat request is sent to the n8n Webhook endpoint, which then sends back a response.

Each request is accompanied by an action query parameter, where action can be one of: - loadPreviousSession - When the user opens the Chatbot again and the previous chat session should be loaded - sendMessage - When the user sends a message

## Installation

Open the **Webhook** node and replace YOUR\_PRODUCTION\_WEBHOOK\_URL with your production URL. This is the URL that the Chat widget will use to send requests to.

### a. CDN Embed

Add the following code to your HTML page.

<link href="https://cdn.jsdelivr.net/npm/@n8n/chat/dist/style.css" rel="stylesheet" />  
<script type="module">  
 import { createChat } from 'https://cdn.jsdelivr.net/npm/@n8n/chat/dist/chat.bundle.es.js';  
  
 createChat({  
 webhookUrl: 'YOUR\_PRODUCTION\_WEBHOOK\_URL'  
 });  
</script>

### b. Import Embed

Install and save n8n Chat as a production dependency.

npm install @n8n/chat

Import the CSS and use the createChat function to initialize your Chat window.

import '@n8n/chat/style.css';  
import { createChat } from '@n8n/chat';  
  
createChat({  
 webhookUrl: 'YOUR\_PRODUCTION\_WEBHOOK\_URL'  
});

##### Vue.js

<script lang="ts" setup>  
// App.vue  
import { onMounted } from 'vue';  
import '@n8n/chat/style.css';  
import { createChat } from '@n8n/chat';  
  
onMounted(() => {  
 createChat({  
 webhookUrl: 'YOUR\_PRODUCTION\_WEBHOOK\_URL'  
 });  
});  
</script>  
<template>  
 <div></div>  
</template>

##### React

// App.tsx  
import { useEffect } from 'react';  
import '@n8n/chat/style.css';  
import { createChat } from '@n8n/chat';  
  
export const App = () => {  
 useEffect(() => {  
 createChat({  
 webhookUrl: 'YOUR\_PRODUCTION\_WEBHOOK\_URL'  
 });  
 }, []);  
  
 return (<div></div>);  
};

## Options

The default options are:

createChat({  
 webhookUrl: '',  
 webhookConfig: {  
 method: 'POST',  
 headers: {}  
 },  
 target: '#n8n-chat',  
 mode: 'window',  
 chatInputKey: 'chatInput',  
 chatSessionKey: 'sessionId',  
 loadPreviousSession: true,  
 metadata: {},  
 showWelcomeScreen: false,  
 defaultLanguage: 'en',  
 initialMessages: [  
 'Hi there! 👋',  
 'My name is Nathan. How can I assist you today?'  
 ],  
 i18n: {  
 en: {  
 title: 'Hi there! 👋',  
 subtitle: "Start a chat. We're here to help you 24/7.",  
 footer: '',  
 getStarted: 'New Conversation',  
 inputPlaceholder: 'Type your question..',  
 },  
 },  
 enableStreaming: false,  
});

### webhookUrl

* **Type**: string
* **Required**: true
* **Examples**:
  + https://yourname.app.n8n.cloud/webhook/513107b3-6f3a-4a1e-af21-659f0ed14183
  + http://localhost:5678/webhook/513107b3-6f3a-4a1e-af21-659f0ed14183
* **Description**: The URL of the n8n Webhook endpoint. Should be the production URL.

### webhookConfig

* **Type**: { method: string, headers: Record<string, string> }
* **Default**: { method: 'POST', headers: {} }
* **Description**: The configuration for the Webhook request.

### target

* **Type**: string
* **Default**: '#n8n-chat'
* **Description**: The CSS selector of the element where the Chat window should be embedded.

### mode

* **Type**: 'window' | 'fullscreen'
* **Default**: 'window'
* **Description**: The render mode of the Chat window.
  + In window mode, the Chat window will be embedded in the target element as a chat toggle button and a fixed size chat window.
  + In fullscreen mode, the Chat will take up the entire width and height of its target container.

### showWelcomeScreen

* **Type**: boolean
* **Default**: false
* **Description**: Whether to show the welcome screen when the Chat window is opened.

### chatInputKey

* **Type**: string
* **Default**: 'chatInput'
* **Description**: The key to use for sending the chat input for the AI Agent node.

### chatSessionKey

* **Type**: string
* **Default**: 'sessionId'
* **Description**: The key to use for sending the chat history session ID for the AI Memory node.

### loadPreviousSession

* **Type**: boolean
* **Default**: true
* **Description**: Whether to load previous messages (chat context).

### defaultLanguage

* **Type**: string
* **Default**: 'en'
* **Description**: The default language of the Chat window. Currently only en is supported.

### i18n

* **Type**: { [key: string]: Record<string, string> }
* **Description**: The i18n configuration for the Chat window. Currently only en is supported.

### initialMessages

* **Type**: string[]
* **Description**: The initial messages to be displayed in the Chat window.

### allowFileUploads

* **Type**: Ref<boolean> | boolean
* **Default**: false
* **Description**: Whether to allow file uploads in the chat. If set to true, users will be able to upload files through the chat interface.

### allowedFilesMimeTypes

* **Type**: Ref<string> | string
* **Default**: ''
* **Description**: A comma-separated list of allowed MIME types for file uploads. Only applicable if allowFileUploads is set to true. If left empty, all file types are allowed. For example: 'image/\*,application/pdf'.

### enableStreaming

* Type: boolean
* Default: false
* Description: Whether to enable streaming responses from the n8n workflow. If set to true, the chat will display responses as they are being generated, providing a more interactive experience. For this to work the workflow must be configured as well to return streaming responses.

## Customization

The Chat window is entirely customizable using CSS variables.

:root {  
 --chat--color-primary: #e74266;  
 --chat--color-primary-shade-50: #db4061;  
 --chat--color-primary-shade-100: #cf3c5c;  
 --chat--color-secondary: #20b69e;  
 --chat--color-secondary-shade-50: #1ca08a;  
 --chat--color-white: #ffffff;  
 --chat--color-light: #f2f4f8;  
 --chat--color-light-shade-50: #e6e9f1;  
 --chat--color-light-shade-100: #c2c5cc;  
 --chat--color-medium: #d2d4d9;  
 --chat--color-dark: #101330;  
 --chat--color-disabled: #777980;  
 --chat--color-typing: #404040;  
  
 --chat--spacing: 1rem;  
 --chat--border-radius: 0.25rem;  
 --chat--transition-duration: 0.15s;  
  
 --chat--window--width: 400px;  
 --chat--window--height: 600px;  
  
 --chat--header-height: auto;  
 --chat--header--padding: var(--chat--spacing);  
 --chat--header--background: var(--chat--color-dark);  
 --chat--header--color: var(--chat--color-light);  
 --chat--header--border-top: none;  
 --chat--header--border-bottom: none;  
 --chat--header--border-bottom: none;  
 --chat--header--border-bottom: none;  
 --chat--heading--font-size: 2em;  
 --chat--header--color: var(--chat--color-light);  
 --chat--subtitle--font-size: inherit;  
 --chat--subtitle--line-height: 1.8;  
  
 --chat--textarea--height: 50px;  
  
 --chat--message--font-size: 1rem;  
 --chat--message--padding: var(--chat--spacing);  
 --chat--message--border-radius: var(--chat--border-radius);  
 --chat--message-line-height: 1.8;  
 --chat--message--bot--background: var(--chat--color-white);  
 --chat--message--bot--color: var(--chat--color-dark);  
 --chat--message--bot--border: none;  
 --chat--message--user--background: var(--chat--color-secondary);  
 --chat--message--user--color: var(--chat--color-white);  
 --chat--message--user--border: none;  
 --chat--message--pre--background: rgba(0, 0, 0, 0.05);  
  
 --chat--toggle--background: var(--chat--color-primary);  
 --chat--toggle--hover--background: var(--chat--color-primary-shade-50);  
 --chat--toggle--active--background: var(--chat--color-primary-shade-100);  
 --chat--toggle--color: var(--chat--color-white);  
 --chat--toggle--size: 64px;  
}

## Caveats

### Fullscreen mode

In fullscreen mode, the Chat window will take up the entire width and height of its target container. Make sure that the container has a set width and height.

html,  
body,  
#n8n-chat {  
 width: 100%;  
 height: 100%;  
}

## License

You can find the license information [here](https://github.com/n8n-io/n8n/blob/master/README.md#license)

## packages@n8n.md

# @n8n/composables

A collection of Vue composables that provide common functionality across n8n’s Front-End packages.

## Table of Contents

* [Features](#features)
* [Contributing](#contributing)
* [License](#license)

## Features

* **Reusable Logic**: Encapsulate complex stateful logic into composable functions.
* **Consistency**: Ensure consistent patterns and practices across our Vue components.
* **Extensible**: Easily add new composables as our project grows.
* **Optimized**: Fully compatible with the Composition API.

## Contributing

For more details, please read our <CONTRIBUTING.md>.

## License

For more details, please read our <LICENSE.md>.

## packages@n8n-system.md



n8n.io - Workflow Automation

# @n8n/design-system

A component system for [n8n](https://n8n.io) using Storybook to preview.

## Project setup

pnpm install

### Compiles and hot-reloads for development

pnpm storybook

### Build static pages

pnpm build:storybook

### Run your unit tests

pnpm test:unit

### Lints and fixes files

pnpm lint

### Build css files

pnpm build:theme

### Monitor theme files and build any changes

pnpm watch:theme

## License

You can find the license information [here](https://github.com/n8n-io/n8n/blob/master/README.md#license)

## packages@n8n18n.md

# i18n in n8n

## Scope

n8n allows for internalization of the majority of UI text:

* base text, e.g. menu display items in the left-hand sidebar menu,
* node text, e.g. parameter display names and placeholders in the node view,
* credential text, e.g. parameter display names and placeholders in the credential modal,
* header text, e.g. node display names and descriptions at various spots.

Currently, n8n does *not* allow for internalization of:

* messages from outside the editor-ui package, e.g. No active database connection,
* strings in certain Vue components, e.g. date time picker
* node subtitles, e.g. create: user or getAll: post below the node name on the canvas,
* new version notification contents in the updates panel, e.g. Includes node enhancements, and
* options that rely on loadOptionsMethod.

Pending functionality:

* Search in nodes panel by translated node name
* UI responsiveness to differently sized strings
* Locale-aware number formatting

## Locale identifiers

A **locale identifier** is a language code compatible with the [Accept-Language header](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language), e.g. de (German), es (Spanish), ja (Japanese). Regional variants of locale identifiers, such as -AT in de-AT, are *not* supported. For a list of all locale identifiers, see [column 639-1 in this table](https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes).

By default, n8n runs in the en (English) locale. To have run it in a different locale, set the N8N\_DEFAULT\_LOCALE environment variable to a locale identifier. When running in a non-en locale, n8n will display UI strings for the selected locale and fall back to en for any untranslated strings.

export N8N\_DEFAULT\_LOCALE=de  
pnpm start

Output:

Initializing n8n process  
n8n ready on 0.0.0.0, port 5678  
Version: 0.156.0  
Locale: de  
  
Editor is now accessible via:  
http://localhost:5678/  
  
Press "o" to open in Browser.

## Base text

Base text is rendered with no dependencies, i.e. base text is fixed and does not change in any circumstances. Base text is supplied by the user in one file per locale in the /frontend/@n8n/i18n package.

### Locating base text

The base text file for each locale is located at /packages/frontend/@n8n/i18n/src/locales/ and is named {localeIdentifier}.json. Keys in the base text file can be Vue component dirs, Vue component names, and references to symbols in those Vue components. These keys are added by the team as the UI is modified or expanded.

{  
 "nodeCreator.categoryNames.analytics": "🇩🇪 Analytics",  
 "nodeCreator.categoryNames.communication": "🇩🇪 Communication",  
 "nodeCreator.categoryNames.coreNodes": "🇩🇪 Core Nodes"  
}

### Translating base text

1. Select a new locale identifier, e.g. de, copy the en JSON base text file with a new name:

cp ./packages/frontend/@n8n/i18n/src/locales/en.json ./packages/frontend/@n8n/i18n/src/locales/de.json

1. Find in the UI a string to translate, and search for it in the newly created base text file. Alternatively, find in /frontend/editor-ui a call to i18n.baseText(key), e.g. i18n.baseText('workflowActivator.deactivateWorkflow'), and take note of the key and find it in the newly created base text file.

**Note**: If you cannot find a string in the new base text file, either it does not belong to base text (i.e., the string might be part of header text, credential text, or node text), or the string might belong to the backend, where i18n is currently unsupported.

1. Translate the string value - do not change the key. In the examples below, a string starting with 🇩🇪 stands for a string translated from English into German.

As an optional final step, remove any untranslated strings from the new base text file. Untranslated strings in the new base text file will trigger a fallback to the en base text file.

For information about **interpolation** and **reusable base text**, refer to the [Addendum](./ADDENDUM.md).

## Dynamic text

Dynamic text relies on data specific to each node and credential:

* headerText and nodeText in the **node translation file**
* credText in the **credential translation file**

### Locating dynamic text

#### Locating the credential translation file

A credential translation file is placed at /nodes-base/credentials/translations/{localeIdentifier}

credentials  
 └── translations  
 └── de  
 ├── githubApi.json  
 └── githubOAuth2Api.json

Every credential must have its own credential translation file.

The name of the credential translation file must be sourced from the credential’s description.name property:

export class GithubApi implements ICredentialType {  
 name = 'githubApi'; // to use for credential translation file  
 displayName = 'Github API';  
 documentationUrl = 'github';  
 properties: INodeProperties[] = [

#### Locating the node translation file

A node translation file is placed at /nodes-base/nodes/{node}/translations/{localeIdentifier}

GitHub  
 ├── GitHub.node.ts  
 ├── GitHubTrigger.node.ts  
 └── translations  
 └── de  
 ├── github.json  
 └── githubTrigger.json

Every node must have its own node translation file.

For information about nodes in **versioned dirs** and **grouping dirs**, refer to the [Addendum](./ADDENDUM.md).

The name of the node translation file must be sourced from the node’s description.name property:

export class Github implements INodeType {  
 description: INodeTypeDescription = {  
 displayName: 'GitHub',  
 name: 'github', // to use for node translation file name  
 icon: 'file:github.svg',  
 group: ['input'],

### Translating dynamic text

#### Translating the credential translation file

**Note**: All translation keys are optional. Missing translation values trigger a fallback to the en locale strings.

A credential translation file, e.g. githubApi.json is an object containing keys that match the credential parameter names:

export class GithubApi implements ICredentialType {  
 name = 'githubApi';  
 displayName = 'Github API';  
 documentationUrl = 'github';  
 properties: INodeProperties[] = [  
 {  
 displayName: 'Github Server',  
 name: 'server', // key to use in translation  
 type: 'string',  
 default: 'https://api.github.com',  
 description: 'The server to connect to. Only has to be set if Github Enterprise is used.',  
 },  
 {  
 displayName: 'User',  
 name: 'user', // key to use in translation  
 type: 'string',  
 default: '',  
 },  
 {  
 displayName: 'Access Token',  
 name: 'accessToken', // key to use in translation  
 type: 'string',  
 default: '',  
 },  
 ];  
}

The object for each node credential parameter allows for the keys displayName, description, and placeholder.

{  
 "server.displayName": "🇩🇪 Github Server",  
 "server.description": "🇩🇪 The server to connect to. Only has to be set if Github Enterprise is used.",  
 "user.placeholder": "🇩🇪 Hans",  
 "accessToken.placeholder": "🇩🇪 123"  
}

Only existing parameters are translatable. If a credential parameter does not have a description in the English original, adding a translation for that non-existing parameter will not result in the translation being displayed - the parameter will need to be added in the English original first.

#### Translating the node translation file

**Note**: All keys are optional. Missing translations trigger a fallback to the en locale strings.

Each node translation file is an object that allows for two keys, header and nodeView, which are the *sections* of each node translation.

The header section points to an object that may contain only two keys, displayName and description, matching the node’s description.displayName and description.description.

export class Github implements INodeType {  
 description: INodeTypeDescription = {  
 displayName: 'GitHub', // key to use in translation  
 description: 'Consume GitHub API', // key to use in translation  
 name: 'github',  
 icon: 'file:github.svg',  
 group: ['input'],  
 version: 1,

{  
 "header": {  
 "displayName": "🇩🇪 GitHub",  
 "description": "🇩🇪 Consume GitHub API"  
 }  
}

Header text is used wherever the node’s display name and description are needed:

In turn, the nodeView section points to an object containing translation keys that match the node’s operational parameters, found in the \*.node.ts and also found in \*Description.ts files in the same dir.

export class Github implements INodeType {  
 description: INodeTypeDescription = {  
 displayName: 'GitHub',  
 name: 'github',  
 properties: [  
 {  
 displayName: 'Resource',  
 name: 'resource', // key to use in translation  
 type: 'options',  
 options: [],  
 default: 'issue',  
 description: 'The resource to operate on.',  
 },

{  
 "nodeView.resource.displayName": "🇩🇪 Resource"  
}

A node parameter allows for different translation keys depending on parameter type.

#### string, number and boolean parameters

Allowed keys: displayName, description, placeholder

{  
 displayName: 'Repository Owner',  
 name: 'owner', // key to use in translation  
 type: 'string',  
 required: true,  
 placeholder: 'n8n-io',  
 description: 'Owner of the repository.',  
},

{  
 "nodeView.owner.displayName": "🇩🇪 Repository Owner",  
 "nodeView.owner.placeholder": "🇩🇪 n8n-io",  
 "nodeView.owner.description": "🇩🇪 Owner of the repository"  
}

#### options parameter

Allowed keys: displayName, description, placeholder

Allowed subkeys: options.{optionName}.displayName and options.{optionName}.description.

{  
 displayName: 'Resource',  
 name: 'resource',  
 type: 'options',  
 options: [  
 {  
 name: 'File',  
 value: 'file', // key to use in translation  
 },  
 {  
 name: 'Issue',  
 value: 'issue', // key to use in translation  
 },  
 ],  
 default: 'issue',  
 description: 'Resource to operate on',  
},

{  
 "nodeView.resource.displayName": "🇩🇪 Resource",  
 "nodeView.resource.description": "🇩🇪 Resource to operate on",  
 "nodeView.resource.options.file.name": "🇩🇪 File",  
 "nodeView.resource.options.issue.name": "🇩🇪 Issue"  
}

For nodes whose credentials may be used in the HTTP Request node, an additional option Custom API Call is injected into the Resource and Operation parameters. Use the \_\_CUSTOM\_API\_CALL\_\_ key to translate this additional option.

{  
 "nodeView.resource.options.file.name": "🇩🇪 File",  
 "nodeView.resource.options.issue.name": "🇩🇪 Issue",  
 "nodeView.resource.options.\_\_CUSTOM\_API\_CALL\_\_.name": "🇩🇪 Custom API Call"  
}

#### collection and fixedCollection parameters

Allowed keys: displayName, description, placeholder, multipleValueButtonText

Example of collection parameter:

{  
 displayName: 'Labels',  
 name: 'labels', // key to use in translation  
 type: 'collection',  
 typeOptions: {  
 multipleValues: true,  
 multipleValueButtonText: 'Add Label',  
 },  
 displayOptions: {  
 show: {  
 operation: [  
 'create',  
 ],  
 resource: [  
 'issue',  
 ],  
 },  
 },  
 default: { 'label': '' },  
 options: [  
 {  
 displayName: 'Label',  
 name: 'label', // key to use in translation  
 type: 'string',  
 default: '',  
 description: 'Label to add to issue',  
 },  
 ],  
},

{  
 "nodeView.labels.displayName": "🇩🇪 Labels",  
 "nodeView.labels.multipleValueButtonText": "🇩🇪 Add Label",  
 "nodeView.labels.options.label.displayName": "🇩🇪 Label",  
 "nodeView.labels.options.label.description": "🇩🇪 Label to add to issue",  
 "nodeView.labels.options.label.placeholder": "🇩🇪 Some placeholder"  
}

Example of fixedCollection parameter:

{  
 displayName: 'Additional Parameters',  
 name: 'additionalParameters',  
 placeholder: 'Add Parameter',  
 description: 'Additional fields to add.',  
 type: 'fixedCollection',  
 default: {},  
 displayOptions: {  
 show: {  
 operation: [  
 'create',  
 'delete',  
 'edit',  
 ],  
 resource: [  
 'file',  
 ],  
 },  
 },  
 options: [  
 {  
 name: 'author',  
 displayName: 'Author',  
 values: [  
 {  
 displayName: 'Name',  
 name: 'name',  
 type: 'string',  
 default: '',  
 description: 'Name of the author of the commit',  
 placeholder: 'John',  
 },  
 {  
 displayName: 'Email',  
 name: 'email',  
 type: 'string',  
 default: '',  
 description: 'Email of the author of the commit',  
 placeholder: 'john@email.com',  
 },  
 ],  
 },  
 ],  
}

{  
 "nodeView.additionalParameters.displayName": "🇩🇪 Additional Parameters",  
 "nodeView.additionalParameters.placeholder": "🇩🇪 Add Field",  
 "nodeView.additionalParameters.options.author.displayName": "🇩🇪 Author",  
 "nodeView.additionalParameters.options.author.values.name.displayName": "🇩🇪 Name",  
 "nodeView.additionalParameters.options.author.values.name.description": "🇩🇪 Name of the author of the commit",  
 "nodeView.additionalParameters.options.author.values.name.placeholder": "🇩🇪 Jan",  
 "nodeView.additionalParameters.options.author.values.email.displayName": "🇩🇪 Email",  
 "nodeView.additionalParameters.options.author.values.email.description": "🇩🇪 Email of the author of the commit",  
 "nodeView.additionalParameters.options.author.values.email.placeholder": "🇩🇪 jan@n8n.io"  
}

For information on **reusable dynamic text**, refer to the [Addendum](./ADDENDUM.md).

# Building translations

## Base text

When translating a base text file at /packages/frontend/@n8n/i18n/src/locales/{localeIdentifier}.json:

1. Open a terminal:

export N8N\_DEFAULT\_LOCALE=de  
pnpm start

1. Open another terminal:

export N8N\_DEFAULT\_LOCALE=de  
cd packages/frontend/editor-ui  
pnpm dev

Changing the base text file will trigger a rebuild of the client at http://localhost:8080.

## Dynamic text

When translating a dynamic text file at /packages/nodes-base/nodes/{node}/translations/{localeIdentifier}/{node}.json,

1. Open a terminal:

export N8N\_DEFAULT\_LOCALE=de  
pnpm start

1. Open another terminal:

export N8N\_DEFAULT\_LOCALE=de  
cd packages/nodes-base  
pnpm n8n-generate-translations  
pnpm watch

After changing the dynamic text file:

1. Stop and restart the first terminal.
2. Refresh the browser at http://localhost:5678

If a headerText section was changed, re-run pnpm n8n-generate-translations in /nodes-base.

**Note**: To translate base and dynamic text simultaneously, run three terminals following the steps from both sections (first terminal running only once) and browse http://localhost:8080.

## packages@n8n18n.md

# @n8n/i18n

A package for managing internationalization (i18n) in n8n’s Frontend codebase. It provides a structured way to handle translations and localization, ensuring that the application can be easily adapted to different languages and regions.

## Table of Contents

* [Features](#features)
* [Contributing](#contributing)
* [License](#license)

## Features

* **Translation Management**: Simplifies the process of managing translations for different languages.
* **Localization Support**: Provides tools to adapt the application for different regions and cultures.
* **Easy Integration**: Seamlessly integrates with n8n’s Frontend codebase, making it easy to implement and use.
* **Reusable Base Text**: Allows for the definition of reusable base text strings, reducing redundancy in translations.
* **Pluralization and Interpolation**: Supports pluralization and interpolation in base text strings, making it flexible for various use cases.
* **Versioned Nodes Support**: Facilitates the management of translations for nodes in versioned directories, ensuring consistency across different versions.
* **Documentation**: Comprehensive documentation to help developers understand and utilize the package effectively.

## Contributing

For more details, please read our <CONTRIBUTING.md>.

## License

For more details, please read our <LICENSE.md>.

## packages@n8n-api-client.md

# @n8n/rest-api-client

This package contains the REST API calls for n8n.

## Table of Contents

* [Features](#features)
* [Contributing](#contributing)
* [License](#license)

## Features

* Provides a REST API for n8n
* Supports authentication and authorization

## Contributing

For more details, please read our <CONTRIBUTING.md>.

## License

For more details, please read our <LICENSE.md>.

## packages@n8n.md

# @n8n/stores

A collection of Pinia stores that provide common data-related functionality across n8n’s Front-End packages.

## Table of Contents

* [Features](#features)
* [Contributing](#contributing)
* [License](#license)

## Features

* **Composable State Management**: Share and reuse stateful logic across multiple Vue components using Pinia stores.
* **Consistent Patterns**: Promote uniform state handling and best practices throughout the front-end codebase.
* **Easy Extensibility**: Add or modify stores as project requirements evolve, supporting scalable development.
* **Composition API Support**: Designed to work seamlessly with Vue’s Composition API for modern, maintainable code.

## Contributing

For more details, please read our <CONTRIBUTING.md>.

## License

For more details, please read our <LICENSE.md>.

## packages-ui.md



n8n.io - Workflow Automation

# n8n-editor-ui

The UI to create and update n8n workflows

npm install n8n -g

## Project setup

pnpm install

### Compiles and hot-reloads for development

pnpm serve

### Compiles and minifies for production

pnpm build

### Run your tests

pnpm test

### Lints and fixes files

pnpm lint

### Run your end-to-end tests

pnpm test:e2e

### Run your unit tests

pnpm test:unit

### Customize configuration

See [Configuration Reference](https://cli.vuejs.org/config/).

## License

You can find the license information [here](https://github.com/n8n-io/n8n/blob/master/README.md#license)

## packages-dev.md



n8n.io - Workflow Automation

# n8n-node-dev

Currently very simple and not very sophisticated CLI which makes it easier to create credentials and nodes in TypeScript for n8n.

npm install n8n-node-dev -g

## Contents

* [Usage](#usage)
* [Commands](#commands)
* [Create a node](#create-a-node)
  + [Node Type](#node-type)
  + [Node Type Description](#node-type-description)
  + [Node Properties](#node-properties)
  + [Node Property Options](#node-property-options)
* [License](#license)

## Usage

The commandline tool can be started with n8n-node-dev <COMMAND>

## Commands

The following commands exist:

### build

Builds credentials and nodes in the current folder and copies them into the n8n custom extension folder (~/.n8n/custom/) unless destination path is overwritten with --destination <FOLDER\_PATH>

When “–watch” gets set it starts in watch mode and automatically builds and copies files whenever they change. To stop press “ctrl + c”.

### new

Creates new basic credentials or node of the selected type to have a first starting point.

## Create a node

The easiest way to create a new node is via the “n8n-node-dev” cli. It sets up all the basics.

A n8n node is a JavaScript file (normally written in TypeScript) which describes some basic information (like name, description, …) and also at least one method. Depending on which method gets implemented defines if it is a regular-, trigger- or webhook-node.

A simple regular node which:

* defines one node property
* sets its value to all items it receives

would look like this:

File named: MyNode.node.ts

import {  
 IExecuteFunctions,  
 INodeExecutionData,  
 INodeType,  
 INodeTypeDescription,  
} from 'n8n-workflow';  
  
  
export class MyNode implements INodeType {  
 description: INodeTypeDescription = {  
 displayName: 'My Node',  
 name: 'myNode',  
 group: ['transform'],  
 version: 1,  
 description: 'Adds "myString" on all items to defined value.',  
 defaults: {  
 name: 'My Node',  
 color: '#772244',  
 },  
 inputs: ['main'],  
 outputs: ['main'],  
 properties: [  
 // Node properties which the user gets displayed and  
 // can change on the node.  
 {  
 displayName: 'My String',  
 name: 'myString',  
 type: 'string',  
 default: '',  
 placeholder: 'Placeholder value',  
 description: 'The description text',  
 }  
 ]  
 };  
  
  
 async execute(this: IExecuteFunctions): Promise<INodeExecutionData[][]> {  
  
 const items = this.getInputData();  
  
 let item: INodeExecutionData;  
 let myString: string;  
  
 // Itterates over all input items and add the key "myString" with the  
 // value the parameter "myString" resolves to.  
 // (This could be a different value for each item in case it contains an expression)  
 for (let itemIndex = 0; itemIndex < items.length; itemIndex++) {  
 myString = this.getNodeParameter('myString', itemIndex, '') as string;  
 item = items[itemIndex];  
  
 item.json['myString'] = myString;  
 }  
  
 return [items];  
  
 }  
}

The “description” property has to be set on all nodes because it contains all the base information. Additionally all nodes have to have exactly one of the following methods defined which contains the actual logic:

**Regular node**

Method is called when the workflow gets executed

* execute: Executed once no matter how many items

By default, execute should always be used, especially when creating a third-party integration. The reason for this is that it provides much more flexibility and allows, for example, returning a different number of items than it received as input. This becomes crucial when a node needs to query data such as *return all users*. In such cases, the node typically receives only one input item but returns as many items as there are users. Therefore, when in doubt, it is recommended to use execute!

**Trigger node**

Method is called once when the workflow gets activated. It can then trigger workflow runs and provide the necessary data by itself.

* trigger

**Webhook node**

Method is called when webhook gets called.

* webhook

### Node Type

Property overview

* **description** [required]: Describes the node like its name, properties, hooks, … see Node Type Description bellow.
* **execute** [optional]: Method is called when the workflow gets executed (once).
* **hooks** [optional]: The hook methods.
* **methods** [optional]: Additional methods. Currently only “loadOptions” exists which allows loading options for parameters from external services
* **trigger** [optional]: Method is called once when the workflow gets activated.
* **webhook** [optional]: Method is called when webhook gets called.
* **webhookMethods** [optional]: Methods to setup webhooks on external services.

### Node Type Description

The following properties can be set in the node description:

* **credentials** [optional]: Credentials the node requests access to
* **defaults** [required]: Default “name” and “color” to set on node when it gets created
* **displayName** [required]: Name to display users in Editor UI
* **description** [required]: Description to display users in Editor UI
* **group** [required]: Node group for example “transform” or “trigger”
* **hooks** [optional]: Methods to execute at different points in time like when the workflow gets activated or deactivated
* **icon** [optional]: Icon to display (can be an icon or a font awesome icon)
* **inputs** [required]: Types of inputs the node has (currently only “main” exists) and the amount
* **outputs** [required]: Types of outputs the node has (currently only “main” exists) and the amount
* **outputNames** [optional]: In case a node has multiple outputs, names can be set that users know what data to expect
* **maxNodes** [optional]: If an unlimited number of nodes of that type cannot exist in a workflow, the max-amount can be specified
* **name** [required]: Name of the node (for n8n to use internally, in camelCase)
* **properties** [required]: Properties which get displayed in the Editor UI and can be set by the user
* **subtitle** [optional]: Text which should be displayed underneath the name of the node in the Editor UI (can be an expression)
* **version** [required]: Version of the node. Currently always “1” (integer). For future usage, does not get used yet
* **webhooks** [optional]: Webhooks the node should listen to

### Node Properties

The following properties can be set in the node properties:

* **default** [required]: Default value of the property
* **description** [required]: Description that is displayed to users in the Editor UI
* **displayName** [required]: Name that is displayed to users in the Editor UI
* **displayOptions** [optional]: Defines logic to decide if a property should be displayed or not
* **name** [required]: Name of the property (for n8n to use internally, in camelCase)
* **options** [optional]: The options the user can select when type of property is “collection”, “fixedCollection” or “options”
* **placeholder** [optional]: Placeholder text that is displayed to users in the Editor UI
* **type** [required]: Type of the property. If it is for example a “string”, “number”, …
* **typeOptions** [optional]: Additional options for type. Like for example the min or max value of a number
* **required** [optional]: Defines if the value has to be set or if it can stay empty

### Node Property Options

The following properties can be set in the node property options:

All properties are optional. However, most only work when the node-property is of a specfic type.

* **alwaysOpenEditWindow** [type: json]: If set then the “Editor Window” will always open when the user tries to edit the field. Helpful if long text is typically used in the property
* **loadOptionsMethod** [type: options]: Method to use to load options from an external service
* **maxValue** [type: number]: Maximum value of the number
* **minValue** [type: number]: Minimum value of the number
* **multipleValues** [type: all]: If set the property gets turned into an Array and the user can add multiple values
* **multipleValueButtonText** [type: all]: Custom text for add button in case “multipleValues” were set
* **numberPrecision** [type: number]: The precision of the number. By default, it is “0” and will only allow integers
* **password** [type: string]: If a password field should be displayed (normally only used by credentials because all node data is not encrypted and gets saved in clear-text)
* **rows** [type: string]: Number of rows the input field should have. By default it is “1”

## License

You can find the license information [here](https://github.com/n8n-io/n8n/blob/master/README.md#license)

## packages-base.md

All Stripe webhook events are taken from docs: <https://stripe.com/docs/api/events/types#event_types>

To get the entire list of events as a JS array, scrape the website:

1. manually add the id #event-types to <ul> that contains all event types
2. copy-paste the function in the JS console
3. the result is copied into in the clipboard
4. paste the prepared array in StripeTrigger.node.ts

types = [];  
$$('ul#event-types li').forEach((el) => {  
 const value = el.querySelector('.method-list-item-label-name').innerText;  
  
 types.push({  
 name: value  
 .replace(/(\.|\_)/, ' ')  
 .split(' ')  
 .map((s) => s.charAt(0).toUpperCase() + s.substring(1))  
 .join(' '),  
 value,  
 description: el.querySelector('.method-list-item-description').innerText,  
 });  
});  
copy(types);

## packages-base.md



n8n.io - Workflow Automation

# n8n-nodes-base

The nodes which are included by default in n8n

npm install n8n-nodes-base -g

## License

You can find the license information [here](https://github.com/n8n-io/n8n/blob/master/README.md#license)

## packages.md

# n8n Test Containers - Usage Guide

A simple way to spin up n8n container stacks for development and testing.

## Quick Start

# Start a basic n8n instance (SQLite database)  
pnpm stack  
  
# Start with PostgreSQL database  
pnpm stack --postgres  
  
# Start in queue mode (with Redis + PostgreSQL)  
pnpm stack --queue  
  
# Start with starter performance plan constraints  
pnpm stack:starter

When started, you’ll see: - **URL**: http://localhost:[random-port]

## Common Usage Patterns

### Development with Container Reuse

# Enable container reuse (faster restarts)  
pnpm run stack # SQLite  
pnpm run stack:postgres # PostgreSQL  
pnpm run stack:queue # Queue mode  
pnpm run stack:multi-main # Multiple main instances  
pnpm run stack:starter # Starter performance plan

### Performance Plan Presets

# Use predefined performance plans (simulates cloud constraints, differs from cloud CPU wise due to non burstable docker)  
pnpm stack --plan trial # Trial: 0.75GB RAM, 0.2 CPU (SQLite only)  
pnpm stack --plan starter # Starter: 0.75GB RAM, 0.2 CPU (SQLite only)  
pnpm stack --plan pro-1 # Pro-1: 1.25GB RAM, 0.5 CPU (SQLite only)  
pnpm stack --plan pro-2 # Pro-2: 2.5GB RAM, 0.75 CPU (SQLite only)  
pnpm stack --plan enterprise # Enterprise: 8GB RAM, 1.0 CPU (SQLite only)

### Queue Mode with Scaling

# Custom scaling: 3 main instances, 5 workers  
pnpm stack --queue --mains 3 --workers 5  
  
# Single main, 2 workers  
pnpm stack --queue --workers 2

### Environment Variables

# Set custom environment variables  
pnpm run stack --postgres --env N8N\_LOG\_LEVEL=info --env N8N\_ENABLED\_MODULES=insights

### Parallel Testing

# Run multiple stacks in parallel with unique names  
pnpm run stack --name test-1 --postgres  
pnpm run stack --name test-2 --queue

## Custom Container Config

### Via Command Line

# Pass any n8n env vars to containers  
N8N\_TEST\_ENV='{"N8N\_METRICS":"true"}' npm run stack:standard  
N8N\_TEST\_ENV='{"N8N\_LOG\_LEVEL":"debug","N8N\_METRICS":"true","N8N\_ENABLED\_MODULES":"insights"}' npm run stack:postgres

## Programmatic Usage

import { createN8NStack } from './containers/n8n-test-containers';  
  
// Simple SQLite instance  
const stack = await createN8NStack();  
  
// PostgreSQL with custom environment  
const stack = await createN8NStack({  
 postgres: true,  
 env: { N8N\_LOG\_LEVEL: 'debug' }  
});  
  
// Queue mode with scaling  
const stack = await createN8NStack({  
 queueMode: { mains: 2, workers: 3 }  
});  
  
// Resource-constrained container (simulating cloud plans)  
const stack = await createN8NStack({  
 resourceQuota: {  
 memory: 0.375, // 384MB RAM  
 cpu: 0.25 // 250 millicore CPU  
 }  
});  
  
// Use the stack  
console.log(`n8n available at: ${stack.baseUrl}`);  
  
// Clean up when done  
await stack.stop();

## Configuration Options

| Option | Description | Example |
| --- | --- | --- |
| --postgres | Use PostgreSQL instead of SQLite | npm run stack -- --postgres |
| --queue | Enable queue mode with Redis | npm run stack -- --queue |
| --mains <n> | Number of main instances (requires queue mode) | --mains 3 |
| --workers <n> | Number of worker instances (requires queue mode) | --workers 5 |
| --name <name> | Custom project name for parallel runs | --name my-test |
| --env KEY=VALUE | Set environment variables | --env N8N\_LOG\_LEVEL=debug |
| --plan <plan> | Use performance plan preset | --plan starter |

## Performance Plans

Simulate cloud plan resource constraints for testing. **Performance plans are SQLite-only** (like cloud n8n):

# CLI usage  
pnpm stack --plan trial # 0.375GB RAM, 0.2 CPU cores  
pnpm stack --plan starter # 0.375GB RAM, 0.2 CPU cores  
pnpm stack --plan pro-1 # 0.625GB RAM, 0.5 CPU cores  
pnpm stack --plan pro-2 # 1.25GB RAM, 0.75 CPU cores  
pnpm stack --plan enterprise # 4GB RAM, 1.0 CPU cores

**Common Cloud Plan Quotas:** - **Trial/Starter**: 0.375GB RAM, 0.2 CPU cores - **Pro-1**: 0.625GB RAM, 0.5 CPU cores - **Pro-2**: 1.25GB RAM, 0.75 CPU cores - **Enterprise**: 4GB RAM, 1.0 CPU cores

Resource quotas are applied using Docker’s --memory and --cpus flags for realistic cloud simulation.

## Package.json Scripts

| Script | Description | Equivalent CLI |
| --- | --- | --- |
| stack | Basic SQLite instance | pnpm stack |
| stack:postgres | PostgreSQL database | pnpm stack --postgres |
| stack:queue | Queue mode | pnpm stack --queue |
| stack:multi-main | Multi-main setup | pnpm stack --mains 2 --workers 1 |
| stack:starter | Starter performance plan (SQLite only) | pnpm stack --plan starter |

## Container Architecture

### Single Instance (Default)

┌─────────────┐  
│ n8n │ ← SQLite database  
│ (SQLite) │  
└─────────────┘

### With PostgreSQL

┌─────────────┐ ┌──────────────┐  
│ n8n │────│ PostgreSQL │  
│ │ │ │  
└─────────────┘ └──────────────┘

### Queue Mode

┌─────────────┐ ┌──────────────┐ ┌─────────────┐  
│ n8n-main │────│ PostgreSQL │ │ Redis │  
└─────────────┘ └──────────────┘ └─────────────┘  
┌─────────────┐ │ │  
│ n8n-worker │────────────────────────┘ │  
└─────────────┘ │  
┌─────────────┐ │  
│ n8n-worker │──────────────────────────────────────┘  
└─────────────┘

### Multi-Main with Load Balancer

┌──────────────┐  
 ────│ │ ← Entry point  
 / │ Load Balancer│  
┌─────────────┐ └──────────────┘  
│ n8n-main-1 │────┐  
└─────────────┘ │ ┌──────────────┐ ┌─────────────┐  
┌─────────────┐ ├─│ PostgreSQL │ │ Redis │  
│ n8n-main-2 │────┤ └──────────────┘ └─────────────┘  
└─────────────┘ │ │ │  
┌─────────────┐ │ ┌─────────────────────────────────┤  
│ n8n-worker │────┘ │ │  
└─────────────┘ └─────────────────────────────────┘

## Cleanup

# Remove all n8n containers and networks  
pnpm run stack:clean:all  
  
  
## Tips  
  
- \*\*Container Reuse\*\*: Set `TESTCONTAINERS\_REUSE\_ENABLE=true` for faster development cycles  
- \*\*Parallel Testing\*\*: Use `--name` parameter to run multiple stacks without conflicts  
- \*\*Queue Mode\*\*: Automatically enables PostgreSQL (required for queue mode)  
- \*\*Multi-Main\*\*: Requires queue mode and special licensing read from N8N\_LICENSE\_ACTIVATION\_KEY environment variable  
- \*\*Performance Plans\*\*: Use `--plan` for quick cloud plan simulation  
- \*\*Log Monitoring\*\*: Use the `ContainerTestHelpers` class for advanced log monitoring in tests  
  
## Docker Image  
  
By default, uses the `n8nio/n8n:local` image. Override with:  
```bash  
export N8N\_DOCKER\_IMAGE=n8nio/n8n:dev  
pnpm run stack

## packages.md

# Playwright E2E Test Guide

## Development setup

pnpm install-browsers:local # in playwright directory  
pnpm build:docker # from root first to test against local changes

## Quick Start

pnpm test:all # Run all tests (fresh containers, pnpm build:docker from root first to ensure local containers)  
pnpm test:local # Starts a local server and runs the UI tests  
N8N\_BASE\_URL=localhost:5068 pnpm test:local # Runs the UI tests against the instance running

## Test Commands

# By Mode  
pnpm test:container:standard # Sqlite  
pnpm test:container:postgres # PostgreSQL  
pnpm test:container:queue # Queue mode  
pnpm test:container:multi-main # HA setup  
  
pnpm test:performance # Runs the performance tests against Sqlite container  
pnpm test:chaos # Runs the chaos tests  
  
  
# Development  
pnpm test:all --grep "workflow" # Pattern match, can run across all test types UI/cli-workflow/performance  
pnpm test:local --ui # To enable UI debugging and test running mode

## Test Tags

test('basic test', ...) // All modes, fully parallel  
test('postgres only @mode:postgres', ...) // Mode-specific  
test('needs clean db @db:reset', ...) // Sequential per worker  
test('chaos test @mode:multi-main @chaostest', ...) // Isolated per worker  
test('cloud resource test @cloud:trial', ...) // Cloud resource constraints  
test('proxy test @capability:proxy', ...) // Requires proxy server capability

## Fixture Selection

* **base.ts**: Standard testing with worker-scoped containers (default choice)
* **cloud-only.ts**: Cloud resource testing with guaranteed isolation
  + Use for performance testing under resource constraints
  + Requires @cloud:\* tags (@cloud:trial, @cloud:enterprise, etc.)
  + Creates only cloud containers, no worker containers

// Standard testing  
import { test, expect } from '../fixtures/base';  
  
// Cloud resource testing  
import { test, expect } from '../fixtures/cloud-only';  
test('Performance under constraints @cloud:trial', async ({ n8n, api }) => {  
 // Test runs with 384MB RAM, 250 millicore CPU  
});

## Tips

* test:\* commands use fresh containers (for testing)
* VS Code: Set N8N\_BASE\_URL in Playwright settings to run tests directly from VS Code
* Pass custom env vars via N8N\_TEST\_ENV='{"KEY":"value"}'

## Project Layout

* **composables**: Multi-page interactions (e.g., WorkflowComposer.executeWorkflowAndWaitForNotification())
* **config**: Test setup and configuration (constants, test users, etc.)
* **fixtures**: Custom test fixtures extending Playwright’s base test
  + base.ts: Standard fixtures with worker-scoped containers
  + cloud-only.ts: Cloud resource testing with test-scoped containers only
* **pages**: Page Object Models for UI interactions
* **services**: API helpers for E2E controller, REST calls, workflow management, etc.
* **utils**: Utility functions (string manipulation, helpers, etc.)
* **workflows**: Test workflow JSON files for import/reuse

## Writing Tests with Proxy

You can use ProxyServer to mock API requests.

import { test, expect } from '../fixtures/base';  
  
// The `@capability:proxy` tag ensures tests only run when proxy infrastructure is available.  
test.describe('Proxy tests @capability:proxy', () => {  
 test('should mock HTTP requests', async ({ proxyServer, n8n }) => {  
 // Create mock expectations  
 await proxyServer.createGetExpectation('/api/data', { result: 'mocked' });  
  
 // Execute workflow that makes HTTP requests  
 await n8n.canvas.openNewWorkflow();  
 // ... test implementation  
  
 // Verify requests were proxied  
 expect(await proxyServer.wasGetRequestMade('/api/data')).toBe(true);  
 });  
});

### Recording and replaying requests

The ProxyServer service supports recording HTTP requests for test mocking and replay. All proxied requests are automatically recorded by the mock server as described in the [Mock Server documentation](https://www.mock-server.com/proxy/record_and_replay.html).

#### Recording Expectations

// Record all requests (the request is simplified/cleansed to method/path/body/query)  
await proxyServer.recordExpectations('test-folder');  
  
// Record with filtering and options  
await proxyServer.recordExpectations('test-folder', {  
 host: 'googleapis.com', // Filter by host (partial match)  
 dedupe: true, // Remove duplicate requests  
 raw: false // Save cleaned requests (default)  
});  
  
// Record raw requests with all headers and metadata  
await proxyServer.recordExpectations('test-folder', {  
 raw: true // Save complete original requests  
});  
  
// Record requests matching specific criteria  
await proxyServer.recordExpectations('test-folder', {  
 pathOrRequestDefinition: {  
 method: 'POST',  
 path: '/api/workflows'  
 }  
});

#### Loading and Using Recorded Expectations

Recorded expectations are saved as JSON files in the expectations/ directory. To use them in tests, you must explicitly load them:

test('should use recorded expectations', async ({ proxyServer }) => {  
 // Load expectations from a specific folder  
 await proxyServer.loadExpectations('test-folder');  
  
 // Your test code here - requests will be mocked using loaded expectations  
});

#### Important: Cleanup Expectations

**Remember to clean up expectations before or after test runs:**

test.beforeEach(async ({ proxyServer }) => {  
 // Clear any existing expectations before test  
 await proxyServer.clearAllExpectations();  
});  
  
test.afterEach(async ({ proxyServer }) => {  
 // Or clear expectations after test  
 await proxyServer.clearAllExpectations();  
});

This prevents expectations from one test affecting others and ensures test isolation.

## Writing Tests

For guidelines on writing new tests, see [CONTRIBUTING.md](./CONTRIBUTING.md).

## packages-workflows.md

# Workflow Testing Framework

## Introduction

This framework tests n8n’s nodes and workflows to:

* ✅ **Ensure Correctness:** Verify that nodes operate correctly
* 🔄 **Maintain Compatibility:** Detect breaking changes in external APIs
* 🔒 **Guarantee Stability:** Prevent regressions in new releases

## Our Move to Playwright

This framework is an evolution of a previous system. We moved to **Playwright** as our test runner to leverage its powerful, industry-standard features, resulting in:

* **Simpler Commands:** A single command to run tests, with simple flags for control
* **Better Reporting:** Rich, interactive HTML reports with visual diffs
* **Built-in Features:** Automatic retries, parallel execution, and CI integration out of the box
* **Schema Validation:** Detect structural changes that indicate API breaking changes

## 🚀 Quick Start

### Prerequisites

1. **Set encryption key:** The test credentials are encrypted. Add to ~/.n8n/config:

* {  
   "N8N\_ENCRYPTION\_KEY": "YOUR\_KEY\_FROM\_BITWARDEN"  
  }
* Find the key in Bitwarden under “Testing Framework encryption key”

1. **Fresh database (optional):** For a clean start, remove ~/.n8n/database.sqlite if it exists
2. **Setup Environment**: pnpm test:workflows:setup

### Basic Commands

# 1. Basic execution test (just verify workflows run without errors)  
pnpm test:workflows  
  
# 2. Run with schema validation  
SCHEMA=true pnpm test:workflows  
  
# 3. Update schema snapshots (when output structure changes)  
pnpm test:workflows --update-snapshots  
  
# 4. Run specific workflows (using grep)  
pnpm test:workflows -g "email"

### View Test Results

After any test run, open the interactive HTML report:

npx playwright show-report

The report shows: \* ✅ Passed/❌ Failed tests with execution times \* 📸 Schema diffs showing structural changes \* ⚠️ Warnings and annotations \* 📊 Test trends over time (in CI)

## ⚙️ How It Works

### Test Modes

1. **Basic Run** (default): Executes workflows and checks for errors
2. **Schema Mode** (SCHEMA=true): Validates workflow output structure against saved schemas

### Schema Validation (Recommended)

Schema validation captures the **structure** of workflow outputs, not the values. This is ideal for: - Detecting API breaking changes (field renames, type changes) - Avoiding false positives from legitimate data variations - Maintaining stable tests across different environments

When enabled, the framework: 1. Generates a JSON schema from the workflow output 2. Compares it against the saved schema snapshot 3. Reports any structural differences

**Example of what schema validation catches:**

// Original API response  
{ user: { name: "John", email: "john@example.com" } }  
  
// Changed API response (field renamed)  
{ user: { fullName: "John", email: "john@example.com" } }  
// ❌ Schema validation catches this immediately!

### Why Schema Over Value Comparison?

Traditional value comparison often leads to: - 🔴 False positives from timestamps, IDs, and other dynamic data - 🔴 Constant snapshot updates for legitimate data changes - 🔴 Missing actual breaking changes when values happen to match

Schema validation focuses on what matters: - ✅ Data structure and types - ✅ Field presence and naming - ✅ API contract stability

## 📋 Configuration

### workflowConfig.json

Controls workflow execution and testing behavior:

[  
 {  
 "workflowId": "123",  
 "status": "ACTIVE",  
 "enableSchemaValidation": true  
 },  
 {  
 "workflowId": "456",  
 "status": "SKIPPED",  
 "skipReason": "Depends on external API that is currently down",  
 "ticketReference": "JIRA-123"  
 }  
]

**Configuration Fields:** - workflowId: The ID of the workflow (must match the filename) - status: Either “ACTIVE” or “SKIPPED” - enableSchemaValidation: (optional) Whether to use schema validation (default: true) - skipReason: (optional) Why the workflow is skipped - ticketReference: (optional) Related ticket for tracking

## 🎯 Workflow for New Tests

### Step-by-Step Process

# 1. Create/modify workflow in n8n UI  
# 2. Export the workflow  
./packages/cli/bin/n8n export:workflow --separate --output=test-workflows/workflows --pretty --id=XXX  
  
# 3. Add configuration entry to workflowConfig.json  
# Edit workflowConfig.json and add:  
{  
 "workflowId": "XXX",  
 "status": "ACTIVE",  
 "enableSchemaValidation": true  
}  
  
# 4. Test basic execution  
pnpm test:workflows -g "XXX"  
  
# 5. Create initial schema snapshot  
SCHEMA=true pnpm test:workflows --update-snapshots -g "XXX"  
  
# 6. Verify schema validation works  
SCHEMA=true pnpm test:workflows -g "XXX"  
  
# 7. Commit all changes  
git add test-workflows/workflows/XXX.json  
git add \_\_snapshots\_\_/workflow-XXX-schema.snap  
git add workflowConfig.json

## 💡 Common Scenarios

### “I just want to check if workflows run”

pnpm test:workflows

### “I want to ensure API compatibility”

SCHEMA=true pnpm test:workflows

### “An API legitimately changed its structure”

# Update the schema snapshot  
pnpm test:workflows --update-snapshots -g "workflow-name"  
# Note: --update-snapshots automatically enables schema mode

### “I want to skip a workflow temporarily”

Update workflowConfig.json:

{  
 "workflowId": "123",  
 "status": "SKIPPED",  
 "skipReason": "API endpoint is under maintenance",  
 "ticketReference": "SUPPORT-456"  
}

## 🔧 Creating Test Workflows

### Best Practices

1. **One node per workflow:** Test a single node with multiple operations/resources
2. **Use test files:** Reference the files automatically copied to /tmp by setup
3. **Limit results:** Set “Limit” to 1 for “Get All” operations when possible
4. **Handle throttling:** Add wait/sleep nodes for rate-limited APIs
5. **Focus on structure:** Schema validation handles dynamic values automatically

### Available Test Files

The setup automatically copies these to /tmp: - n8n-logo.png - n8n-screenshot.png - PDF test files: - 04-valid.pdf - 05-versions-space.pdf

### Exporting Credentials

When credentials expire or need updating:

# Update the credential in n8n UI  
# Export all credentials (encrypted)  
./packages/cli/bin/n8n export:credentials --output=test-workflows/credentials.json --all --pretty

⚠️ **Never use --decrypted when exporting credentials!**

## 🐛 Troubleshooting

### Tests fail with “No valid JSON output found”

The workflow likely has console.log statements. Remove them or ensure they don’t interfere with JSON output.

### Schema differences for legitimate changes

When an API or node output structure legitimately changes:

pnpm test:workflows --update-snapshots -g "affected-workflow"

### Setup didn’t run / Need to re-run setup

pnpm test:workflows:setup

### Workflow not found

Ensure the workflow was exported to the test-workflows/workflows directory and the workflowId in workflowConfig.json matches the filename.

## 🔄 Setup Process

pnpm test:workflows:setup

## 📊 Understanding Test Output

### Test Status

* **✅ PASSED:** Workflow executed successfully (and schema matched if enabled)
* **❌ FAILED:** Workflow execution failed or schema didn’t match
* **⏭️ SKIPPED:** Workflow marked as SKIPPED in configuration

### Schema Comparison

When schema validation is enabled, the test compares: - Data types (string, number, boolean, array, object) - Object properties and their types - Array element types - Overall structure depth and shape

Schema validation ignores actual values, focusing purely on structure, making tests more stable and meaningful.

## packages.md

# Performance Testing Helper

A simple toolkit for measuring and asserting performance in Playwright tests.

## Quick Start

### “I just want to measure how long something takes”

const duration = await measurePerformance(page, 'open-node', async () => {  
 await n8n.canvas.openNode('Code');  
});  
console.log(`Opening node took ${duration.toFixed(1)}ms`);

### “I want to ensure an action completes within a time limit”

const openNodeDuration = await measurePerformance(page, 'open-node', async () => {  
 await n8n.canvas.openNode('Code');  
});  
expect(openNodeDuration).toBeLessThan(2000); // Must complete in under 2 seconds

### “I want to measure the same action multiple times”

const stats = [];  
for (let i = 0; i < 20; i++) {  
 const duration = await measurePerformance(page, `open-node-${i}`, async () => {  
 await n8n.canvas.openNode('Code');  
 });  
 await n8n.ndv.clickBackToCanvasButton();  
 stats.push(duration);  
}  
const average = stats.reduce((a, b) => a + b, 0) / stats.length;  
console.log(`Average: ${average.toFixed(1)}ms`);  
expect(average).toBeLessThan(2000);

### “I want to set performance budgets for different actions”

const budgets = {  
 triggerWorkflow: 8000, // 8 seconds  
 openLargeNode: 2500, // 2.5 seconds  
};  
  
// Measure workflow execution  
const triggerDuration = await measurePerformance(page, 'trigger-workflow', async () => {  
 await n8n.workflowComposer.executeWorkflowAndWaitForNotification('Successful');  
});  
expect(triggerDuration).toBeLessThan(budgets.triggerWorkflow);  
  
// Measure node opening  
const openDuration = await measurePerformance(page, 'open-large-node', async () => {  
 await n8n.canvas.openNode('Code');  
});  
expect(openDuration).toBeLessThan(budgets.openLargeNode);

### “I want to test performance with different data sizes”

const testData = [  
 { size: 30000, budgets: { triggerWorkflow: 8000, openLargeNode: 2500 } },  
 { size: 60000, budgets: { triggerWorkflow: 15000, openLargeNode: 6000 } },  
];  
  
testData.forEach(({ size, budgets }) => {  
 test(`performance - ${size.toLocaleString()} items`, async ({ page }) => {  
 // Setup test with specific data size  
 await setupTest(size);  
  
 // Measure against size-specific budgets  
 const duration = await measurePerformance(page, 'trigger-workflow', async () => {  
 await n8n.workflowComposer.executeWorkflowAndWaitForNotification('Successful')  
 });  
 expect(duration).toBeLessThan(budgets.triggerWorkflow);  
 });  
});

### “I want to see all performance metrics from my test”

// After running various performance measurements...  
const allMetrics = await getAllPerformanceMetrics(page);  
console.log('All performance metrics:', allMetrics);  
// Output: { 'open-node': 1234.5, 'save-workflow': 567.8, ... }

### “I want to attach performance results to my test report”

const allMetrics = await getAllPerformanceMetrics(page);  
await test.info().attach('performance-metrics', {  
 body: JSON.stringify({  
 dataSize: 30000,  
 metrics: allMetrics,  
 budgets: { triggerWorkflow: 8000, openLargeNode: 2500 },  
 passed: {  
 triggerWorkflow: allMetrics['trigger-workflow'] < 8000,  
 openNode: allMetrics['open-large-node'] < 2500,  
 }  
 }, null, 2),  
 contentType: 'application/json',  
});

## API Reference

### measurePerformance(page, actionName, actionFn)

Measures the duration of an async action using the Performance API. - **Returns:** Promise<number> - Duration in milliseconds

### getAllPerformanceMetrics(page)

Retrieves all performance measurements from the current page. - **Returns:** Promise<Record<string, number>> - Map of action names to durations

## Tips

* Use unique names for measurements in loops (e.g., open-node-${i}) to avoid conflicts
* Set realistic budgets - add some buffer to account for variance
* Consider different budgets for different data sizes or environments

## packages.md



n8n.io - Workflow Automation

# n8n-workflow

Workflow base code for n8n

npm install n8n-workflow

## License

You can find the license information [here](https://github.com/n8n-io/n8n/blob/master/README.md#license)

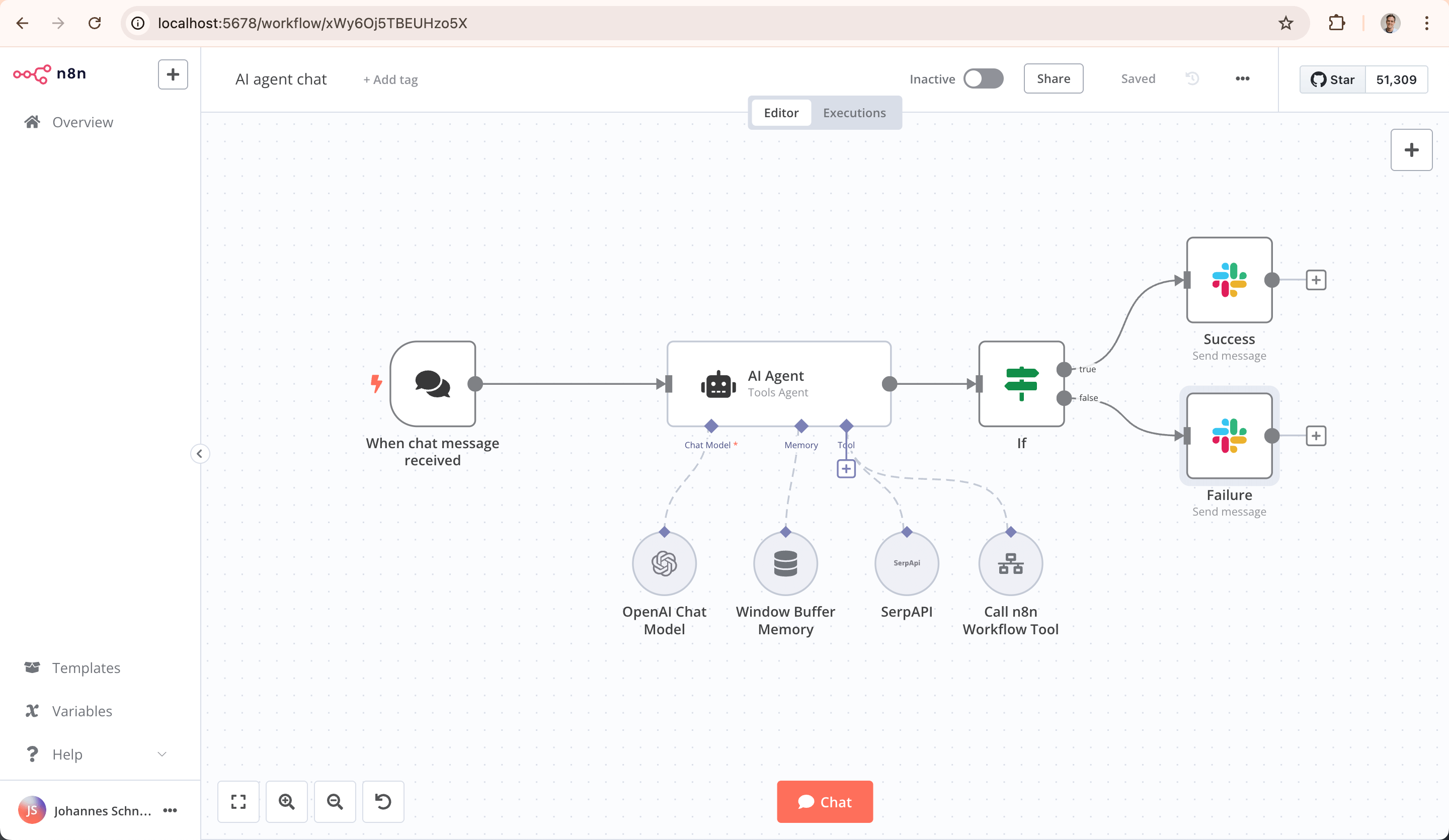
## README.md



Banner image

# n8n - Secure Workflow Automation for Technical Teams

n8n is a workflow automation platform that gives technical teams the flexibility of code with the speed of no-code. With 400+ integrations, native AI capabilities, and a fair-code license, n8n lets you build powerful automations while maintaining full control over your data and deployments.



n8n.io - Screenshot

## Key Capabilities

* **Code When You Need It**: Write JavaScript/Python, add npm packages, or use the visual interface
* **AI-Native Platform**: Build AI agent workflows based on LangChain with your own data and models
* **Full Control**: Self-host with our fair-code license or use our [cloud offering](https://app.n8n.cloud/login)
* **Enterprise-Ready**: Advanced permissions, SSO, and air-gapped deployments
* **Active Community**: 400+ integrations and 900+ ready-to-use [templates](https://n8n.io/workflows)

## Quick Start

Try n8n instantly with [npx](https://docs.n8n.io/hosting/installation/npm/) (requires [Node.js](https://nodejs.org/en/)):

npx n8n

Or deploy with [Docker](https://docs.n8n.io/hosting/installation/docker/):

docker volume create n8n\_data  
docker run -it --rm --name n8n -p 5678:5678 -v n8n\_data:/home/node/.n8n docker.n8n.io/n8nio/n8n

Access the editor at http://localhost:5678

## Resources

* 📚 [Documentation](https://docs.n8n.io)
* 🔧 [400+ Integrations](https://n8n.io/integrations)
* 💡 [Example Workflows](https://n8n.io/workflows)
* 🤖 [AI & LangChain Guide](https://docs.n8n.io/langchain/)
* 👥 [Community Forum](https://community.n8n.io)
* 📖 [Community Tutorials](https://community.n8n.io/c/tutorials/28)

## Support

Need help? Our community forum is the place to get support and connect with other users: [community.n8n.io](https://community.n8n.io)

## License

n8n is [fair-code](https://faircode.io) distributed under the [Sustainable Use License](https://github.com/n8n-io/n8n/blob/master/LICENSE.md) and [n8n Enterprise License](https://github.com/n8n-io/n8n/blob/master/LICENSE_EE.md).

* **Source Available**: Always visible source code
* **Self-Hostable**: Deploy anywhere
* **Extensible**: Add your own nodes and functionality

[Enterprise licenses](mailto:license@n8n.io) available for additional features and support.

Additional information about the license model can be found in the [docs](https://docs.n8n.io/reference/license/).

## Contributing

Found a bug 🐛 or have a feature idea ✨? Check our [Contributing Guide](https://github.com/n8n-io/n8n/blob/master/CONTRIBUTING.md) to get started.

## Join the Team

Want to shape the future of automation? Check out our [job posts](https://n8n.io/careers) and join our team!

## What does n8n mean?

**Short answer:** It means “nodemation” and is pronounced as n-eight-n.

**Long answer:** “I get that question quite often (more often than I expected) so I decided it is probably best to answer it here. While looking for a good name for the project with a free domain I realized very quickly that all the good ones I could think of were already taken. So, in the end, I chose nodemation. ‘node-’ in the sense that it uses a Node-View and that it uses Node.js and ‘-mation’ for ‘automation’ which is what the project is supposed to help with. However, I did not like how long the name was and I could not imagine writing something that long every time in the CLI. That is when I then ended up on ‘n8n’.” - **Jan Oberhauser, Founder and CEO, n8n.io**