ChatGPT

n8n Workflow Automation Reference

Overview of n8n and Its Architecture

- **What is n8n:** n8n is a **source-available, low-code workflow automation tool** that can be self-hosted [1] [2]. It enables users to integrate various services and APIs via workflows without extensive coding, while still allowing code where needed.

- **Core Architecture:** n8n runs as a Node.js application using a database (SQLite by default) for persistence [3]. By default, it runs in a single-instance mode (all workflows execute in one process). For scalability, n8n supports a **queue mode** where multiple worker processes execute jobs from a Redis queue [4]. In queue mode, one main instance handles triggers (like webhooks and schedules) and pushes execution tasks to Redis; multiple worker instances then pull from Redis, execute workflows, and report back [5]. This architecture allows horizontal scaling by adding or removing workers on demand [6].

- **Database and Persistence:** n8n uses an SQL database to store workflow configurations, execution logs, user credentials, etc. By default it uses SQLite, though other databases (e.g. PostgreSQL) can be configured [3]. Database tables include collections for workflows, credentials, executions, etc., which n8n manages via an ORM [7] [8].

- **Execution Model:** n8n executes workflows either manually (on demand) or automatically when triggered. Each workflow can be **active** or **inactive** – only active workflows respond to trigger events. New workflows are created inactive by default and need to be **activated** (enabled) to run automatically [9]. When a workflow is triggered (by an event, schedule, etc.), n8n creates an execution and processes each node step-by-step. If a workflow is inactive, it will not respond to triggers and can only be run manually for testing [9].

Using Nodes and Integrations

- **Nodes as Building Blocks:** In n8n, **nodes** represent individual steps or integrations in a workflow. They are the basic building blocks that perform actions such as **starting the workflow, retrieving data, processing data, and sending data** to other systems [10]. A workflow is a collection of nodes connected in a sequence or branching structure to define an automation process [11]. You can connect multiple nodes to build complex workflows that involve multiple apps or services [11].

- **Triggers vs Actions:** Nodes can operate as either **Trigger** nodes or **Action** nodes. **Trigger nodes** are entry points that start a workflow in response to some event or condition (e.g. an incoming webhook, a CRON schedule, or an app-specific event) [12]. Triggers initiate the execution of the workflow. **Action nodes** (non-trigger nodes) perform specific tasks within an active workflow, such as querying an API, transforming data, or sending an email [12]. When adding a node in the editor, n8n distinguishes trigger operations (marked with a bolt icon) from regular actions [12]. Each integration may provide multiple operations – for example, an HTTP node might offer both a trigger operation (listening for requests) and action operations (making outbound requests).

- **Built-in and Community Integrations:** n8n comes with hundreds of **built-in nodes (integrations)** for popular services and databases [13]. These cover common applications (e.g. Slack, Google Sheets, Twitter, databases, etc.) and core functionality (HTTP requests, webhooks, code execution, logic). In addition, n8n supports an ecosystem of **community nodes** developed by the community, which can be installed to extend n8n's integration library [13]. If an integration is not available out-of-the-box, you can often use the generic **HTTP Request** node to call any external API, or create a custom node. n8n even allows developers to **create their own nodes** (custom code packages) to add new integrations [13].

- **Node Structure:** Each node has a type (which determines its behavior/integration), a set of **parameters** (inputs or settings for the node's operation), optional **credentials** for authentication, and **connections** defining how data flows in and out. Nodes may produce one or more output data items which subsequent nodes can consume. Under the hood, an exported workflow's JSON represents nodes with fields like `name`, `type`, `typeVersion`, `position` (UI location), `parameters` (settings/fields values), and `credentials` (references to auth credentials) along with a global **connections** map that links node outputs to inputs of other nodes.

- **Credentials Management:** Many integration nodes require authentication credentials (API keys, tokens, OAuth, etc.). n8n manages these via a separate **Credentials** system. **Credentials** in n8n are securely stored pieces of info that allow nodes to authenticate with external services [14]. Users create credentials (for example, an OAuth token for Google Sheets or an API key for SendGrid) in the Credentials section of n8n. Nodes then reference the credential by name. This separation allows reusing credentials across nodes and workflows. When exporting workflows, credential names/IDs are included as references (but not the secret values) [15]. Credentials can also be shared between users in multi-user setups [16].

- **Data Flow and Items:** n8n operates on a streaming data model. **Data passes between nodes as an array of objects**, where each object is one **item** (a JSON structure containing relevant fields) [17]. Nodes receive an array of items as input and **automatically iterate** their operation over each item [17]. For example, if one node outputs 5 items, the next node in the workflow will execute its action 5 times (once per item). This built-in looping mechanism means you usually *don't need explicit loop constructs* for item-wise processing – each node handles each incoming item independently. Nodes can also output multiple items (e.g. splitting one item into many, or combining multiple inputs). The array-of-objects data structure is central to how n8n handles transformations (it's essentially an ETL pattern: Extract, Transform, Load) [18] [17].

- **Flow Logic Nodes:** Besides integration nodes, n8n provides **core logic nodes** to control workflow execution flow. These include nodes like **If**, **Switch**, **Merge**, **Wait**, etc. The **If** node allows conditional branching (executing different branches based on a boolean condition), **Switch** can route data based on a field value or expression (like a switch/case statement), and **Merge** can combine or aggregate data from parallel branches. Using these, you can build complex workflows with forks and joins in logic [19]. For instance, you might use an IF node to check a condition and send data to different paths, or use Merge to wait for two parallel processes to finish and then continue. n8n's flow nodes thus enable implementing patterns like conditional execution, looping (e.g. a circular loop can be made by feeding an output back into a previous node), parallel branches, and error handling flows.

Workflow Configuration and Execution

- **Creating Workflows:** Workflows are created and edited in n8n's visual editor (the *workflow canvas*). A workflow is essentially a directed graph of nodes connected by edges. You typically start by **adding a trigger node** (for scheduled workflows you might use a Cron trigger; for event-based workflows you might use a Webhook or app trigger). Then you **connect additional nodes** for each step of your automation. Nodes are connected by dragging from one node's output port to another node's input port in the editor. This connection defines the execution order and data flow. For example, you could start with a trigger (say, "Webhook" listening for an HTTP request), then connect it to an "IF" node to filter data, then to an email-sending node (like Gmail) to notify someone, etc. There can be multiple branches and even multiple trigger entry points in advanced cases (though most workflows have a single start).

- **Workflow Settings:** Each workflow has settings such as name, active status, and execution behavior options. **Activation** is crucial for trigger-based workflows: by default workflows are inactive (disabled) when created, meaning triggers won't work [9]. You must activate a workflow to have n8n start the trigger listeners (e.g. schedule timers, webhook endpoints). While active, any incoming trigger event will cause n8n to execute the workflow. Workflows also have settings for error behavior and result handling. For instance, you can choose if successful or failed executions should be saved to the database (for logging) or not. In the workflow settings you can also specify if the workflow should stop on error or continue (this can be overridden by node-level error settings too). By configuring these, you can control reliability and performance (e.g. turning off saving execution data for high-volume workflows).

- **Parameters and Mapping:** Configuring a workflow largely involves setting **node parameters**. Each node type exposes fields or options in the editor which you fill in. For example, an HTTP Request node has parameters like URL, method, request body, etc.; a Google Sheets node might have parameters for spreadsheet ID, sheet name, and so on. Many node parameters can accept not just static values but also **expressions** that reference data from previous nodes. n8n uses a syntax `{{$json["fieldName"]}}` (or similar) to allow you to map fields from prior node outputs into subsequent node inputs. This way, you can **dynamically pass data** between nodes (for instance, using an ID fetched by one node as input in the next node's API call) without hardcoding values.

- **Expressions: Expressions** in n8n are snippets of JavaScript code (enclosed in `{{ ... }}`) that are evaluated at runtime to produce a value. They allow you to manipulate and combine data on the fly. Within expressions, you have access to contextual data like the current item's JSON (`$json`), variables like `$node` (to get data from other nodes), and utility functions. For example, an expression could format a date: `{{ new Date($json["timestamp"]).toLocaleString() }}` or combine strings: `{{ $json.firstName + " " + $json.lastName }}`. Expressions can be used in almost any input field of a node, providing a powerful way to transform data without writing a full code node. n8n provides many **built-in methods and variables** for expressions (such as `$now` for current time, `$jmespath` for JSON querying, etc.) [20]. This makes workflows very flexible and dynamic.

- **Executing Workflows:** There are two main ways to run a workflow: **manually** or **automatically**. In the editor, you can do manual test runs (using the "Execute Workflow" or "Test" button). This runs the workflow immediately with either sample data or manual trigger input. Manual executions are great for development and debugging – you can examine each node's output in

the execution preview. For automatic runs, you rely on triggers: e.g. a Time trigger will run on schedule, or a Webhook trigger will wait for an external call. Once active, the workflow will execute whenever its trigger condition is met [21]. n8n ensures that if multiple triggers happen (e.g. multiple webhook calls), each event will spawn a separate execution (they queue if needed). You can monitor executions in the **Executions list** in the UI, which shows all past workflow runs (if logging is enabled). n8n also supports **partial executions** (running a workflow from a specific node onward for testing) and **retrying failed executions** from the point of failure, which are useful during development [22] [23].

- **Error Handling:** Workflows can be designed to handle errors gracefully. By default, an unhandled error in any node will cause the workflow to stop execution (and mark it as failed). However, n8n offers options to change this behavior. On a **global level**, in workflow settings you can choose "continue on fail" to attempt to continue a workflow even if a node errors. On a **node level**, most nodes have an **"On Error" option** or **"Continue On Fail"** toggle in their settings [22] [23]. For example, you can set a node to *not* stop the workflow if it fails, and instead simply output an empty item or send the error to a special output (some nodes have a second output for errors). There is also an event-trigger node called **Error Trigger** which can start a separate workflow if any workflow fails. Using these mechanisms, you can implement error-handling patterns like try-catch flows: e.g. route errors to an error handling branch or workflow (logging the error, sending a notification, etc.) while the main workflow either stops or continues with fallback logic.

- **Importing/Exporting Workflows:** n8n allows you to easily share and reuse workflows via **JSON blueprints**. A workflow can be exported to a JSON format that captures all nodes, connections, and settings (credentials are referenced by name/ID but not included for security). In the Editor UI, you can export by clicking *Download* to get a `.json` file, or simply selecting all nodes and copying (`Ctrl+C`), which copies a JSON workflow representation to your clipboard [24]. You can import a workflow by pasting a JSON into the canvas (`Ctrl+V`) or using the *Import from File/URL* options [25]. This JSON is what we refer to as the **workflow blueprint**. It includes a top-level JSON object with keys like `"name"` (workflow name), `"nodes"` (array of node objects), `"connections"` (object describing how nodes connect), `"active"` (boolean), and optional `"settings"` and `"tags"`. Each node in the JSON has its parameters and connections listed. This blueprint format is what you would use if programmatically generating workflows or storing them in version control. (When using the REST API, the same JSON structure is used to create or update workflows.)

Hosting and Deployment Options

- **n8n Cloud (Hosted): n8n Cloud** is the official managed hosting option for n8n [26]. With n8n Cloud, the n8n instance runs in the cloud and is maintained by the n8n team. Key benefits of Cloud include: no installation or server management required, **no technical setup or maintenance**, continuous **uptime monitoring**, managed credential OAuth for integrations, automatic updates to the latest n8n versions with one-click, and a web-based admin interface [26]. Cloud is a good choice for users who don't want to self-host; it offers a free trial and then various paid plans for different usage levels [27]. (Note: n8n Cloud may not be available in all regions for legal reasons [28].)

- **Self-Hosted (Community Edition):** n8n can be self-hosted on your own infrastructure. The **Community Edition** of n8n is free and offers full core functionality [29]. You can install n8n on practically any platform – common methods include using **Docker** (Docker image), **npm** (Node.js

package), or deploying via **server guides** for services like AWS, DigitalOcean, Azure, etc. [30]. Self-hosting gives you more control (you manage the database, files, and environment) and allows customization (like installing community nodes or modifying the source). However, it requires that you handle maintenance, scaling, and security of the instance. n8n provides documentation for different deployment scenarios (Docker setup, configuring reverse proxies for webhooks, using environment variables for config, database migration, etc.). Important configuration for self-hosted instances is done via **environment variables** (for example, to set the base URL, turn on queue mode, set encryption keys for credentials, etc.). Basic self-host requires at least Node.js and a database, but the all-in-one Docker container is the easiest way to start quickly.

- **Enterprise Edition:** For self-host users with advanced needs, n8n offers an Enterprise version (a paid upgrade) which includes additional features: role-based access control, multiple user support with fine-grained permissions, SSO/SAML integration, advanced logging (audit logs), and priority support. Enterprise also supports **API key scopes** (limiting API keys to certain operations) and other enterprise-oriented capabilities [31]. The Enterprise edition is under a special license and requires a subscription. Community edition is free and open under the Sustainable Use License, whereas Enterprise adds on top of that for businesses. (Essentially, the Community version is fully functional for single-user or small team scenarios, and Enterprise is needed for large teams or embedding n8n in a commercial product with OEM licensing).

- **Embed n8n:** Another deployment option is **n8n Embed**, which allows you to integrate n8n's workflow engine and editor into your own product or platform. This is essentially a special licensing and configuration of n8n geared towards white-label integration. *Embed* enables you to **white-label n8n and embed the editor UI** into your application, so your end-users can build automations within your software under your branding [32]. This requires an **Embed license** (a commercial arrangement) and is intended for software companies who want to offer automation features natively. With n8n Embed, you get the building blocks to instantiate n8n inside your app (via iframe or direct integration) and control user management, etc., through the n8n REST API. The embed documentation provides guidance on how to deploy and configure n8n in this mode (for example, disabling certain features, using single-sign-on with your app, managing workflow data via API) [32]. *See the Embedding n8n section below for more on white-label customization.*

- **Scaling and High Availability:** For higher loads, you can scale a self-hosted n8n deployment by leveraging **queue mode** (as mentioned in architecture). In a container environment (e.g. Kubernetes), you might run multiple n8n worker replicas. n8n can also be configured with external databases (e.g. PostgreSQL) and external file storage for binaries (e.g. S3) for better performance and durability in distributed setups. Clustering n8n involves setting one instance as **main** (for receiving webhooks and scheduling) and others as **workers**. They all share the same database and Redis queue. Using a load balancer for webhook URLs and enabling sticky sessions (or using the built-in webhook delegation to main) ensures incoming events go to the right n8n instance [5] [4]. Detailed docs are available for scaling, covering topics like setting concurrency, autoscaling workers, and running multiple main instances for redundancy [33]. In addition, you should plan for backup and recovery of the database (to safeguard workflow definitions and execution logs) and consider using monitoring/alerting to keep an eye on your n8n instance's health.

Using Code: Expressions and Function Nodes

- **Low-Code, not No-Code:** While n8n emphasizes no-code integrations, it is designed to be **"low-code"**, meaning you can incorporate code when the need arises [2]. There are two primary mechanisms to inject custom code into n8n workflows: **Expressions** and the **Code node**.

- **Expressions:** As noted, expressions are inline snippets of JavaScript used within node parameter fields. They are enclosed in double curly braces `{{ }}`. Expressions run in a sandbox and can access current item data (`$json`), outputs of other nodes (`$node["Node Name"].json`), environment variables, and helper functions. Use expressions for simple transformations or dynamic values (e.g., constructing an email message with values from previous steps, performing a small calculation, or formatting text) [20]. Expressions are evaluated per item – if a node processes an array of items, the expression will be evaluated for each item separately, producing an array of results.

- **Code Node:** For more complex logic, n8n provides a special node called the **Code node** (previously known as Function node). The Code node lets you write arbitrary code to manipulate data or implement custom functionality that might be difficult to achieve with the preset nodes. Uniquely, n8n's Code node supports both **JavaScript and Python** runtime execution [34] [35]. By default, you can choose JavaScript (which runs in Node.js VM context) or Python (which runs using a Python interpreter under the hood). This means you could, for example, use Python libraries or do advanced data science tasks in the middle of your workflow, or simply use JavaScript for familiarity. In the editor, you select the language and then write the code in the provided editor. The Code node provides access to the incoming items via an array `items` variable and expects you to return an array of output items. For instance, a simple JS Code node might be:

```
// JavaScript Code node example
items.forEach(item => {
  item.json.sum = item.json.value1 + item.json.value2;
});
return items;
```

This would add a new field `sum` to each JSON item combining two input values. The Code node is powerful – you can perform loops, conditionals, and any logic. Keep in mind that running code is generally slower than native nodes (especially Python, which involves additional startup overhead [36]), so use it when necessary. Common use cases for the Code node include: data transformation that isn't handled by built-in nodes (e.g. complex date parsing), calling external libraries or performing calculations, or even making custom API calls (though the HTTP Request node is usually preferred for calls).

- **External Libraries:** In JavaScript Code nodes, you have access to the Node.js standard library and some globally available utilities provided by n8n. However, by default you cannot `require` arbitrary npm packages unless they are pre-installed in the n8n instance. Advanced users who self-host might extend the Code node by preloading certain npm modules (via custom Docker build or the `NODE_FUNCTION_ALLOW_EXTERNAL` setting to allow specific imports). In Python Code nodes, similarly, only standard library is available by default. The documentation provides guidance if you need to use external libraries – typically one would use the Function node primarily for relatively contained logic due to these limitations.

- **Built-in Helpers:** n8n offers some built-in helper functions for use in Code nodes and expressions. Examples include: `$executeWebhook()` to call another webhook, `$itemIndex` to get the index of the current item, and **data transformation functions** like `JSON.stringify()` or specific ones documented under n8n's **Data Transformation** docs. Additionally, in expressions and code, you can use **Luxon** (a date library) via the global `DateTime` object, and certain lodash-like utilities for convenience, as mentioned in the **Built-in methods and variables** reference [20].

- **When to use Code:** Use an expression for simple one-liners or field mappings. Use a Code node when you need multi-step calculations or when logic becomes too cumbersome to express in a single expression. The key advantage is that you can overcome limitations of built-in nodes. For example, if no node exists for a certain API endpoint, you might use an HTTP Request node; but if the data needs heavy processing after fetching, a Code node can handle it. Another scenario is implementing loops or aggregations: e.g., if you want to combine data from multiple items into one, a Code node can accumulate results (since normal nodes process each item independently). In summary, Code nodes add ultimate flexibility to n8n, essentially letting you do anything that a custom script could do, but within your workflow context.

Embedding n8n and White-Labeling

- **n8n Embed Overview:** *Embedding* n8n refers to integrating the n8n workflow editor and engine into another application, typically under a custom branding (white-label). This is a feature for product developers who want to offer automation capabilities in their own software. **n8n Embed allows you to white-label n8n and build it into your own product** [32]. For example, a SaaS platform could embed n8n so that its users can create automation workflows in the SaaS's UI, without knowing it's n8n under the hood. To use n8n in this way, you need to obtain an embed license and typically work with the n8n team (this is a paid option).

- **How Embedding Works:** Under the hood, embedding n8n usually means running a modified n8n instance (or cluster) and using the **n8n front-end in an iframe or as a library** in your app. The embed documentation provides steps to set up prerequisites and deployment for an embedded instance (for example, enabling CORS, configuring authentication, etc. to integrate with your app's auth). The n8n editor UI can then be rendered in your app, and you can interact with n8n via its REST API to manage workflows, users, credentials, etc. [32]. Essentially, your application becomes a client of n8n's API, controlling it in the background, while your users see a seamless interface. There are features like the **Workflow Templates API** that you can use to load pre-built workflows for your users, and you can programmatically create workflows on behalf of users through the API.

- **White-Label Customization: White-labeling** means customizing n8n's appearance and some behaviors so it blends in with your product's branding. With an embed license, you get access to n8n's source code (or a method) to modify styling, logos, and even text strings. According to the docs, **white-labeling n8n involves changing parts of the frontend code (Vue.js components and CSS)** to apply your brand's theme [37]. Specifically, n8n's UI is built with a design system and an editor UI package; you would fork these packages and adjust things like colors, fonts, and logos to match your brand [37]. For instance, you can replace the n8n logo with your own logo, change the primary color scheme, and edit certain UI texts. The docs mention modifying SCSS token files for theme colors and providing alternative logo image files [38] [39]. You can also localize text (the White Labelling guide includes steps to customize or translate interface text)

[40]. After making these changes, you'd rebuild the n8n front-end. The result is an instance of n8n that looks and feels like a native part of your application.

- **Embedding Considerations:** When embedding, you typically will **disable certain n8n features** that don't apply in an embedded context. For example, user management might be handled by your app rather than n8n, so you'd configure n8n to trust an external JWT or SSO. You might restrict the nodes available (perhaps only enabling certain integrations relevant to your app). The embed documentation covers configuration options to tailor the experience [32]. Security is another consideration: if end-users are building workflows in your app via n8n, you need to sandbox their access (so they can't, for example, perform malicious actions on your infrastructure). n8n's role-based access control (enterprise feature) can help here, as you could assign limited roles to embedded end-users.

- **Conclusion:** n8n Embed is a powerful way to add automation features to an existing app without building from scratch. It essentially leverages all of n8n's capabilities under your own UX. Many companies use this to provide integration hubs or workflow automation to their customers. However, it requires a good amount of setup (licenses, custom building, and maintenance of the forked code). The official docs and n8n team provide guidance and support for partners using the embed route.

Advanced AI Integration in Workflows

- **Overview of AI Features:** n8n has introduced features to integrate **AI and LLM (Large Language Model) capabilities** into workflows, under the banner of **Advanced AI** (available from n8n v1.19.4+ on both cloud and self-hosted editions) [41] [42]. These features enable users to build workflows that can, for example, process natural language, interact with language models (like OpenAI's GPT-4), or integrate with vector databases and AI APIs. In essence, n8n acts as an orchestration layer where you can chain AI actions with other nodes (for data retrieval, sending results, etc.), effectively building AI-powered automation.

- **LangChain Integration:** n8n's approach to AI integration leverages **LangChain**, a popular framework for building applications with language models. n8n includes specialized **LangChain nodes** and concepts. For instance, n8n provides a **LangChain Code** node which allows constructing chains/agents using LangChain directly within n8n, and various support for memory, tools, etc. (Under Advanced AI > LangChain in n8n docs, you'll find that n8n wraps LangChain functionality). This means you can create a workflow where the AI can do things like: take a user query, use a language model to interpret it, maybe call a vector search to retrieve relevant info, and then respond or perform actions based on that. n8n essentially becomes a canvas not just for connecting APIs, but also for stringing together AI reasoning steps with your business logic.

- **AI Nodes and Capabilities:** Some capabilities provided:

- **LLM Nodes**: There are nodes to interact with large language model APIs (for example, an OpenAI node for chat or completion, or other provider integrations). These nodes send prompts to the AI service and return the response. You can incorporate those responses into the workflow (e.g., generate text then email it, or parse it, etc.).
- **Vector Store Nodes**: n8n has integration for vector databases (like Pinecone, Milvus, Qdrant, etc.) which are used in semantic search and retrieval augmented generation. You could, for

instance, take documents, split them, create embeddings, store in a vector DB via n8n nodes, and later query them based on an AI prompt.
- **Chat-oriented features**: n8n introduced a **Chat Trigger** node and a **Chatbot UI widget** to simplify creating chatbots [43] . The **Chat Trigger** node can kick off a workflow when a user sends a message in a chat interface (for example, a chat widget on a website or in an app) [43] . The **Chatbot widget** is a frontend component that can be embedded to allow user interaction with an AI-powered workflow [44] . Together, these allow building custom chatbots: the Chat Trigger provides the user's input to the workflow, the workflow can process it (perhaps consult an LLM, lookup info, etc.), and then respond (the response can be sent back to the chat widget).

- **Agents/Tools**: With LangChain, n8n can implement agent paradigms where an AI dynamically decides to use certain tools (which can be other n8n nodes) to fulfill a task. For example, an AI agent in n8n might choose to use a "Google Sheets" node as a tool to look up some data, or use an HTTP node to fetch information, if prompted accordingly. This is quite advanced – essentially n8n provides an environment for AI agents with controlled tool access. The docs discuss concepts like what an agent is, what memory is (for maintaining context in multi-turn conversations), and what tools are in the LangChain context [45] .

- **AI Workflow Examples:** Some example use cases that n8n's Advanced AI features enable:

- Building a **custom chatbot** (for a website or product) that can answer questions or perform tasks. Using the Chat Trigger and AI nodes, you can route chat messages to an OpenAI model, and then possibly to other nodes (if the query is about user data, fetch from DB, etc.) and then return an answer. The chatbot could be context-aware if you integrate memory (chat history) or vector search for knowledge base articles.
- **Document processing** automation: e.g., a workflow that takes a PDF or email, uses AI to extract meaning or summary, and then routes that output (maybe storing it or sending a notification). N8n can connect to document sources (email, Drive, etc.), then use an AI node to summarize or classify the text.
- **AI-assisted data enrichment**: e.g., fetching data from an API and asking the AI to analyze or categorize it before proceeding.

- **Human-in-the-loop flows**: n8n allows combining AI with human decision points. For instance, after an AI generates a draft email response, you could use n8n to send it to a human for approval (perhaps via a manual trigger or an interface), then continue the workflow.

- **Starter Kit:** The n8n docs mention a *Self-hosted AI Starter Kit* [46] which is a pre-packaged set of tools (n8n plus some AI components like, say, an open-source LLM or vector DB) to quickly get started building AI workflows on your own infrastructure. This underlines how n8n is positioning as an easy way to orchestrate AI: you bring your API keys or open-source models, and let n8n coordinate the logic.

- **Important Notes:** Using AI features in n8n still requires understanding the external AI services. For example, if using OpenAI's API, you must provide your API key via n8n credentials, and you will be subject to OpenAI's rate limits and costs. Also, AI nodes may introduce latency (calls to AI models can be relatively slow), so workflows that use them might run longer. n8n's execution engine can handle long-running workflows, but you should be mindful of timeouts especially if you have synchronous responses (like an HTTP trigger waiting for a reply from an AI process). For heavy use, consider using queue mode so that workers handle AI calls asynchronously and scale out if needed.

In summary, n8n's Advanced AI features allow seamless combination of LLM intelligence with the wide range of integrations n8n supports. This means you can build workflows that not only connect services but also *think* (to an extent) and make decisions or generate content, bringing powerful automation possibilities with relatively little effort [42].

n8n REST API (Public API) and Programmatic Access

- **Public REST API:** n8n provides a **Public REST API** that allows you to perform most operations programmatically, similar to what you can do via the web interface [47]. This API is useful for automating the management of n8n itself – for example, you can create, update, or delete workflows via API calls, execute workflows, manage credentials, read execution logs, manage users (in enterprise), etc., all through HTTP endpoints. In essence, if you want to integrate n8n with CI/CD or control it from an external script or system, this API is the way to do it [47]. (One common scenario: using the API to import/export workflows as part of version control or deployment pipeline, or building an admin panel on top of n8n's capabilities.)

- **Enabling API & Security:** The public API is available on both n8n cloud and self-hosted, but note that on n8n cloud you must be on a paid plan (it's not enabled during the free trial) [48]. On self-hosted, it's enabled by default unless you deliberately disable it. If you are not using the API, it's recommended to **disable it for security** (to prevent unwanted access) by setting environment variable `N8N_PUBLIC_API_DISABLED=true` [49]. Likewise, the API comes with an interactive **API Playground** (Swagger UI at `<your-n8n-host>/api`), which you can disable via `N8N_PUBLIC_API_SWAGGERUI_DISABLED=true` if desired [50].

- **Authentication:** All API requests must be authenticated using an **API Key**. You can create API keys in the n8n UI: go to *Settings* > *n8n API* and generate a key (label and expiration can be set, and if you have enterprise, you can scope the key's permissions) [51] [52]. n8n uses API keys for auth – there's no separate OAuth or user token here. **Non-enterprise keys have full access** equivalent to your account (be cautious with these) [53]. Once you have an API key, you include it in request headers: `X-N8N-API-KEY: <your key>` on each call [52]. For example, using `curl` to fetch active workflows, you'd do:

```
curl -X GET '<N8N_URL>/api/v1/workflows?active=true' \
     -H 'X-N8N-API-KEY: <your-api-key>' \
     -H 'accept: application/json'
```

This would return a JSON list of all workflows where `"active": true` [54] [55]. The base URL for the API is your n8n host plus `/api/v1/`. (The version may increment in future, but as of now v1 is used.)

- **API Endpoints:** The API endpoints cover resources such as:
- **Workflows:** e.g. `GET /workflows` (list workflows), `POST /workflows` (create a new workflow from JSON), `GET /workflows/{id}` (get one), `PATCH /workflows/{id}` (update), `DELETE /workflows/{id}`. You can create workflows by sending the JSON blueprint in the request body. There are also endpoints to **activate/deactivate** a workflow (which might be a flag in the update call or a dedicated route).
- **Executions:** e.g. `GET /executions` (list past executions, supports query filters like workflowId or status), `GET /executions/{id}` (get details of a specific execution including error if failed and output), possibly `POST /executions/{id}/retry` to retry a failed execution (if enabled).

- **Credentials:** e.g. `GET /credentials` (list stored creds metadata), `POST /credentials` (create new credential; you provide the credential data which n8n will encrypt using its encryption key), `GET /credentials/{id}` (details), `DELETE /credentials/{id}`. When creating credentials via API, you must provide the data in the same format the UI would (usually a combination of credential type and fields).
- **Users & Management:** In an instance with user management (enterprise), endpoints exist to manage users, roles, etc.
- **Other entities:** Possibly endpoints for tags, for workflow sharing settings, etc., as reflected in the UI functionality.

The full API is documented in the **API reference** section of docs [56], and you can also explore it using the Swagger UI (API playground) in your n8n instance.

- **API Usage Examples:** Some practical examples of what you can do:
- *CI/CD for Workflows:* You might store workflow JSON in a Git repo. Using the API, you could script the deployment of workflows to different environments (dev/prod). For instance, a script could read a JSON file and `POST /workflows` to import it. Conversely, `GET /workflows/{id}` could retrieve the JSON to save in git.
- *Operate n8n from another app:* If you have a separate frontend and you want to list workflows or trigger them, you can call n8n's API. For example, to trigger a workflow execution manually, there isn't a direct "execute workflow now" endpoint in v1 API (workflow runs are normally trigger-based), but you can simulate it by calling the webhook endpoint of a workflow or using the `POST /workflows` to create a one-time execution. There is also an **external hook** to start executions via API in certain contexts (like using the *Webhook* node, or the *Trigger* nodes).
- *Monitoring and extracting logs:* Use `GET /executions?workflowId=<id>` to fetch execution history for a particular workflow, maybe to display stats or detect failures from an external monitoring system.

- *User and Credential provisioning:* In enterprise setups, you might use the API to programmatically create user accounts or credentials. For example, when a new team joins, you could auto-provision some default credentials (if you have a way to input their secrets securely).

- **API and Embed:** When embedding n8n, the REST API is a primary interface to control the embedded instance. For example, your application might create a new workflow for a user via the API when they sign up, or copy template workflows via API calls. The API keys in that case might be scoped or tied to your application's backend.

- **Note on Swagger UI:** The API Playground accessible in self-hosted instances (usually at `http://<host>:<port>/api/`) is a Swagger interface where you can input your API key and test endpoints in your browser. This is convenient for development and discovery of the API. It's recommended to disable it in production or restrict access, as mentioned, to reduce attack surface [49] [57].

In summary, the n8n public API opens up the possibility for **Infrastructure as Code for automations**, remote controlling of your n8n instance, and embedding scenarios. It allows external programs to create or modify workflows and other entities, enabling a high degree of automation and integration of n8n itself into larger systems. Always keep your API key secure, and leverage the API's power responsibly (e.g., avoid infinite loops of triggering workflows via API from workflows). If not needed, disable it to harden your instance [49]. Otherwise, enjoy the flexibility of treating n8n not just as a GUI tool, but also as an automation engine accessible via REST.

Common Workflow Patterns and Best Practices

When designing workflows in n8n, certain **patterns** and best practices can help solve common automation scenarios:

- **Sequential Processing:** The most straightforward pattern is a simple sequence of nodes (trigger -> step1 -> step2 -> ... -> end). This is used when each step's output feeds into the next step's input. n8n handles sequential dependencies naturally via connections. Ensure the data you need is passed along; you can use the *Set* node or *Move Binary Data* node to adjust what each step passes forward if needed.

- **Conditional Branching:** Use the **If** node to create two paths (true/false) based on a condition (e.g., "if status == success then..."). Only one branch will execute depending on the condition. For multi-branch scenarios (more than two outcomes), the **Switch** node is useful; you can have multiple cases and an optional default path. This pattern is essential for workflows that need to handle different data differently, or skip certain actions based on a criteria (e.g., if an order value > X, route to high-value path, else normal path). Under the hood, these nodes do not duplicate data automatically; they simply forward items to one output or another. If a case isn't met, items might go to a "null" path (not forwarded). Design your workflow so that each item ultimately goes somewhere (or intentionally gets filtered out).

- **Parallel Execution:** n8n can execute branches in parallel when a node has multiple outgoing connections from the same output. For instance, you might have one node that feeds into two different processing chains (imagine a single webhook trigger connecting to both a logging node and a main processing node). In n8n, if you draw two connections out of one node, it will **fork** the data – each next node receives the same input concurrently. This is used for things like: do some processing and at the same time notify someone. Keep in mind that these branches run simultaneously in the single-threaded event loop – effectively interleaved, but conceptually parallel. If they later need to join back, you can use **Merge** node.

- **Merging and Aggregation:** The **Merge** node can take two inputs (Input 1 and Input 2) and combine them in various ways: you can simply concatenate data (merge by index or by key), or wait for both branches to finish and then output one combined result. One common use is *Wait All*: if you set Merge node to wait for both connections, it will pause until it has received data from both branches, then continue. This is useful if you had parallel branches and you need to synchronize before moving on. Another mode is *Merge by Key* which can join items from two inputs based on an ID field (similar to a database join). Use merge patterns when dealing with parallel branches or when re-combining split data. Note that merging by index will pair items 1-to-1 from two inputs; merging by key requires data sorted by keys.

- **Looping (Iteration):** Although each node iterates over items, sometimes you need a more complex loop (e.g., repeat a sequence of nodes until a condition is met). n8n doesn't have a direct "loop" node, but you can achieve loops by using the **Execute Workflow** node or flow logic. One trick is to use a combination of an **IF** node and a **Recycle**: for example, have an IF that checks a condition, and on the false branch, connect back to some prior node – effectively creating a loop. However, you must be careful to avoid infinite loops. Another approach is using a **Function (Code) node** to manually loop and call other nodes (though that can be advanced). In many cases, because n8n processes arrays, you might not need an explicit loop – e.g., if you want to send a message to 10 users, you can just have 10 items and connect to a single Email node (it will send 10 emails). But for scenarios like "retry 3 times until success" or "loop until

condition X is true", you often use a counter variable (perhaps stored in workflow static data or within item data) and an IF to decide whether to loop again. n8n's upcoming features might include native looping nodes, but as of now, these patterns work.

- **Error Handling Pattern:** As discussed earlier, one pattern is to have a separate workflow that is triggered by the Error Trigger node whenever any workflow fails. This "error workflow" can take the error details and do something (like send an alert or create an incident). Within a workflow, if you expect a node might fail but you want to handle that gracefully, you can enable **Continue On Fail** for that node – then the workflow won't stop, and you can check the output. Many nodes output an "error" field or route to a second output when continue-on-fail is used or on certain nodes like the HTTP Request node (which has a first output for 2xx responses and second output for errors or non-2xx). Designing flows that account for failures will make your automation more robust.

- **Reusable Workflows (Subworkflows):** n8n supports calling one workflow from another using the **Execute Workflow** node. This is a powerful pattern for reusability. If you have a common sequence of steps that you need in several workflows (for example, a standard data cleanup or notification routine), you can encapsulate that in its own workflow and then call it from various parent workflows. The Execute Workflow node allows passing input data to the subworkflow and receiving its output. This pattern also helps in simplifying complex workflows by breaking them into smaller, more manageable pieces. It's akin to function calls in programming. Be aware that recursive calls (workflow A calls workflow A) could cause loops – n8n typically prevents direct recursion.

- **Using Global Variables:** n8n has a concept of workflow static data (data that persists between executions of a workflow) and also environment variables (via expressions like `$env`). If you need to maintain state across runs (e.g., last run timestamp, or an incremental counter), you can use the static data (accessible in Code node via `this.workflowStaticData`). This can be a pattern for things like polling implementations or ensuring idempotency. However, static data is not easily accessible to other workflows or via API (except through the Execute Workflow to call a getter). Another "variable" pattern is simply using a dedicated credential or data store node (like Data Store node) to keep values. For example, an **Interval** trigger that runs every hour could store the last processed record ID in a file or in the built-in Data Store, then next run read it to know where it left off.

- **Blueprint Pattern (Infrastructure as Code):** If you treat workflow JSON exports as "blueprints", you might maintain these in a Git repository and even generate them via code. Some advanced users use the API or CLI to import these automatically. The pattern here is to separate development (designing workflows in a staging n8n or in JSON) from production (running in a prod n8n). Using the JSON format, you can systematically search/replace IDs (for example, adjusting credential IDs between environments) and promote workflows through different stages. Since the JSON is quite structured, you could even templatize parts of it for automation (though that requires good knowledge of the schema).

- **Optimization Pattern:** If performance is a concern (e.g., processing thousands of items), consider using the **Batching** option in node settings (under *Settings* > *Batching* for some nodes) [58]. This allows a node to process items in chunks (like 50 at a time) instead of all at once, which can prevent memory spikes. Another pattern is to offload heavy data handling to an external system: for instance, instead of having n8n sort or aggregate 10k records, use a database node or an external script to do that, and let n8n orchestrate the high-level flow.

- **Using Comments/Notes:** n8n allows adding **Sticky Notes** or notes on nodes. A best practice pattern is to annotate your workflow with notes explaining what different sections do. This is helpful for team collaboration or when you revisit a flow after some time. Similarly, naming your nodes descriptively (you can rename any node) is important – instead of "Function Item1", call it "Calculate Discount" etc. These habits make complex workflows understandable.

In conclusion, n8n is flexible and many patterns emerge from combining its primitives. As you design, remember that each node processes items independently and in parallel (per node). Use that to your advantage. Also, test edge cases: e.g., how does your workflow behave when an empty result comes from a node (zero items)? Using the Sandbox (manual run with test data) can help simulate different scenarios. The **community forums** and documentation have more examples and use-case guides if you're looking for a pattern for a specific scenario. With these building blocks – triggers, actions, logic nodes, and a bit of code – you can accomplish a vast array of automation tasks in n8n, from simple to highly sophisticated workflows. Keep iterating and refining your blueprints, and consider source-controlling them for reliability. Happy automating!

---

1 What are vector databases? | n8n Docs
https://docs.n8n.io/advanced-ai/examples/understand-vector-databases/

2 19 20 34 Code in n8n Documentation and Guides | n8n Docs
https://docs.n8n.io/code/

3 7 8 Database structure | n8n Docs
https://docs.n8n.io/hosting/architecture/database-structure/

4 5 6 33 Configuring queue mode | n8n Docs
https://docs.n8n.io/hosting/scaling/queue-mode/

9 21 Create and run | n8n Docs
https://docs.n8n.io/workflows/create/

10 12 13 22 23 58 Nodes | n8n Docs
https://docs.n8n.io/workflows/components/nodes/

11 n8n Integrations Documentation and Guides | n8n Docs
https://docs.n8n.io/integrations/

14 16 Credentials | n8n Docs
https://docs.n8n.io/credentials/

15 24 25 Exporting and importing workflows | n8n Docs
https://docs.n8n.io/courses/level-one/chapter-6/

17 18 Understanding the data structure | n8n Docs
https://docs.n8n.io/courses/level-two/chapter-1/

26 28 Overview | n8n Docs
https://docs.n8n.io/manage-cloud/overview/

27 30 32 Choose your n8n | n8n Docs
https://docs.n8n.io/choose-n8n/

29 46 n8n Hosting Documentation and Guides | n8n Docs
https://docs.n8n.io/hosting/

31 48 51 52 53 54 55 Authentication | n8n Docs
https://docs.n8n.io/api/authentication/

35 36 Using the Code node | n8n Docs
https://docs.n8n.io/code/code-node/

37 38 39 40 White labelling | n8n Docs
https://docs.n8n.io/embed/white-labelling/

41 42 43 44 45 n8n Advanced AI Documentation and Guides | n8n Docs
https://docs.n8n.io/advanced-ai/

47 49 50 57 Disable the public REST API | n8n Docs
https://docs.n8n.io/hosting/securing/disable-public-api/

56 Introducing the n8n public API – n8n Blog
https://blog.n8n.io/introducing-the-n8n-public-api/