

Workflows

A workflow is a connection of nodes connected together to automate a process.

- [Create](#) a workflow.
- Use [Workflow templates](#) to help you get started.
- Learn about the key [components](#) of an automation in n8n.
- Debug using the [Executions](#) list.
- [Share](#) workflows between users.

If it's your first time building a workflow, you may want to use the [quickstart guides](#) to quickly try out n8n features.

Create a workflow#

A workflow is a collection of nodes connected together to automate a process. You build workflows on the workflow canvas.

Create a workflow#

1. On the **Workflows** list, select **Add Workflow**.
2. Get started by adding a trigger node: select **Add first step...**

If it's your first time building a workflow, you may want to use the [quickstart guides](#) to quickly try out n8n features.

Run workflows manually#

You may need to run your workflow manually when building and testing, or if your workflow doesn't have a trigger node.

To run manually, select **Test Workflow**.

Run workflows automatically#

All new workflows are inactive by default.

You need to activate workflows that start with a trigger node or Webhook node so that they can run automatically. When a workflow is inactive, you must run it manually.

To activate or deactivate your workflow, open your workflow and toggle **Inactive / Active**.

Once a workflow is active, it runs whenever its trigger conditions are met.

Workflow templates#

When creating a new workflow, you can choose whether to start with an empty workflow, or use an existing template.

Templates provide:

- Help getting started: n8n might already have a template that does what you need.
- Examples of what you can build
- Best practices for creating your own workflows

Access templates#

Select  **Templates** to view the templates library.

If you use n8n's template library, this takes you to browse [Workflows on the n8n website](#). If you use a custom library provided by your organization, you'll be able to search and browse the templates within the app.

Add your workflow to the n8n library#

You can submit your workflows to n8n's template library.

n8n is working on a creator program, and developing a marketplace of templates. This is an ongoing project, and details are likely to change.

Refer to [n8n Creator hub](#) for information on how to submit templates and become a creator.

Self-hosted n8n: Disable templates#

In your environment variables, set `N8N_TEMPLATES_ENABLED` to false.

Self-hosted n8n: Use your own library#

In your environment variables, set `N8N_TEMPLATES_HOST` to the base URL of your API.

Your API must provide the same endpoints and data structure as n8n's.

The endpoints are:

Method	Path
GET	/templates/workflows/<id>
GET	/templates/workflows
GET	/templates/collections/<id>
GET	/templates/collections
GET	/templates/categories

GET /health

To learn about the data structure, try out n8n's API endpoints:

<https://api.n8n.io/templates/categories>

<https://api.n8n.io/templates/collections>

<https://api.n8n.io/templates/workflows>

<https://api.n8n.io/health>

You can also contact us for more support.

Workflow components#

This section contains:

- [Nodes](#): integrations and operations.
- [Connections](#): node connectors.
- [Sticky notes](#): document your workflows.

Nodes#

Nodes are the key building blocks of a workflow. They perform a range of actions, including:

- Starting the workflow.
- Fetching and sending data.
- Processing and manipulating data.

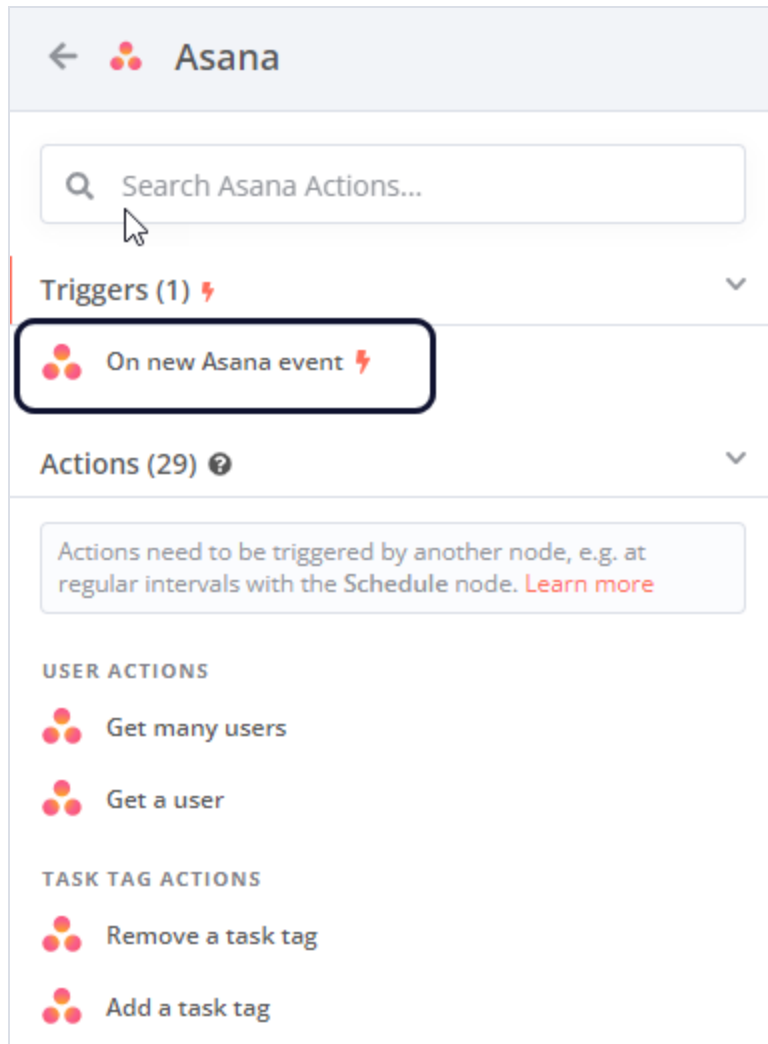
n8n provides a collection of built-in nodes, as well as the ability to create your own nodes. Refer to:

- [Built-in integrations](#) to browse the node library.
- [Community nodes](#) for guidance on finding and installing community-created nodes.
- [Creating nodes](#) to start building your own nodes.

Add a node to your workflow#


Add a node to an empty workflow#

1. Select **Add first step**. n8n opens the nodes panel, where you can search or browse trigger nodes.
2. Select the trigger you want to use.
3. **Choose the correct app event**
4. If you select **On App Event**, n8n shows a list of all the supported services. This allows you to browse n8n's integrations and trigger a workflow in response to an event in your chosen service. However, not all integrations have triggers. To see which ones you can use as a trigger, select the node. If a trigger is available, you'll see it at the top of the available operations list.
5. For example, this is the trigger for Asana:



6.


Add a node to an existing workflow#

Select the **Add node**  connector. n8n opens the nodes panel, where you can search or browse all nodes.

Node operations: Triggers and Actions#



When you add a node to a workflow, n8n displays a list of available operations. An operation is something a node does, such as getting or sending data.

There are two types of operation:

- Triggers start a workflow in response to specific events or conditions in your services. When you select a Trigger, n8n adds a trigger node to your workflow, with the Trigger operation you chose pre-selected. When you search for a node in n8n, Trigger operations have a bolt icon .
- Actions are operations that represent specific tasks or actions within a workflow, allowing you to manipulate data, perform operations on external systems, and trigger events in other systems as part of your workflows. When you select an Action, n8n adds a node to your workflow, with the Action operation you chose pre-selected.

Node controls#

To view node controls, hover over the node on the canvas:

- **Play**  : run the node.
- **Node context menu**  : select node actions. Available actions:
 - Open node
 - Execute node
 - Rename node
 - Deactivate node
 - Pin node
 - Copy node
 - Duplicate node
 - Select all

- Clear selection
- Delete node

Node settings#

The node settings allow you to control node behaviors and add node notes.

There are four toggles. When active, they do the following:

- **Always Output Data:** the node returns an empty item even if the node returns no data during execution. Be careful setting this on IF nodes, as it could cause an infinite loop.
- **Execute Once:** the node executes once, with data from the first item it receives. It doesn't process any additional items.
- **Retry On Fail:** when an execution fails, the node reruns until it succeeds.
- **Continue On Fail:** the workflow continues even if the execution of the node fails. When this happens, the node passes along input data from previous nodes, so if you enable this setting, the workflow design must handle unexpected output data.

You can document your workflow using node notes:


- **Notes:** note to save with the node.
- **Display note in flow:** if active, n8n displays the note in the workflow as a subtitle.

Connections#


A connection establishes a link between nodes to route data through the workflow. A connection between two nodes passes data from one node's output to another node's input.



Create a connection#

To create a connection between two nodes, select the grey dot or **Add node**  on the right side of a node and slide the arrow to the grey rectangle on the left side of the following node.

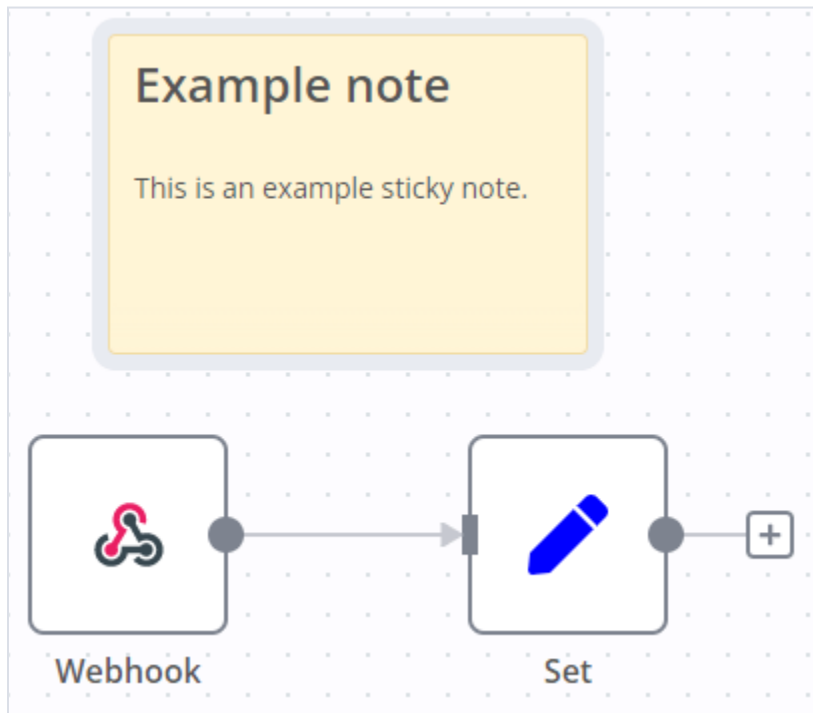
Delete a connection#

Hover over the connection, then select **Delete** .

Sticky Notes#

Sticky Notes allow you to annotate and comment on your workflows.

n8n recommends using Sticky Notes heavily, especially on template workflows, to help other users understand your workflow.



Create a Sticky Note#

Sticky Notes are a core node. To add a new Sticky Note:

1. Open the nodes panel.
2. Search for note.
3. Click the **Sticky Note** node. n8n adds a new Sticky Note to the canvas.


Edit a Sticky Note#

1. Double click the Sticky Note you want to edit.

2. Write your note. [This guide](#) explains how to format your text with Markdown. n8n uses [markdown-it](#), which implements the CommonMark specification.
3. Click away from the note, or press Esc, to stop editing.


Change the color#

To change the Sticky Note color:

1. Hover over the Sticky Note
2. Select **Change color** 

Sticky Note positioning#

You can:

- Drag a Sticky Note anywhere on the canvas.
- Drag Sticky Notes behind nodes. You can use this to visually group nodes.
- Resize Sticky Notes by hovering over the edge of the note and dragging to resize.
- Change the color: select **Options**  to open the color selector.

Writing in Markdown#

Sticky Notes support Markdown formatting. This section describes some common options.

```
1 The text in double asterisks will be **bold**
2 The text in single asterisks will be *italic*
3
4 Use # to indicate headings:
5 # This is a top-level heading
6 ## This is a sub-heading
7 ### This is a smaller sub-heading
8
9 You can add links:
10 [Example](https://example.com/)
11
12 Create lists with asterisks:
13
14 * Item one
15 * Item two
16
17 Or created ordered lists with numbers:
18
19 1. Item one
20 2. Item two
21
```

For a more detailed guide, refer to [CommonMark's help](#). n8n uses [markdown-it](#), which implements the CommonMark specification.

Make images full width#

You can force images to be 100% width of the sticky note by appending `#full-width` to the filename:

```
![Source example](https://<IMAGE-URL>/<IMAGE-NAME>.png#full-width)
```

```
1
```

Executions#

An execution is a single run of a workflow.

Execution modes#

There are two execution modes:

- **Manual:** run workflows manually when testing. Select **Test Workflow** to start a manual execution. You can do manual executions of active workflows, but n8n recommends keeping your workflow set to **Inactive** while developing and testing.
- **Production:** a production workflow is one that runs automatically. To enable this, set the workflow to **Active**.

Execution lists#

n8n provides two execution lists:

- [Workflow-level executions](#): this execution list shows the executions for a single workflow.
- [All executions](#): this list shows all executions for all your workflows.

n8n supports [adding custom data to executions](#).

Workflow-level executions list#

The **Executions** list in a workflow shows all executions for that workflow.

Deleted workflows

When you delete a workflow, n8n deletes its execution history as well. This means you can't view executions for deleted workflows.

Execution history and workflow history

Don't confuse the execution list with [Workflow history](#).

Executions are workflow runs. With the executions list, you can see previous runs of the current version of the workflow. You can copy previous executions into the editor to [Debug and re-run past executions](#) in your current workflow.

Workflow history is previous versions of the workflow: for example, a version with a different node, or different parameters set.

View executions for a single workflow#

In the workflow, select the **Executions** tab in the top menu. You can preview all executions of that workflow.

Filter executions#


You can filter the executions list.

1. In your workflow, select **Executions**.
2. Select **Filters**.
3. Enter your filters. You can filter by:

- **Status:** choose from **Failed**, **Running**, **Success**, or **Waiting**.
- **Execution start:** see executions that started in the given time.
- **Saved custom data:** this is data you create within the workflow using the Code node. Enter the key and value to filter. Refer to [Custom executions data](#) for information on adding custom data.
- **Feature availability**
 - Custom executions data is available on:
 - Cloud: Pro, Enterprise
 - Self-Hosted: Enterprise
 - Available in version 0.222.0 and above.

Retry failed workflows#

If your workflow execution fails, you can retry the execution. To retry a failed workflow:

1. Open the **Executions** list.
2. For the workflow execution you want to retry, select **Refresh** .
3. Select either of the following options to retry the execution:
 - **Retry with currently saved workflow:** Once you make changes to your workflow, you can select this option to execute the workflow with the previous execution data.
 - **Retry with original workflow:** If you want to retry the execution without making changes to your workflow, you can select this option to retry the execution with the previous execution data.

All executions list#

The **All executions** list shows all executions for all workflows you have access to.

Deleted workflows


When you delete a workflow, n8n deletes its execution history as well. This means you can't view executions for deleted workflows.

Browse executions#

Select **All executions**  to open the list.

Filter executions#



You can filter the executions list.

1. Select **All executions**  to open the list.
2. Select **Filters**.
3. Enter your filters. You can filter by:
 - **Workflows**: choose all workflows, or a specific workflow name.
 - **Status**: choose from **Failed**, **Running**, **Success**, or **Waiting**.
 - **Execution start**: see executions that started in the given time.
 - **Saved custom data**: this is data you create within the workflow using the Code node. Enter the key and value to filter. Refer to [Custom executions data](#) for information on adding custom data.
 - **Feature availability**

- Custom executions data is available on:
 - Cloud: Pro, Enterprise
 - Self-Hosted: Enterprise
- Available in version 0.222.0 and above.

Retry failed workflows#

If your workflow execution fails, you can retry the execution. To retry a failed workflow:

1. Select **All executions**  to open the list.
2. On the execution you want to retry, select **Retry execution** .
3. Select either of the following options to retry the execution:
 - **Retry with currently saved workflow:** Once you make changes to your workflow, you can select this option to execute the workflow with the previous execution data.
 - **Retry with original workflow:** If you want to retry the execution without making changes to your workflow, you can select this option to retry the execution with the previous execution data.

Load data from previous executions into your current workflow#

You can load data from a previous workflow back into the canvas. Refer to [Debug executions](#) for more information.

Tags#

Workflow tags allow you to label your workflows. You can then filter workflows by tag.

Tags are global. This means when you create a tag, it's available to all users on your n8n instance.

Add a tag to a workflow#

To add a tag to your workflow:

1. In your workflow, select **+ Add tag**.
2. Select an existing tag, or enter a new tag name.
3. Once you select a tag and click away from the tag modal, n8n displays the tag next to the workflow name.

You can add more than one tag.



Filter by tag#

When browsing the workflows on your instance, you can filter by tag.

1. On the **Workflows** page, select **Filters**.
2. Select **Tags**.
3. Select the tag or tags you want to filter by. n8n lists the workflows with that tag.

Manage tags#

You can edit existing tags. Instance owners can delete tags.

1. Select **Manage tags**. This is available from **Filters > Tags** on the **Workflows** page, or in the **+ Add tag** modal in your workflow.
2. Hover over the tag you want to change.
3. Select **Edit**  to rename it, or **Delete**  to delete it.


Global tags

Tags are global. If you edit or delete a tag, this affects all users of your n8n instance.

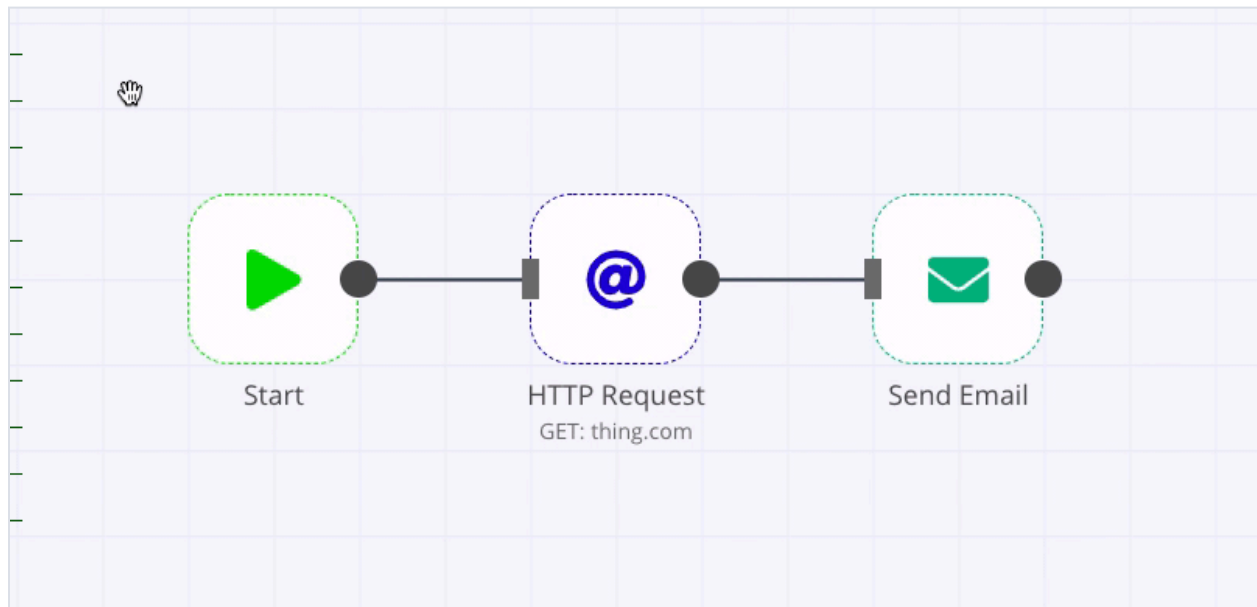
Export and import workflows#

n8n saves workflows in JSON format. You can export your workflows as JSON files or import JSON files into your n8n library.

You can export a workflow as a JSON file in two ways:

- **Download:** Select **Workflow menu**  > **Download**. This will download the workflow as a JSON file.
- **Copy-Paste:** Select all the workflow nodes in the Editor UI, copy them (Ctrl + c or cmd +c), then paste them (Ctrl + v or cmd + v) in your desired file.

To select all nodes, or a group of nodes, click and drag:




You can import JSON files as workflows by copying the JSON workflow to the clipboard (Ctrl + c or cmd +c) and paste it (Ctrl + v or cmd + v) into the Editor UI.

Workflow settings#

You can customize workflow behavior for individual workflows using workflow settings.

To open the settings:

1. Open your workflow.
2. Select the **Options**  menu.
3. Select **Settings**. n8n opens the **Workflow settings** modal.

The following settings are available:

- **Execution order:** choose the execution order for multi-branch workflows. **v0 (legacy)** executes the first node of each branch, then the second node of each branch, and so on. **v1 (recommended)**

executes each branch in turn, completing one branch before starting another. n8n orders the branches based on their position on the canvas, from topmost to bottommost. If two branches are at the same height, the leftmost branch executes first.

- **Error Workflow:** select a workflow to trigger if the current workflow fails. See [Error workflows](#) for more details.
- **This workflow can be called by:** choose whether other workflow can call this workflow. Requires [Workflow sharing](#).
- **Timezone:** sets the timezone for the workflow to use. The default timezone is EDT (New York). The timezone setting is important for the Schedule trigger node.
- **Save failed production executions:** whether n8n should save failed executions for active workflows.
- **Save successful production executions:** whether n8n should save successful executions for active workflows.
- **Save manual executions:** whether n8n should save executions for workflows started by the user in the editor.
- **Save execution progress:** whether n8n should save execution data for each node. If set to **Save**, the workflow resumes from where it stopped in case of an error. This might increase latency.
- **Timeout Workflow:** toggle to enable setting a duration after which n8n should cancel the current workflow execution.
 - **Timeout After:** Set the time in hours, minutes, and seconds after which the workflow should timeout. For n8n Cloud users n8n enforces a maximum available timeout for each plan.

Workflow history#

Feature availability

- Full workflow history is available on Enterprise Cloud and Enterprise Self-hosted.
- Versions from the last five days are available for Cloud Pro users.

Use workflow history to view and restore previous versions of your workflows.

Understand workflow history#

n8n creates a new version when you:

- Save your workflow.
- Restore an old version. n8n saves the latest version before restoring.
- Pull from a Git repository using [Source control](#). Note that n8n saves versions to the instance database, not to Git.

Workflow history and execution history


Don't confuse workflow history with the [Workflow-level executions list](#).

Executions are workflow runs. With the executions list, you can see previous runs of the current version of the workflow. You can copy previous executions into the editor to [Debug and re-run past executions](#) in your current workflow.

Workflow history is previous versions of the workflow: for example, a version with a different node, or different parameters set.


View workflow history#

To view a workflow's history:

1. Open the workflow.
2. Select **Workflow history** . n8n opens a menu showing the saved workflow versions, and a canvas with a preview of the selected version.

Restore or copy previous versions#

You can restore a previous workflow version, or make a copy of it:

1. On the version you want to restore or copy, select **Options** .
2. Choose what you want to do:
 - **Restore this version:** replace your current workflow with the selected version.
 - **Clone to new workflow:** create a new workflow based on the selected version.
 - **Open version in new tab:** open a second tab displaying the selected version. Use this to compare versions.
 - **Download:** download the version as JSON.

Find your workflow ID#

Your workflow ID is available in:

- The URL of the open workflow.
- The workflow settings title.

Credentials#

Credentials are private pieces of information issued by apps and services to authenticate you as a user and allow you to connect and share information between the app or service and the n8n node.

Access the credentials UI by opening the left menu and selecting **Credentials**. n8n lists credentials you created on the **My credentials** tab. The **All credentials** tab shows all credentials you can use, included credentials shared with you by other users.

- [Create and edit credentials](#).
- Learn about [credential sharing](#).
- Find information on setting up credentials for your services in the [credentials library](#).

Create and edit credentials#

You can get to the credential modal by either:

- Opening the left menu, then selecting **Credentials > Add Credential** and browsing for the service you want to connect to.
- Selecting **Create New** in the **Credential** dropdown in a node.

Once in the credential modal, enter the details required by your service. Refer to your service's page in the [credentials library](#) for guidance.

When you save a credential, n8n tests it to confirm it works.

Credentials library#

This section contains step-by-step information about authenticating the different nodes in n8n.

To learn more about creating, managing, and sharing credentials, refer to [Manage credentials](#).

User management#

User management in n8n allows you to invite people to work in your n8n instance. It includes:

- Login and password management
- Adding and removing users

- Two account types: owner and member

Privacy

The user management feature doesn't send personal information, such as email or username, to n8n.

Keyboard shortcuts and controls#

n8n provides keyboard shortcuts for some actions.

General#

- **Ctrl + Alt + n**: create new workflow
- **Ctrl + o**: open workflow
- **Ctrl + s**: save the current workflow
- **Ctrl + z**: undo
- **Ctrl + shift + z**: redo
- **Ctrl + Enter**: execute workflow

Canvas#

Move the canvas#

- **Ctrl + Left Mouse Button + drag**: move node view
- **Ctrl + Middle mouse button + drag**: move node view
- **Space + drag**: move node view

- **Middle mouse button** + drag: move node view
- Two fingers on a touch screen: move node view

Canvas zoom#

- **+** or **=**: zoom in
- **-** or **_**: zoom out
- **0**: reset zoom level
- **1**: zoom to fit workflow
- **Ctrl + Mouse wheel**: zoom in/out

Nodes on the canvas#

- **Ctrl + a**: select all nodes
- **Ctrl + v**: paste nodes
- **Shift + s**: add sticky note

With one or more nodes selected in canvas#

- **ArrowDown**: select sibling node below the current one
- **ArrowLeft**: select node left of the current one
- **ArrowRight**: select node right of the current one
- **ArrowUp**: select sibling node above the current one
- **Ctrl + c**: copy
- **Ctrl + x**: cut
- **D**: deactivate
- **Delete**: delete
- **Enter**: open
- **F2**: rename

- **P**: pin data in node. Refer to [Data pinning](#) for more information.
- **Shift + ArrowLeft**: select all nodes left of the current one
- **Shift + ArrowRight**: select all nodes right of the current one

Node panel#

- **Tab**: open the Node Panel
- **Enter**: insert selected node into workflow
- **Escape**: close Node panel

Node panel categories#

- **Enter**: insert node into workflow, collapse/expand category, open subcategory
- **ArrowRight**: expand category, open subcategory
- **ArrowLeft**: collapse category, close subcategory view

Within nodes#

- **=**: in an empty parameter input, this switches to expressions mode.

Flow logic#

n8n allows you to represent complex logic in your workflows.

This section covers:

- [Splitting with conditionals](#)
- [Merging data](#)
- [Looping](#)
- [Waiting](#)
- [Sub-workflows](#)
- [Error handling](#)
- [Execution order in multi-branch workflows](#)

Related sections#

You need some understanding of [Data](#) in n8n, including [Data structure](#) and [Data flow within nodes](#).

When building your logic, you'll use n8n's [Core nodes](#), including:

- Splitting: [IF](#) and [Switch](#).
- Merging: [Merge](#), [Compare Datasets](#), and [Code](#).
- Looping: [IF](#) and [Loop Over Items](#).
- Waiting: [Wait](#).
- Creating sub-workflows: [Execute Workflow](#) and [Execute Workflow Trigger](#).
- Error handling: [Stop And Error](#) and [Error Trigger](#).

-

Splitting workflows with conditional nodes#

Splitting uses the [IF](#) or [Switch](#) nodes. It turns a single-branch workflow into a multi-branch workflow. This is a key piece of representing complex logic in n8n.

Use the If node to split a workflow conditionally based on comparison operations.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [IF integrations](#) list.

Add conditions#

Add comparison conditions using the **Add Condition** filter. The available comparison operations vary for each data type.

String:

- exists
- doesn't exist
- is equal to

- isn't equal to
- contains
- doesn't contain
- starts with
- doesn't start with
- ends with
- doesn't end with
- matches regex
- doesn't match regex

Number:

- exists
- doesn't exist
- is equal to
- isn't equal to
- is greater than
- is less than
- is greater than or equal
- is less than or equal

Date & Time:

- exists
- doesn't exist
- is equal to
- isn't equal to
- is after
- is before
- is after or equal
- is before or equal

Boolean:

- exists
- doesn't exist
- is true
- is false
- is equal to
- isn't equal to

Array:

- exists
- doesn't exist
- is equal to
- isn't equal to
- contains
- doesn't contain
- length equal to
- length not equal to
- length greater than
- length less than
- length greater than or equal
- length less than or equal

Object:

- exists
- doesn't exist
- is empty
- isn't empty

AND / OR#

If you have more than one condition, you can choose either:

- **AND**: data must match all conditions to be true.
- **OR**: data only needs to match one of the conditions to be true.

Branch execution with If and Merge nodes#

0.236.0 and below

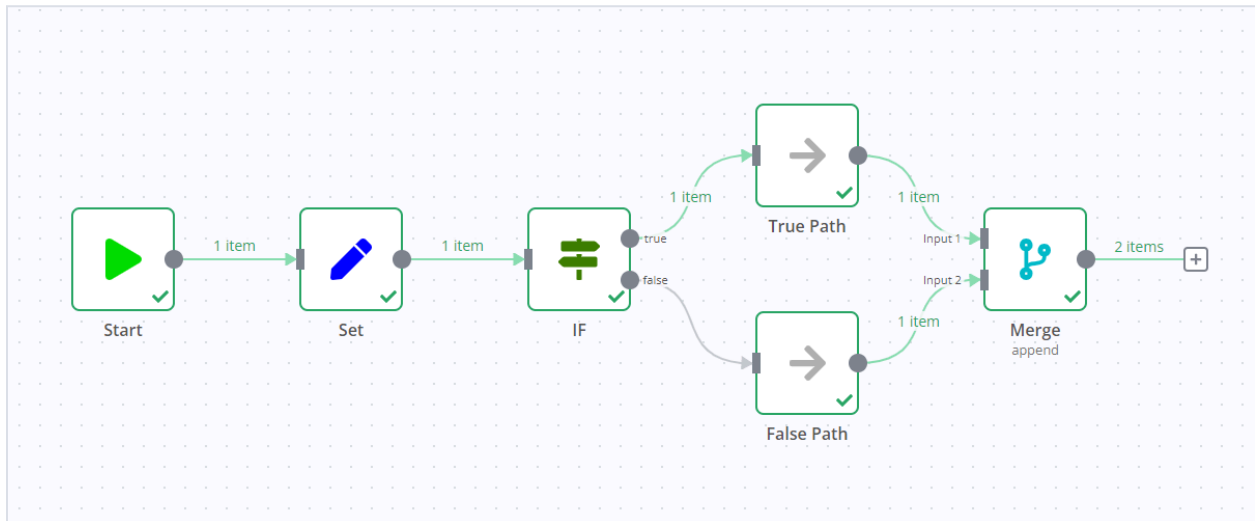
n8n removed this execution behavior in version 1.0. This section applies to workflows using the **v0 (legacy)** workflow execution order. By default, this is all workflows built before version 1.0. You can change the execution order in your [workflow settings](#).

If you add a Merge node to a workflow containing an If node, it can result in both output branches of the If node executing.

The Merge node is triggered by one branch, then goes and executes the other branch.

For example, in the screenshot below there's a workflow containing a Edit Fields node, If node, and Merge node. The standard If node behavior is to execute one branch (in the screenshot, this is the **true** output). However, due

to the Merge node, both branches execute, despite the If node not sending any data down the **false** branch.



Related resources#

View [example workflows and related content](#) on n8n's website.

Refer to [Splitting with conditionals](#) for more information on using conditionals to create complex logic in n8n.

If you need more than two conditional outputs, use the [Switch node](#).

Switch#

Use the Switch node to route a workflow conditionally based on comparison operations. It's similar to the [IF](#) node, but supports multiple output routes.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Switch integrations](#) page.

Node parameters#

- **Mode:** select whether to define the conditions as rules in the node, or as an expression, programmatically.
- **Routing Rules:** appears when you select **Mode > Rules**. Add comparison conditions using the dropdown. The available comparison operations vary for each data type.
- **Number of Outputs:** appears when you select **Mode > Expression**. Set how many outputs the node should have.
- **Output Index:** create an expression to determine which node output to send data to.

Node options#

- **Fallback Output:** choose how to route the workflow when none of the conditions match.
- **Ignore Case:** whether to ignore letter case.
- **Less Strict Type Validation:** enable this if you want n8n to attempt to convert value types based on the operator you choose.
- **Send data to all matching outputs:** enable this to send data to all outputs matching the conditions. When disabled, n8n sends data to the first output matching the conditions.

Related resources#

View [example workflows and related content](#) on n8n's website.

Refer to [Splitting with conditionals](#) for more information on using conditionals to create complex logic in n8n.

Merging data streams#

Merging allows you to bring multiple data streams together.

Merging data from different branches or nodes uses the [Merge](#) node. To merge data from multiple node executions, use the [Code](#) node.

To compare data, merge it, and output data streams depending on the comparison, use the [Compare Datasets](#) node.

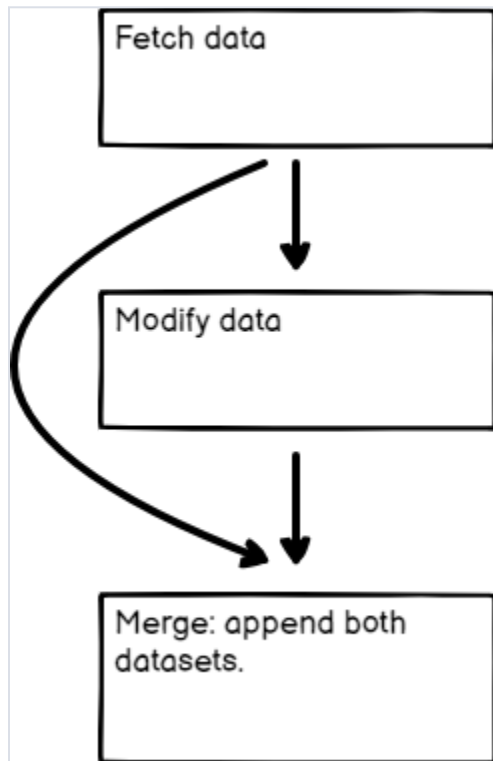
Merge data from different branches#

If your workflow [splits](#), you combine the separate branches back into one branch.

Here's an [example workflow](#) showing different types of merging: appending data sets, keeping only new items, and keeping only existing items. The [Merge node](#) documentation contains details on each of the merge operations.

Merge data from different nodes#

You can use the Merge node to combine data from two previous nodes, even if the workflow hasn't split into branches. This can be useful if you want to generate a single dataset from the data generated by multiple nodes.



Merge data from multiple node executions#

Use the Code node to merge data from multiple node executions. This is useful in some [Looping](#) scenarios.

Node executions and workflow executions

This section describes merging data from multiple node executions. This is when a node executes multiple times during a single workflow execution.

Refer to this [example workflow](#) using Loop Over Items and Wait to artificially create multiple executions.

Compare, merge, and split again#

The [Compare Datasets](#) node compares data streams before merging them. It outputs up to four different branches.

Refer to this [example workflow](#) for an example.

Merge#

Use the Merge node to combine data from two streams, once data of both streams is available.

Major changes in 0.194.0

This node was overhauled in n8n 0.194.0. This document reflects the latest version of the node. If you're using an older version of n8n, you can find the previous version of this document [here](#).

Examples and templates

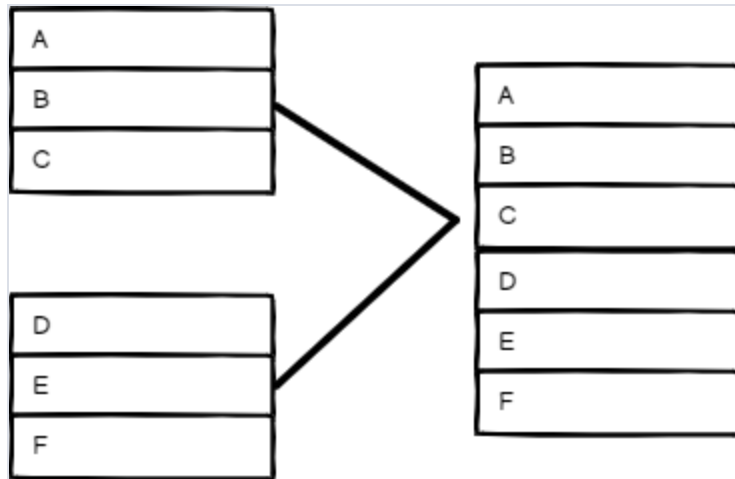
For usage examples and templates to help you get started, refer to n8n's [Merge integrations](#) page.

Merge mode#

You can specify how the Merge node should combine data from different branches. The following options are available:

Append#

Keep data from both inputs. The output contains items from Input 1, followed by all items from Input 2.



Combine#

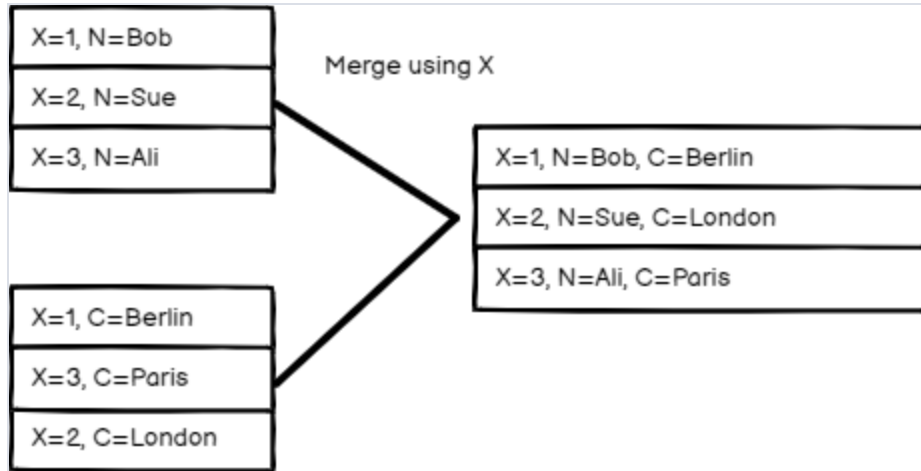
Combine data from both inputs. Choose a **Combination Mode** to control how n8n merges the data.

Merge by fields#

Compare items by field values. Enter the fields you want to compare in **Fields to Match**.

n8n's default behavior is to keep matching items. You can change this using the **Output Type** setting:

- Keep matches: merge items that match.
- Keep non-matches: merge items that don't match.
- Enrich Input 1: keep all data from Input 1, and add matching data from Input 2.
- Enrich Input 2: keep all data from Input 2, and add matching data from Input 1.



FIELD VALUE CLASHES#

If both items at an index have a field with the same name, this is a clash. For example, if all items in both Input 1 and Input 2 have a field named `language`, these fields clash. By default, n8n prioritizes Input 2, meaning if `language` has a value in Input 2, n8n uses that value when merging the items.

You can change this behavior:

1. Select **Add Option > Clash Handling**.
2. Choose which input to prioritize, or choose **Always Add Input Number to Field Names** to keep all fields and values, with the input number appended to the field name to show which input it came from.

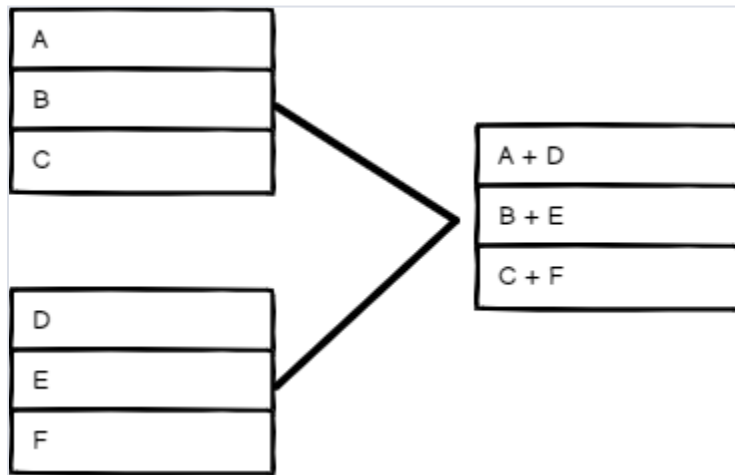
MULTIPLE MATCHES#

Matching by field can generate multiple matches if the inputs contain duplicate data. To handle this, select **Add Option > Multiple Matches**. Then choose:

- **Include All Matches:** output multiple items (one for each match).
- **Include First Match Only:** keep the first item, discard subsequent items.

Merge by position#

Combine items based on their order. The item at index 0 in Input 1 merges with the item at index 0 in Input 2, and so on.



INPUTS WITH DIFFERENT NUMBERS OF ITEMS#

If there are more items in one input than the other, the default behavior is to leave out the items without a match. Choose **Add Option > Include Any Unpaired Items** to keep the unmatched items.

FIELD VALUE CLASHES#

If both items at an index have a field with the same name, this is a clash. For example, if all items in both Input 1 and Input 2 have a field named `language`, these fields clash. By default, n8n prioritizes Input 2, meaning if `language` has a value in Input 2, n8n uses that value when merging the items.

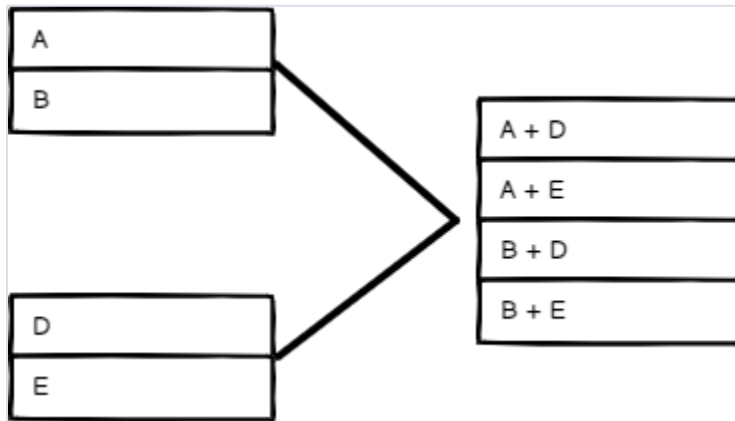
You can change this behavior:

1. Select **Add Option > Clash Handling**.
2. Choose which input to prioritize, or choose **Always Add Input Number to Field Names** to keep all fields and values, with the input

number appended to the field name to show which input it came from.

Multiplex#

Output all possible item combinations, while merging fields with the same name.



FIELD VALUE CLASHES#

If both items at an index have a field with the same name, this is a clash. For example, if all items in both Input 1 and Input 2 have a field named `language`, these fields clash. By default, n8n prioritizes Input 2, meaning if `language` has a value in Input 2, n8n uses that value when merging the items.

You can change this behavior:

1. Select **Add Option > Clash Handling**.
2. Choose which input to prioritize, or choose **Always Add Input Number to Field Names** to keep all fields and values, with the input number appended to the field name to show which input it came from.

Options#

When combining branches, you can set **Options**:

For all modes:

- **Clash handling**: choose how to merge when branches clash, or when there are sub-fields.
- **Fuzzy compare**: whether to tolerate type differences when comparing fields (enabled), or not (disabled, default). For example, when you enable this, n8n treats "3" and 3 as the same.

When merging by field:

- **Disable dot notation**: this prevents accessing child fields using `parent.child` in the field name.
- **Multiple matches**: choose how n8n handles multiple matches when comparing branches.

When merging by position:

Include Any Unpaired Items: choose whether to keep or discard unpaired items.

Choose branch#

Choose which input to keep. This option always waits until the data from both inputs is available. You can keep the data from Input 1 or Input 2, or you can output a single empty item. The node outputs the data from the chosen input, without changing it.

Merging branches with uneven numbers of items#

The items passed into Input 1 of the Merge node will take precedence. For example, if the Merge node receives five items in Input 1 and 10 items in Input 2, it only processes five items. The remaining five items from Input 2 aren't processed.

Branch execution with If and Merge nodes#

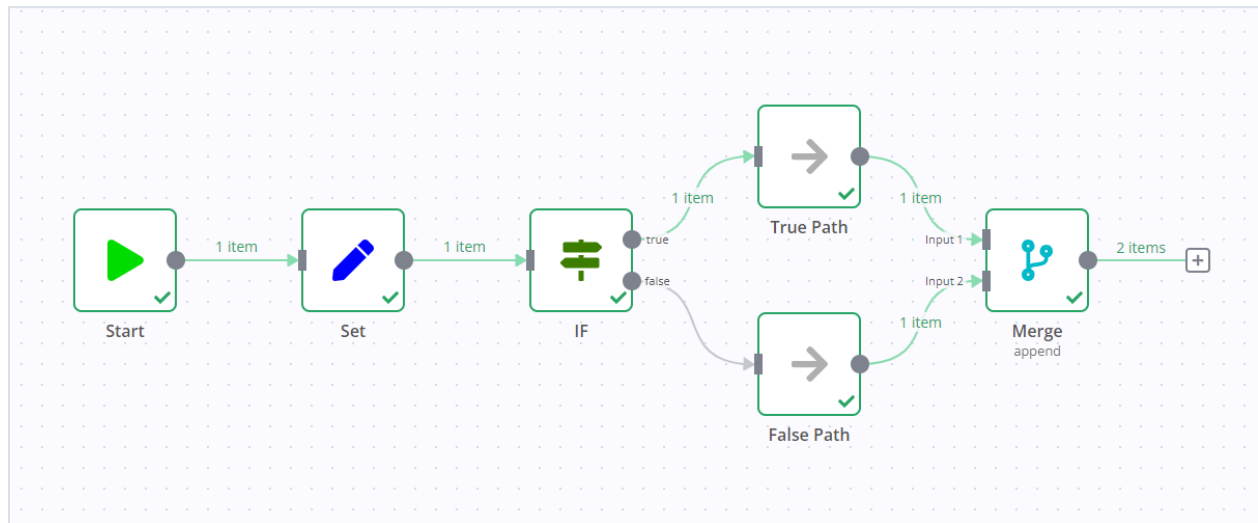
0.236.0 and below

n8n removed this execution behavior in version 1.0. This section applies to workflows using the **v0 (legacy)** workflow execution order. By default, this is all workflows built before version 1.0. You can change the execution order in your [workflow settings](#).

If you add a Merge node to a workflow containing an If node, it can result in both output branches of the If node executing.

The Merge node is triggered by one branch, then goes and executes the other branch.

For example, in the screenshot below there's a workflow containing a Edit Fields node, If node, and Merge node. The standard If node behavior is to execute one branch (in the screenshot, this is the **true** output). However, due to the Merge node, both branches execute, despite the If node not sending any data down the **false** branch.



Try it out: A step by step example#

Create a workflow with some example input data to try out the Merge node.

Set up sample data using the Code nodes#

1. Add a Code node to the canvas and connect it to the Start node.
2. Paste the following JavaScript code snippet in the **JavaScript Code** field:

```
1  return [  
2    {  
3      json: {  
4        name: 'Stefan',  
5        language: 'de',  
6      }  
7    },  
8    {  
9      json: {  
10       name: 'Jim',  
11       language: 'en',  
12     }  
13   },  
14   {  
15     json: {  
16       name: 'Hans',  
17       language: 'de',  
18     }  
19   }  
20 ];
```

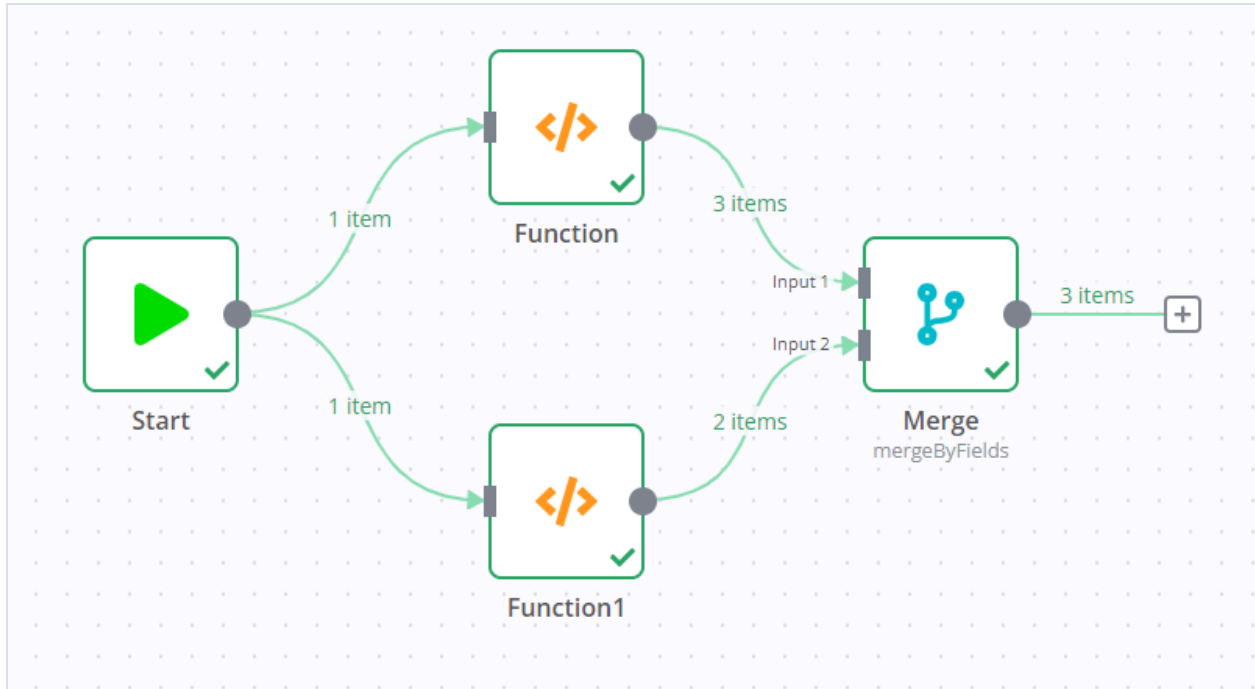
3. Add a second Code node, and connect it to the Start node.
4. Paste the following JavaScript code snippet in the **JavaScript Code** field:

```
1 return [  
2     {  
3     json: {  
4         greeting: 'Hello',  
5         language: 'en',  
6     }  
7 },  
8 {  
9     json: {  
10        greeting: 'Hallo',  
11        language: 'de',  
12    }  
13 }  
14 ];
```

Try out different merge modes#

Add the Merge node. Connect the first Code node to **Input 1**, and the second Code node to **Input 2**. Run the workflow to load data into the Merge node.

The final workflow should look like the following image.



Now try different options in **Mode** to see how it affects the output data.

Append#

Select **Mode** > **Append**, then select **Test step**.

Output data in table view:

name	language	greeting
Stefan	de	
Jim	en	
Hans	de	
	en	Hello
	de	Hallo

Merge by fields#

You can merge these two data inputs so that each person gets the correct greeting for their language.

1. Select **Mode > Merge By Fields**.
2. In both **Input 1 Field** and **Input 2 Field**, enter language. This tells n8n to combine the data by matching the values in the language field in each data set.
3. Select **Test step**.

Output in table view:

name	language	greeting
Stefan	de	Hallo
Jim	en	Hello
Hans	de	Hallo

Merge by position#

Select **Mode > Merge By Position**, then select **Test step**.

Default output in table view:

name	language	greeting
Stefan	en	Hello
Jim	de	Hallo

KEEP UNPAIRED ITEMS#

If you want to keep all items, select **Add Option > Include Any Unpaired Items**, then enable **Include Any Unpaired Items**.

Output with unpaired items in table view:

name	language	greeting
Stefan	en	Hello
Jim	de	Hallo
Hans	de	

Multiplex#

Select **Mode > Multiplex**, then select **Test step**.

Output in table view:

name	language	greeting
Stefan	en	Hello
Stefan	de	Hallo
Jim	en	Hello
Jim	de	Hallo
Hans	en	Hello
Hans	de	Hallo

Try it out: Load a workflow#

n8n provides an example workflow that demonstrates key Merge node concepts.

Go to [Joining different datasets](#) and select **Use workflow** to copy the example workflow. You can then paste it into your n8n instance.

Compare Datasets#

The Compare Datasets node helps you compare data from two input streams.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Compare Datasets integrations](#) page.

Usage#

1. Decide which fields to compare. In **Input A Field**, enter the name of the field you want to use from input stream A. In **Input B Field**, enter the name of the field you want to use from input stream B.
2. **Optional**: you can compare by multiple fields. Select **Add Fields to Match** to set up more comparisons.
3. Choose how to handle differences between the datasets. In **When There Are Differences**, select one of the following:
 - **Use Input A Version**
 - **Use Input B Version**
 - **Use a Mix of Versions**
 - **Include Both Versions**

Understand item comparison#

Item comparison is a two stage process:

1. n8n checks if the values of the fields you selected to compare match across both inputs.
2. If the fields to compare match, n8n then compares all fields within the items, to determine if the items are the same or different.

Options#

You can use additional options to refine your comparison or modify comparison behavior.

Select **Add Option**, then choose the option you want to use.

Fields to Skip Comparing#

Enter field names that you want to ignore.

For example, if you compare the two datasets below using **person.language** as the **Fields to Match**, n8n returns them as different. If you add **person.name** to **Fields to Skip Comparing**, n8n returns them as matching.

```

1      // Input 1
2      [
3          {
4              "person":
5              {
6                  "name":    "Stefan",
7                  "language": "de"
8              }
9          },
10         {
11             "person":
12             {
13                 "name":    "Jim",
14                 "language": "en"
15             }
16         },
17         {
18             "person":
19             {
20                 "name":    "Hans",
21                 "language": "de"
22             }
23         }
24     ]
25     // Input 2
26     [
27         {
28             "person":
29             {
30                 "name":    "Sara",
31                 "language": "de"
32             }
33         },
34         {
35             "person":
36             {
37                 "name":    "Jane",
38                 "language": "en"
39             }
40         },
41         {
            "person":

```

```

42         {
43             "name":    "Harriet",
44             "language": "de"
45         }
46     ]
47
48

```

Fuzzy Compare#

Whether to tolerate type differences when comparing fields (enabled), or not (disabled, default). For example, when you enable this, n8n treats "3" and 3 as the same.

Disable Dot Notation#

Whether to disallow referencing child fields using `parent.child` in the field name (enabled), or allow it (disabled, default).

Multiple Matches#

Choose how to handle duplicate data. The default is **Include All Matches**. You can choose **Include First Match Only**.

For example, given these two datasets:

```
1      // Input 1
2      [
3          {
4              "fruit": {
5                  "type": "apple",
6                  "color": "red"
7              },
8          },
9          {
10             "fruit": {
11                 "type": "apple",
12                 "color": "red"
13             },
14         },
15         {
16             "fruit": {
17                 "type": "banana",
18                 "color": "yellow"
19             }
20     ]
21     // Input 2
22     [
23         {
24             "fruit": {
25                 "type": "apple",
26                 "color": "red"
27             },
28         },
29         {
30             "fruit": {
31                 "type": "apple",
32                 "color": "red"
33             },
34         },
35         {
36             "fruit": {
37                 "type": "banana",
38                 "color": "yellow"
39             }
40     ]
41 ]
```


n8n returns three items, in the **Same Branch** tab. The data is the same in both branches.

If you select **Include First Match Only**, n8n returns two items, in the **Same Branch** tab. The data is the same in both branches, but n8n only returns the first occurrence of the matching "apple" items.

Understand the output#

There are four output options:

- **In A only Branch:** data that occurs only in the first input.
- **Same Branch:** data that's the same in both inputs.
- **Different Branch:** data that's different between inputs.
- **In B only Branch:** data that occurs only in the second output.

Related resources#

View [example workflows and related content](#) on n8n's website.

Code#

The Code node allows you to write custom JavaScript or Python and run it as a step in your workflow.

Coding in n8n

This page gives usage information about the Code node. For more guidance on coding in n8n, refer to the [Code](#) section. It includes:

- Reference documentation on [Built-in methods and variables](#)
- Guidance on [Handling dates](#) and [Querying JSON](#)
- A growing collection of examples in the [Cookbook](#)

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Code integrations](#) page.

Function and Function Item nodes

The Code node replaces the Function and Function Item nodes from version 0.198.0. If you're using an older version of n8n, you can still view the [Function node documentation](#) and [Function Item node documentation](#).

Usage#

How to use the Code node.

Choose a mode#

There are two modes:

- **Run Once for All Items:** this is the default. When your workflow runs, the code in the code node executes once, regardless of how many input items there are.
- **Run Once for Each Item:** choose this if you want your code to run for every input item.

JavaScript#

The Code node supports Node.js.

Supported JavaScript features#

The Code node supports:

- Promises. Instead of returning the items directly, you can return a promise which resolves accordingly.
- Writing to your browser console using `console.log`. This is useful for debugging and troubleshooting your workflows.

External libraries#

If you self-host n8n, you can import and use built-in and external npm modules in the Code node. To learn how to enable external modules, refer the [Configuration](#) guide.

Built-in methods and variables#

n8n provides built-in methods and variables for working with data and accessing n8n data. Refer to [Built-in methods and variables](#) for more information.

The syntax to use the built-in methods and variables is `$variableName` or `$methodName()`. Type `$` in the Code node or expressions editor to see a list of suggested methods and variables.

Python#

n8n added Python support in version 1.0. It doesn't include a Python executable. Instead, n8n provides Python support using [Pyodide](#), which is a port of CPython to WebAssembly. This limits the available Python packages to the [Packages included with Pyodide](#). n8n downloads the package automatically the first time you use it.

Slower than JavaScript

The Code node takes longer to process Python than JavaScript. This is due to the extra compilation steps.

Built-in methods and variables#

n8n provides built-in methods and variables for working with data and accessing n8n data. Refer to [Built-in methods and variables](#) for more information.

The syntax to use the built-in methods and variables is `_variableName` or `_methodName()`. Type `_` in the Code node to see a list of suggested methods and variables.

File system and HTTP requests#

You can't access the file system or make HTTP requests. Use the following nodes instead:

- [Read/Write File From Disk](#)
- [HTTP Request](#)

Coding in n8n#

There are two places where you can use code in n8n: the Code node and the expressions editor. When using either area, there are some key concepts you need to know, as well as some built-in methods and variables to help with common tasks.

Key concepts#

When working with the Code node, you need to understand the following concepts:

- [Data structure](#): understand the data you receive in the Code node, and requirements for outputting data from the node.
- [Item linking](#): learn how data items work, and how to link to items from previous nodes. You need to handle item linking in your code when the number of input and output items doesn't match.

Built-in methods and variables#

n8n includes built-in methods and variables. These provide support for:

- Accessing specific item data
- Accessing data about workflows, executions, and your n8n environment
- Convenience variables to help with data and time

Refer to [Built-in methods and variables](#) for more information.

Use AI in the Code node#

AI generated code overwrites your code

If you've already written some code on the **Code** tab, the AI generated code will replace it. n8n recommends using AI as a starting point to create your initial code, then editing it as needed.

To use ChatGPT to generate code in the Code node:

1. In the Code node, set **Language** to **JavaScript**.
2. Select the **Ask AI** tab.
3. Write your query.
4. Select **Generate Code**. n8n sends your query to ChatGPT, then displays the result in the **Code** tab.

Looping in n8n#

Looping is useful when you want to process multiple items or perform an action repeatedly, such as sending a message to every contact in your address book. n8n handles this repetitive processing automatically, meaning you don't need to specifically build loops into your workflows. There are [some nodes](#) where this isn't true.

Using loops in n8n#

n8n nodes take any number of items as input, process these items, and output the results. You can think of each item as a single data point, or a single row in the output table of a node.

The screenshot shows the 'Customer Datastore (n8n training)' node interface. On the left, the 'Parameters' tab is active, showing the 'Operation' set to 'Get All People' and 'Return All' toggled on. An arrow points from a dark blue box labeled 'Total items returned by the node' to the '5 Items' count in the 'OUTPUT' tab. The 'OUTPUT' tab displays a table with 5 items. On the right, two arrows labeled 'Item 1' and 'Item 2' point to the first two rows of the table.

id	name	email	notes	country	created
23423532	Jay Gatsby	gatsby@west-egg.com	Keeps asking about a green light??	US	1925-04-10
23423533	José Arcadio Buendía	jab@macondo.co	Lots of people named after him. Very confusing	CO	1967-05-05
23423534	Max Sendak	info@in-and-out-of-weeks.org	Keeps rolling his terrible eyes	US	1963-04-09
23423535	Zaphod Beeblebrox	captain@heartofgold.com	Felt like I was talking to more than one person	[null]	1979-10-12
23423536	Edmund Pevensie	edmund@narnia.gov	Passionate sailor	UK	1950-10-16

Nodes usually run once for each item. For example, if you wanted to send the name and notes of the customers in the Customer Datastore node as a message on Slack, you would:

1. Connect the Slack node to the Customer Datastore node.
2. Configure the parameters.
3. Execute the node.

You would receive five messages: one for each item.

This is how you can process multiple items without having to explicitly connect nodes in a loop.

Executing nodes once#

For situations where you don't want a node to process all received items, for example sending a Slack message only to the first customer, you can do so by toggling the **Execute Once** parameter in the **Settings** tab of that node. This setting is helpful when the incoming data contains multiple items and you want to only process the first one.

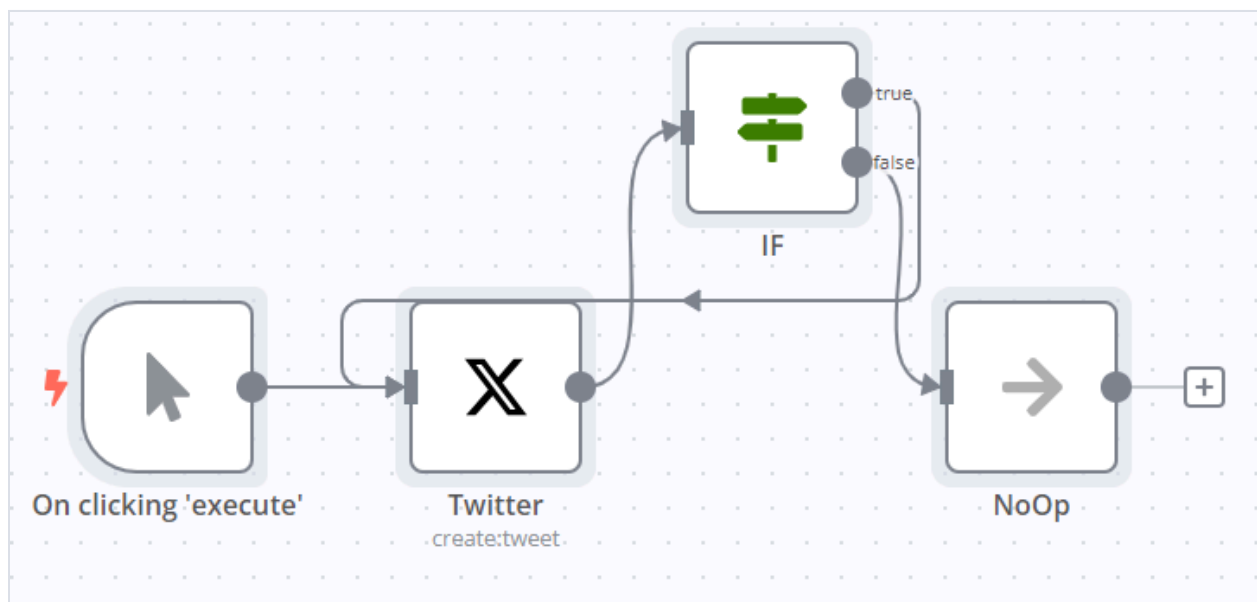
Creating loops#

n8n typically handles the iteration for all incoming items. However, there are certain scenarios where you will have to create a loop to iterate through all items. Refer to [Node exceptions](#) for a list of nodes that don't automatically iterate over all incoming items.

Loop until a condition is met#

To create a loop in an n8n workflow, connect the output of one node to the input of a previous node. Add an **IF** node to check when to stop the loop.

Here is an [example workflow](#) that implements a loop with an IF node:



Loop until all items are processed#

Use the [Loop Over Items](#) node when you want to loop until all items are processed. To process each item individually, set **Batch Size** to 1.

You can batch the data in groups and process these batches. This approach is useful for avoiding API rate limits when processing large incoming data or when you want to process a specific group of returned items.

The Loop Over Items node stops executing after all the incoming items get divided into batches and passed on to the next node in the workflow so it's not necessary to add an IF node to stop the loop.

Node exceptions#

Nodes and operations where you need to design a loop into your workflow:

- [Airtable](#):
 - List: this operation executes once, not for each incoming item.
- [Coda](#):
 - Get All: for the Table and View resources, this operation executes once.
- [CrateDB](#) node will execute and iterate over all incoming items only for Postgres related functions (for example, pgInsert, pgUpdate, pqQuery).
- [Code](#) node processes all the items based on the entered code snippet.
- [Google Cloud Firestore](#):
 - Get All: for the Collection and Document resources, this operation executes only once.

- **Google Drive:**
 - List: this operation executes only once, not for each incoming item.
- **Google Sheets:**
 - Read: this operation executes only once for the Sheet resource.
 - Update: this operation updates multiple rows if they're in the same range. It doesn't iterate through additional ranges.
- **HTTP Request:** you must handle pagination yourself. If your API call returns paginated results you must create a loop to fetch one page at a time.
- **Iterable** handles list operations in a single request, using the list ID defined for the first item. To address different lists in a single execution, you must create a loop with a batch size of 1.
- **Microsoft SQL** doesn't natively handle looping, so if you want the node to process all incoming items you must create a loop.
- **MongoDB** executes Find once, regardless of the number of incoming items.
- **Postgres** node will execute and iterate over all incoming items only for Postgres related functions (for example, pgInsert, pgUpdate, pqQuery).
- **QuestDB** node will execute and iterate over all incoming items only for Postgres related functions (for example, pgInsert, pgUpdate, pqQuery).
- **Read/Write File From Disk** node will fetch the files from the specified path only once. This node doesn't execute multiple times based on the incoming data. However, if the path is referenced from the incoming data, the node will fetch the files for all the valid paths.
- **Redis:**

- Info: this operation executes only once, regardless of the number of items in the incoming data.
- [RSS](#) nodes executes only once regardless of the number of items in the incoming data.
- [TimescaleDB](#) node will execute and iterate over all incoming items only for Postgres related functions (for example, `pgInsert`, `pgUpdate`, `pgQuery`).

Loop Over Items#

The Loop Over Items node helps you loop through data.

The node saves the original incoming data, and with each iteration, returns a predefined amount of data through the **loop** output.

When the node execution completes, it combines all the data and returns it through the **done** output.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Loop Over Items \(Split in Batches\) integrations](#) page.

Node reference#

- **Batch Size:** the number of items to return with each call.
- **Options:**
 - **Reset:** if set to true, the node will reset.

Check if you need this node

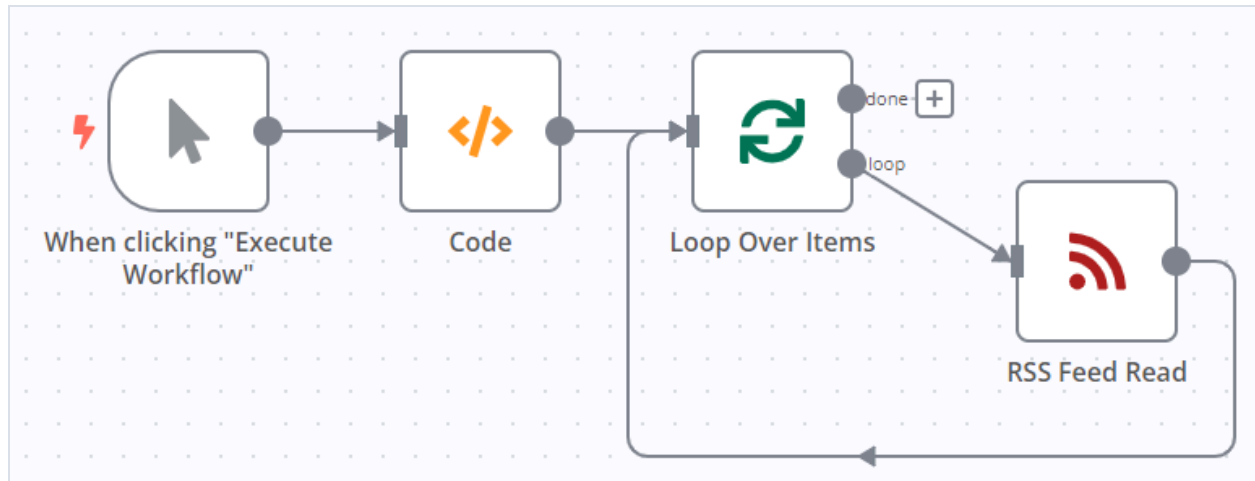
n8n automatically processes incoming items. You may not need the Loop Over Items node in your workflow. To learn more about how n8n handles multiple items, refer to the documentation on [Looping in n8n](#).

Example usage: Read RSS feed from two different sources#

This workflow allows you to read an RSS feed from two different sources using the Loop Over Items node. You need the Loop Over Items node in the workflow as the RSS Feed Read node only processes the first item it receives. You can also find the [workflow](#) on n8n.io.

The example walks through building the workflow, but assumes you are already familiar with n8n. To build your first workflow, including learning how to add nodes to a workflow, refer to [Try it out](#).

The final workflow looks like this:



1. Add the manual trigger.
2. Add the Code node.
3. Copy this code into the Code node:

```
1 return [  
2     {  
3         json: {  
4             url: 'https://medium.com/feed/n8n-io',  
5         },  
6     },  
7     {  
8         json: {  
9             url: 'https://dev.to/feed/n8n',  
10        },  
11    }  
12];
```

4. Add the Loop Over Items node.
5. Configure Loop Over Items: set the batch size to 1 in the **Batch Size** field.

6. Add the RSS Feed Read node.
7. Select **Test Workflow**. This runs the workflow to load data into the RSS Feed Read node.
8. Configure RSS Feed Read: map `url` from the input to the **URL** field. You can do this by dragging and dropping from the **INPUT** panel, or using this expression: `{{ $json.url }}`.
9. Select **Test Workflow** to run the workflow and see the resulting data.

Check that the node has processed all items#

To check if the node still has items to process, use the following expression: `{{ $node["Loop Over Items"].context["noItemsLeft"] }}`. This expression returns a boolean value. If the node still has data to process, the expression returns `false`, otherwise it returns `true`.

Get the current running index of the node#

To get the current running index of the node, use the following expression: `{{ $node["Loop Over Items"].context["currentRunIndex"] ; }}`.

Wait#

Use the Wait node pause your workflow's execution. When the workflow pauses it offloads the execution data to the database. When the resume condition is met, the workflow reloads the data and the execution continues.

The Wait node can resume on the following conditions:

- After time interval
- At specified time
- On webhook call

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Wait integrations](#) page.

Operations#

- Resume
 - After Time Interval
 - At Specified Time
 - On Webhook Call

Related resources#

View [example workflows and related content](#) on n8n's website.

Time-based operations#

For the time-based resume operations, note that:

- For wait times less than 65 seconds, the workflow doesn't offload execution data offloaded to the database. Instead, the process continues to run and execution resumes after the specified interval passes.
- The n8n server time is always used regardless of the timezone setting. Workflow timezone settings, and any changes made to them, don't affect the Wait node interval or specified time.

On Webhook Call usage#

The resume **On webhook call** option enables your workflows to resume when the Wait node receives an HTTP call.

The webhook URL that resumes the execution when called is generated at runtime. The Wait node provides the `$resumeWebhookUrl` variable so that you can reference and send the yet-to-be-generated URL wherever needed, for example to a third-party service or in an email.

When the workflow executes, the Wait node generates the resume URL and the webhook(s) in your workflow using the `$resumeWebhookUrl` reference become functional. This generated URL is unique to each execution, meaning that your workflow can contain multiple Wait nodes and as the webhook URL is called it will resume each Wait node sequentially.

Webhooks reference#

See the [Webhook node](#) documentation to learn more about the Authentication, Method, and Response parameters when configuring the Wait node to resume on a webhook call.

In addition to the parameters shared with the Webhook node, the Wait node has the following configuration options:

- **Limit wait time:** Set the maximum amount of time to wait before the execution resumes.
- **Add Option > Webhook Suffix:** Provide a suffix that you want to appended to the resume URL. This is useful for creating unique webhook URLs for each Wait node when a workflow contains multiple Wait nodes. Note that the generated `$resumeWebhookUrl` won't automatically include this suffix, you must manually append it to the webhook URL before exposing it.

Limitations#

There are some limitations to keep in mind when using On Webhook Call:

- Partial executions of your workflow change the `$resumeWebhookUrl`, so be sure that the node sending this URL to your desired third-party runs in the same execution as the Wait node.
- When testing your workflow using the Editor UI, you can't see the rest of the execution following the Wait node. To inspect the execution results enable **Save Manual Executions** in your [workflow settings](#) to be able to review the execution results there.


Sub-workflows#

You can call one workflow from another workflow. This allows you to build modular, microservice-like workflows. It can also help if your workflow grows large enough to encounter [memory issues](#). Creating sub-workflows uses the [Execute Workflow](#) and [Execute Workflow Trigger](#) nodes.

Set up and use a sub-workflow#

This section walks through setting up both the parent workflow and sub-workflow.

Create the sub-workflow#

1. Create a new workflow.
2. **Optional:** configure which workflows can call the sub-workflow:
 - a. Select the **Options**  menu > **Settings**. n8n opens the **Workflow settings** modal.
 - b. Change the **This workflow can be called by** setting. Refer to [Workflow settings](#) for more information on configuring your workflows.
3. Add the **Execute Workflow Trigger** node.
4. Add other nodes as needed to build your sub-workflow functionality.
5. Save the sub-workflow.

Sub-workflow mustn't contain errors

If there are errors in the sub-workflow, the parent workflow can't trigger it.

Load data into sub-workflow before building

This requires the ability to [load data from previous executions](#), which is available to Pro and Enterprise users.

If you want to load data into your sub-workflow to use while building it:

1. Create the sub-workflow and add the **Execute Workflow Trigger**.
2. In the sub-workflow [settings](#), set **Save successful production executions** to **Save**.
3. Skip ahead to setting up the parent workflow, and run it.
4. Follow the steps to [load data from previous executions](#).

You'll now have example data pinned in the trigger node, which allows you to work with real data when configuring the rest of the workflow.

Call the sub-workflow#

1. Open the workflow where you want to call the sub-workflow.
2. Add the **Execute Workflow** node.

3. In the **Execute Workflow** node, set the sub-workflow you want to call. You can choose to call the workflow by ID, load a workflow from a local file, add workflow JSON as a parameter in the node, or target a workflow by URL.
4. **Find your workflow ID**
5. Your sub-workflow's ID is the alphanumeric string at the end of its URL.
6. Save your workflow.

When your workflow executes, it will send data to the sub-workflow, and run it.

How data passes between workflows#

For example, there's an Execute Workflow node in **Workflow A**. The Execute Workflow node calls another workflow, **Workflow B**:

1. The Execute Workflow node passes the data to the Execute Workflow trigger node of **Workflow B**.
2. The last node of **Workflow B** sends the data back to the Execute Workflow node in **Workflow A**.

Error handling#

When designing your flow logic, it's a good practice to consider potential errors, and set up methods to handle them gracefully. With an error workflow, you can control how n8n responds to a workflow execution failure.

Investigating errors


To investigate failed executions, you can:

- Review your [Executions](#), for a [single workflow](#) or [all workflows you have access to](#). You can [load data from previous execution](#) into your current workflow.
- Enable [Log streaming](#).

Create and set an error workflow#

For each workflow, you can set an error workflow in **Workflow Settings**. It runs if an execution fails. This means you can, for example, send email or Slack alerts when a workflow execution errors. The error workflow must start with the [Error Trigger](#).

You can use the same error workflow for multiple workflows.

1. Create a new workflow, with the Error Trigger as the first node.
2. Give the workflow a name, for example Error Handler.
3. Select **Save**.
4. In the workflow where you want to use this error workflow:
 - a. Select **Options**  > **Settings**.

- b. In **Error workflow**, select the workflow you just created.
For example, if you used the name Error Handler, select **Error handler**.
- c. Select **Save**. Now, when this workflow errors, the related error workflow runs.

Error data#

The default error data received by the Error Trigger is:

```
1  [
2      {
3          "execution": {
4              "id": "231",
5              "url": "https://n8n.example.com/execution/231",
6              "retryOf": "34",
7              "error": {
8                  "message": "Example Error Message",
9                  "stack": "Stacktrace"
10             },
11             "lastNodeExecuted": "Node With Error",
12             "mode": "manual"
13         },
14         "workflow": {
15             "id": "1",
16             "name": "Example Workflow"
17         }
18     }
19 ]
```

All information is always present, except:

- `execution.id`: requires the execution to be saved in the database. Not present if the error is in the trigger node of the main workflow, as the workflow doesn't execute.
- `execution.url`: requires the execution to be saved in the database. Not present if the error is in the trigger node of the main workflow, as the workflow doesn't execute.
- `execution.retryOf`: only present when the execution is a retry of a failed execution.

If the error is caused by the trigger node of the main workflow, rather than a later stage, the data sent to the error workflow is different. There's less information in `execution{}` and more in `trigger{}`:

```

1  {
2    "trigger": {
3      "error": {
4        "context": {},
5        "name": "WorkflowActivationError",
6        "cause": {
7          "message": "",
8          "stack": ""
9        },
10       "timestamp": 1654609328787,
11       "message": "",
12       "node": {
13         ...
14       },
15       "mode": "trigger"
16     },
17     "workflow": {
18       "id": "",
19       "name": ""
20     }
21 }

```


Cause a workflow execution failure using Stop And Error#

When you create and set an error workflow, n8n runs it when an execution fails. Usually, this is due to things like errors in node settings, or the workflow running out of memory.

You can add the [Stop And Error](#) node to your workflow to force executions to fail under your chosen circumstances, and trigger the error workflow.

Execution order in multi-branch workflows#

n8n's node execution order depends on the version of n8n you're using:

- For workflows created before version 1.0: n8n executes the first node of each branch, then the second node of each branch, and so on.
- For workflows created in version 1.0 and above: executes each branch in turn, completing one branch before starting another. n8n orders the branches based on their position on the canvas, from

topmost to bottommost. If two branches are at the same height, the leftmost branch executes first.

You can change the execution order in your [workflow settings](#).

Data#

Data is the information that n8n nodes receive and process. For basic usage of n8n you don't need to understand data structures and manipulation.

However, it becomes important if you want to:

- Create your own node
- Write custom expressions
- Use the Function or Function Item node

This section covers:

- [Data structure](#)
- [Data flow within nodes](#)
- [Transforming data](#)
- [Process data using code](#)
- [Pinning](#) and [editing](#) data during workflow development.
- [Data mapping](#) and [Item linking](#): how data items link to each other.

Related resources#

Data transformation nodes#

n8n provides a collection of nodes to transform data:

- **Aggregate**: take separate items, or portions of them, and group them together into individual items.
- **Limit**: remove items beyond a defined maximum number.
- **Remove Duplicates**: identify and delete items that are identical across all fields or a subset of fields.
- **Sort**: organize lists of in a desired ordering, or generate a random selection.
- **Split Out**: separate a single data item containing a list into multiple items.
- **Summarize**: aggregate items together, in a manner similar to Excel pivot tables

Data structure#

In n8n, all data passed between nodes is an array of objects. It has the following structure:

```

1  [
2      {
3          // For most data:
4          // Wrap each item in another object, with the key 'json'
5          "json": {
6              // Example data
7              "apple": "beets",
8              "carrot": {
9                  "dill": 1
10             }
11         },
12         // For binary data:
13         // Wrap each item in another object, with the key 'binary'
14         "binary": {
15             // Example data
16             "apple-picture": {
17                 "data": "....", // Base64 encoded binary data (required)
18                 "mimeType": "image/png", // Best practice to set if
19                 // possible (optional)
20                 "fileExtension": "png", // Best practice to set if
21                 // (optional)
22                 "fileName": "example.png", // Best practice to set if
23                 // possible (optional)
24             }
25         }
26     },
27 ]

```

Skipping the `json` key and array syntax

From 0.166.0 on, when using the Function node or Code node, n8n automatically adds the `json` key if it's missing. It also automatically wraps your items in an array (`[]`) if needed. This is only the case when using the Function or Code nodes. When

building your own nodes, you must still make sure the node returns data with the `json` key.

Data item processing#

Nodes can process multiple items.

For example, if you set the Trello node to Create-Card, and create an expression that sets Name using a property called `name-input-value` from the incoming data, the node creates a card for each item, always choosing the `name-input-value` of the current item.

For example, this input will create two cards. One named `test1` the other one named `test2`:

```
1 [
2   {
3     name-input-value: "test1"
4   },
5   {
6     name-input-value: "test2"
7   }
8 ]
```

Data flow within nodes#

Nodes can process multiple items.

For example, if you set the Trello node to Create-Card, and create an expression that sets Name using a property called name-input-value from the incoming data, the node creates a card for each item, always choosing the name-input-value of the current item.

For example, this input will create two cards. One named test1 the other one named test2:

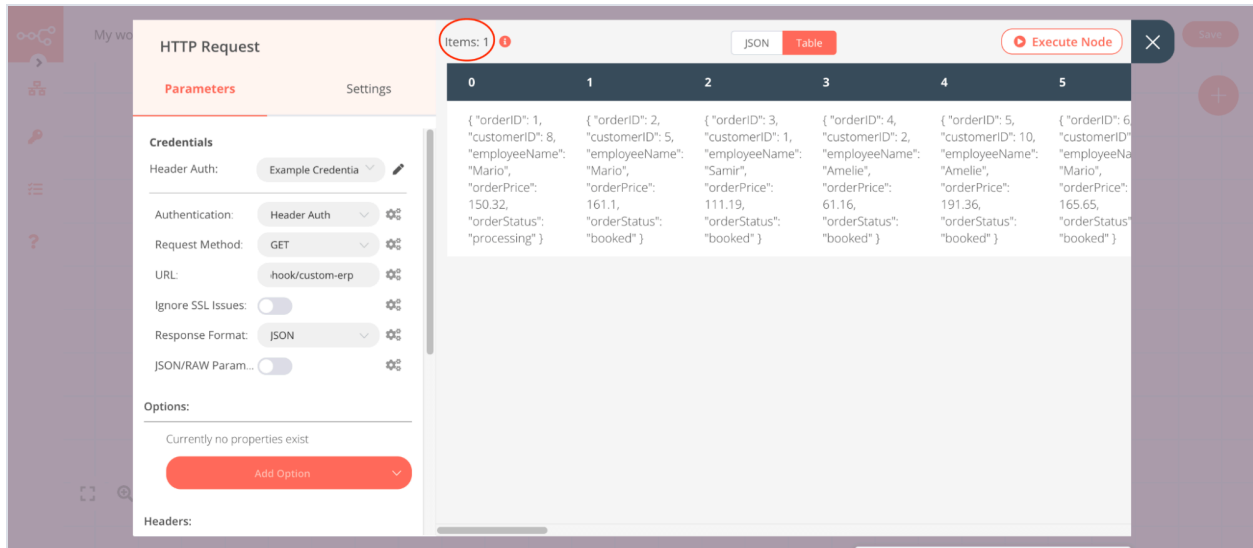
```
1 [
2   {
3     name-input-value: "test1"
4   },
5   {
6     name-input-value: "test2"
7   }
8 ]
```

Transforming data#

n8n uses a predefined [data structure](#) that allows all nodes to process incoming data correctly.

Your incoming data may have a different data structure, in which case you will need to transform it to allow each item to be processed individually.

For example, the image below shows the output of an [HTTP Request](#) node that returns data incompatible with n8n's data structure. The node returns the data and displays that only one item was returned.



To transform this kind of structure into the n8n data structure you can use the data transformation nodes:

- **Aggregate**: take separate items, or portions of them, and group them together into individual items.
- **Limit**: remove items beyond a defined maximum number.
- **Remove Duplicates**: identify and delete items that are identical across all fields or a subset of fields.
- **Sort**: organize lists of in a desired ordering, or generate a random selection.
- **Split Out**: separate a single data item containing a list into multiple items.
- **Summarize**: aggregate items together, in a manner similar to Excel pivot tables.

Aggregate#

Use the Aggregate node to take separate items, or portions of them, and group them together into individual items.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Aggregate integrations](#) page.

Node parameters#

- **Aggregate:** choose whether to aggregate **Individual Fields** or **All Item Data**.
- If you choose **Individual Fields**, you can then configure the fields you want to aggregate with the following parameters:
 - **Input Field Name:** the name of the field in the input data to be aggregated together.
 - **Rename Field:** enable this toggle to enter a field name for the aggregated output data. When aggregating multiple fields you must provide new output field names. You can't leave multiple fields undefined.
 - **Output Field Name:** displayed when you enable **Rename Field**. The field name for the aggregated output data.
 - **Options > Add Field:** use this to add more optional settings, including:
 - **Disable Dot Notation:** when disabled, you can't reference child fields (in the format `parent.child`).
 - **Merge Lists:** enable this if the field to aggregate is a list, and you want to output a single flat list rather than a list of lists.

- **Include Binaries:** include binary data from the input in the new output.
- **Keep Missing And Null Values:** enable this to add a null (empty) entry in the output list when there is a null or missing value in the input.
- If you choose **All Item Data**, you can then set:
 - **Put Output in Field:** the name of the output field.
 - **Include:** choose from **All fields**, **Specified Fields**, or **All Fields Except**.
 - **Options > Add Field:** use this to add the setting:
 - **Include Binaries:** include binary data from the input in the new output.

Related resources#

View [example workflows and related content](#) on n8n's website.

Learn more about [data structure and data flow](#) in n8n workflows.

Remove Duplicates#

Use the Remove Duplicates node to identify and delete items that are identical across all fields or a subset of fields. This is helpful in situations where you can end up with duplicate data, such as a user creating multiple accounts, or a customer submitting the same order multiple times. When working with large datasets it becomes more difficult to spot and remove these items.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Remove Duplicates integrations](#) page.

Node parameters#

- **Compare:** specify which fields of the input data n8n should compare to check if they're the same. The following options are available:
 - **All Fields:** compares all fields of the input data.
 - **All Fields Except:** enter which input data fields n8n should exclude from the comparison. You can provide multiple values separated by commas.
 - **Selected Fields:** enter which input data fields n8n should include in the comparison. You can provide multiple values separated by commas.
- If you choose **All Fields Except** or **Selected Fields**, n8n displays **Options > Add Field**. Use this to add more optional settings, including:
 - **Disable Dot Notation:** when disabled, you can't reference child fields (in the format `parent.child`).
 - **Remove Other Fields:** keep the fields that you're comparing and remove the others.
-

Sort#

Use the Sort node to organize lists of items in a desired ordering, or generate a random selection.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Sort integrations](#) page.

Array sort behavior

The Sort operation uses the default JavaScript operation where the elements to be sorted are converted into strings and their values compared. Refer to [Mozilla's guide to Array sort](#) to learn more.

Node parameters#

- **Type:** use the dropdown to select how you want to input the sorting. The following options are available:
 - **Simple:** when selected, you can use the **Add Field To Sort By** button to input the fields, and select whether to use **Ascending** or **Descending** order.
 - **Random:** select to create a random order in the list.
 - **Code:** when selected, displays a code input field where you can enter custom JavaScript code to perform the sort operation.

- **Options > Add Field:** use this to add more optional settings, including:
 - **Disable Dot Notation:** when disabled, you can't reference child fields (in the format `parent.child`).

Split Out#

Use the Split Out node to separate a single data item containing a list into multiple items. For example, a list of customers, and you want to split them so that you have an item for each customer.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Split Out integrations](#) page.

Node parameters#

- **Field to Split Out:** the field containing the list you want to separate out into individual items.
 - If working with binary data inputs, use `$binary` in an expression to set the field to split out.
- **Include:** select if you want n8n to keep any other fields from the input data with each new individual item. You can select:
 - **No Other Fields**
 - **All Other Fields**

- **Selected Other Fields:** when selected, n8n displays **Fields to Include**. Enter a comma separated list of desired fields.
- **Options > Add Field:** use this to add more optional settings, including:
 - **Disable Dot Notation:** when disabled, you can't reference child fields (in the format `parent.child`).
 - **Destination Field Name:** optionally set the field name under which to put the new split contents.
 - **Include Binary:** include binary data from the input in the new output.
-

Summarize#

Use the Summarize node to aggregate items together, in a manner similar to Excel pivot tables.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Summarize integrations](#) page.

Node parameters#

- **Fields to Summarize:**

- To combine values, select an **Aggregation** method, and enter a **Field** name.
 - To split values, enter a field name or list of names in **Fields to Split By**.
- **Options > Add Field**: use this to add more optional settings, including:
 - **Disable Dot Notation**: when disabled, you can't reference child fields (in the format `parent.child`).
 - **Output Format**:
 - **Each Split in a Separate Item**: splitting generates a separate output item for each split out field.
 - **All Splits in a Single Item**: splitting generates a single item, which lists the split out fields.
 - **Ignore items without valid fields to group by**: ignore input items that don't contain the field specified in **Fields to Summarize**.
-

Custom API operations#

One of the most complex parts of setting up API calls is managing authentication. n8n provides credentials support for operations and services beyond those supported by built-in nodes.

- Custom operations for existing nodes: n8n supplies hundreds of nodes, allowing you to create workflows that link multiple products. However, some nodes don't include all the possible operations

supported by a product's API. You can work around this by making a custom API call using the [HTTP Request](#) node.

- **Credential-only nodes:** n8n includes credential-only nodes. These are integrations where n8n supports setting up credentials for use in the HTTP Request node, but doesn't provide a standalone node. You can find a credential-only node in the nodes panel, as you would for any other integration.

Predefined credential types#

A predefined credential type is a credential that already exists in n8n. You can use predefined credential types instead of generic credentials in the HTTP Request node.

For example: you create an Asana credential, for use with the Asana node. Later, you want to perform an operation that isn't supported by the Asana node, using Asana's API. You can use your existing Asana credential in the HTTP Request node to perform the operation, without additional authentication setup.

Using predefined credential types#

To use a predefined credential type:

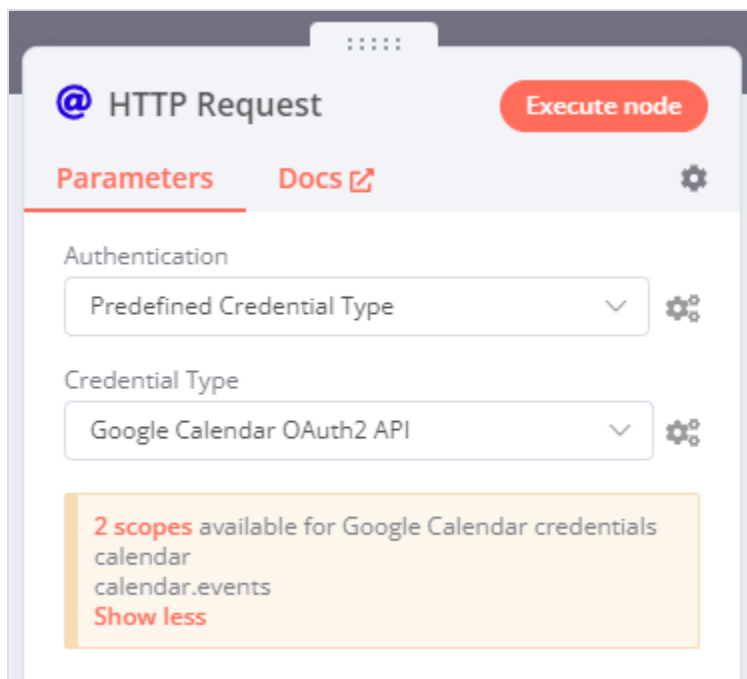
1. Open your HTTP Request node, or add a new one to your workflow.
2. In **Authentication**, select **Predefined Credential Type**.
3. In **Credential Type**, select the API you want to use.
4. In **Credential for <API name>**, you can:

- a. Select an existing credential for that platform, if available.
- b. Select **Create New** to create a new credential.

Credential scopes#

Some existing credential types have specific scopes: endpoints that they work with. n8n warns you about this when you select the credential type.

For example, follow the steps in [Using predefined credential types](#), and select **Google Calendar OAuth2 API** as your **Credential Type**. n8n displays a box listing the two endpoints you can use this credential type with:



Install community nodes in the n8n app#

Limited to n8n instance owners

Only the n8n instance owner can install and manage community nodes. The instance owner is the person who sets up and manages user management.

Install a community node#

To install a community node:

1. Go to **Settings > Community Nodes**.
2. Select **Install**.
3. Find the node you want to install:
 - Select **Browse**. n8n takes you to an npm search results page, showing all npm packages tagged with the keyword `n8n-community-node-package`.
 - Browse the list of results. You can filter the results or add more keywords.
 - Once you find the package you want, make a note of the package name. If you want to install a specific version, make a note of the version number as well.
 - Return to n8n.
4. Enter the npm package name, and version number if required. For example, consider a community node designed to access a weather API called "Storms." The package name is `n8n-node-storms`, and it has three major versions.
 - To install the latest version of a package called `n8n-node-weather`: enter `n8n-nodes-storms` in **Enter npm package name**.


- To install version 2.3: enter `n8n-node-storms@2.3` in **Enter npm package name**.
5. Agree to the [risks](#) of using community nodes: select **I understand the risks of installing unverified code from a public source**.
 6. Select **Install**. n8n installs the node, and returns to the **Community Nodes** list in **Settings**.

Nodes on the blocklist

n8n maintains a blocklist of community nodes that it prevents you from installing. Refer to [n8n community node blocklist](#) for more information.

Uninstall a community node#

To uninstall a community node:

1. Go to **Settings > Community nodes**.
2. On the node you want to install, select **Options** .
3. Select **Uninstall package**.
4. Select **Uninstall Package** in the confirmation modal.

Upgrade a community node#

Breaking changes in versions

Node developers may introduce breaking changes in new versions of their nodes. A breaking change is an update that breaks previous functionality. Depending on the node versioning approach that a node developer chooses, upgrading to a version with a breaking change could cause all workflows using the node to break. Be careful when upgrading your nodes. If you find that an upgrade causes issues, you can [downgrade](#).

Upgrade to the latest version#

You can upgrade community nodes to the latest version from the node list in **Settings > community nodes**.

When a new version of a community node is available, n8n displays an **Update** button on the node. Click the button to upgrade to the latest version.

Upgrade to a specific version#

To upgrade to a specific version (a version other than the latest), uninstall the node, then reinstall it, making sure to specify the target version. Follow the [Installation](#) instructions for more guidance.

Downgrade a community node#

If there is a problem with a particular version of a community node, you may want to roll back to a previous version.

To do this, uninstall the community node, then reinstall it, targeting a specific node version. Follow the [Installation](#) instructions for more guidance.

Core nodes library

Chat Trigger#

Use the Chat Trigger node when building AI workflows for chatbots and other chat interfaces. You can configure how users access the chat, using one of n8n's provided interfaces, or your own. You can add authentication.

You must connect either an agent or chain [root node](#).

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Chat Trigger integrations](#) page.

Workflow execution usage

Every message to the Chat Trigger executes your workflow. This means that one conversation where a user sends 10 messages uses 10 executions from your execution allowance. Check your payment plan for details of your allowance.

Manual Chat Trigger

This node replaces the Manual Chat Trigger node from version 1.24.0.

Node parameters#

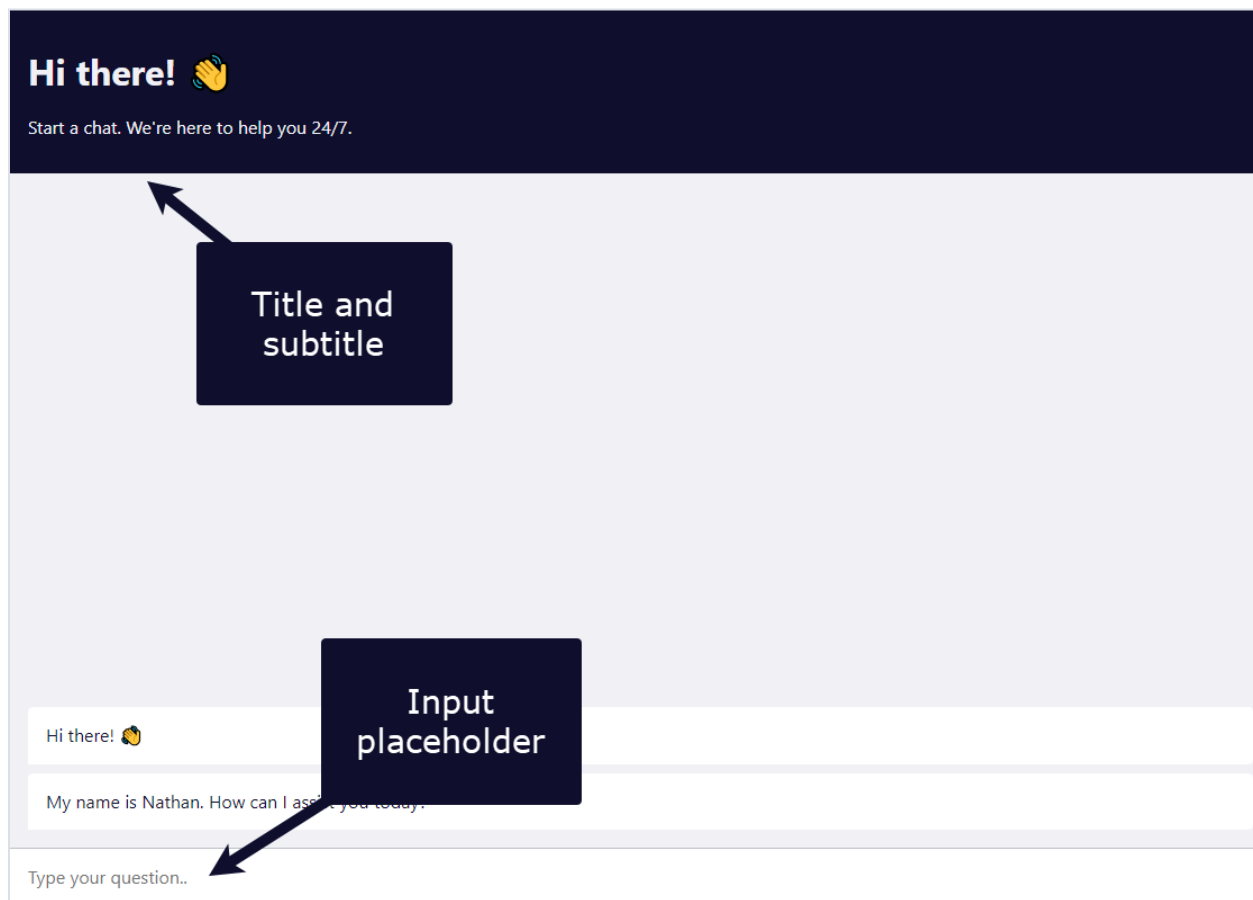
- **Make Chat Publicly Available:** enable this when you're ready to activate the workflow and allow users to access the chat. Leave it disabled when building the workflow.
- **Mode:** choose how users access the chat.
 - Choose **Hosted Chat** to use n8n's hosted chat interface. n8n recommends this for most users: you can configure the interface using the [node options](#), and don't have to do any other setup.
 - **Embedded Chat** requires you to create your own chat interface. You can use n8n's [chat widget](#) or build your own. Your chat interface must call the webhook URL shown in **Chat URL** in the node.
- **Authentication:** you can restrict access to the chat.
 - **None:** no authentication. Anyone can use the chat.
 - **Basic Auth:** set up a username and password. The same username and password must be used by all users.
 - **n8n User Auth:** the user must have an n8n account.
- If using hosted chat, you can configure the **Initial Message(s)**. This is the message the n8n chat interface displays when the user arrives on the page.

Node options#

Available options depend on the chat mode.

Hosted chat options#

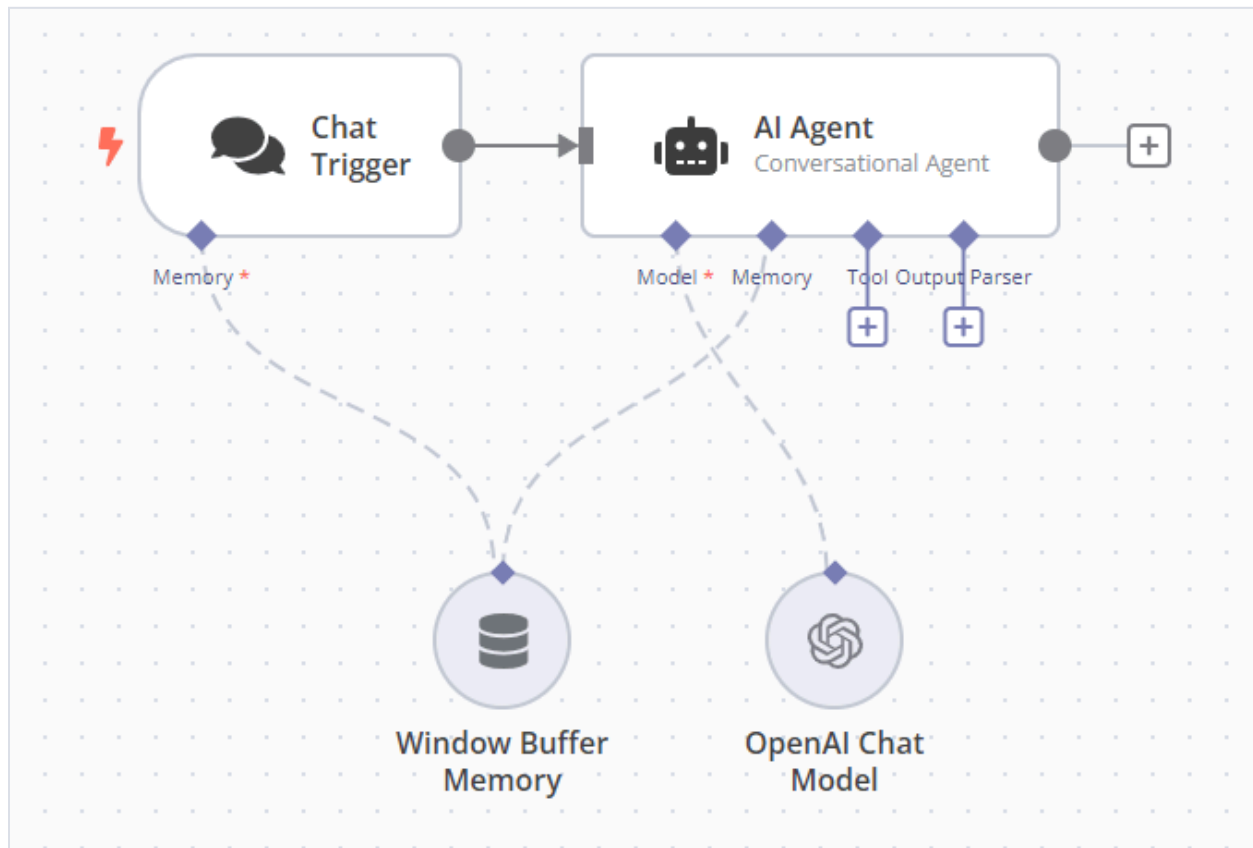
- **Input Placeholder, Title, and Subtitle:** set text elements in the chat interface.
- **View screenshot**



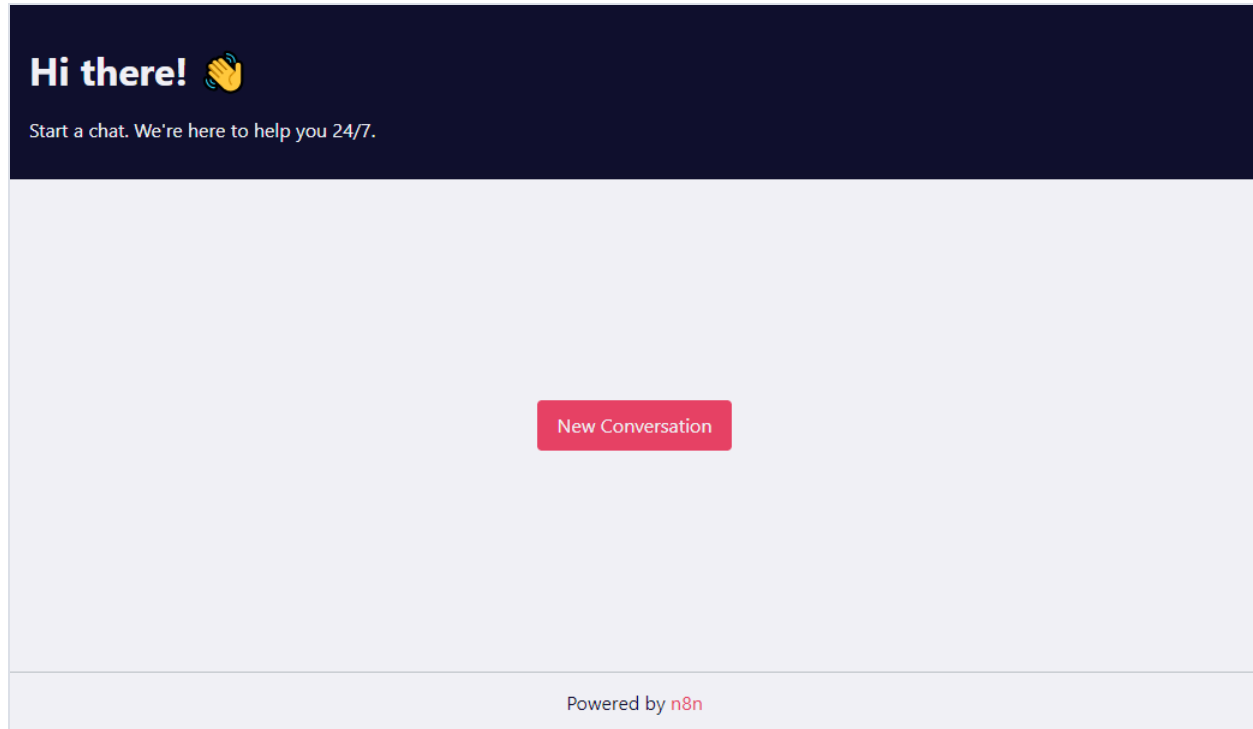
- **Load Previous Session:** whether to load chat messages from a previous chat session. When you enable this, you must connect the Chat Trigger and the Agent you're using to a memory sub-node. The memory connector on the Chat Trigger appears when you set

Load Previous Session to From Memory. n8n recommends connecting both the Chat Trigger and Agent to the same memory sub-node, as this ensures a single source of truth for both nodes.

- **View screenshot**



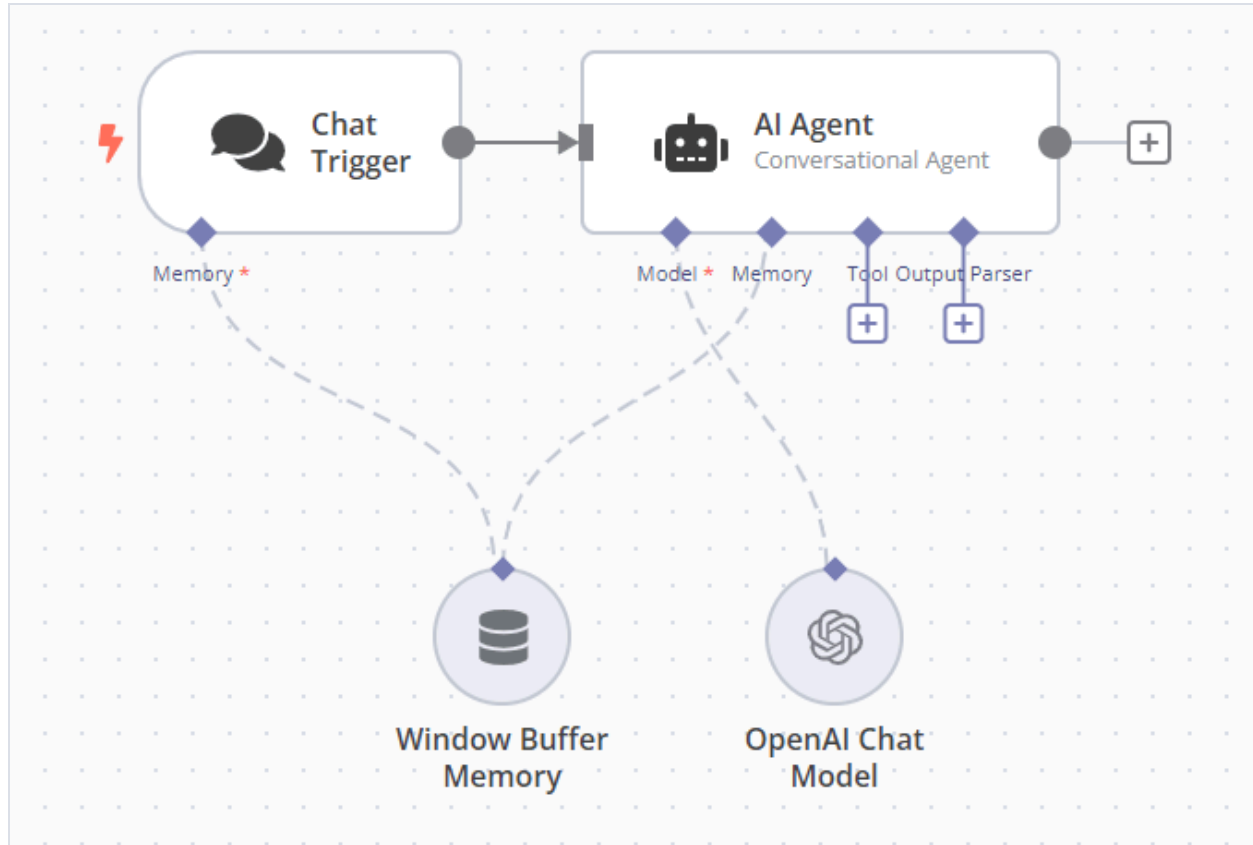
-
- **Response Mode:** use this when building a workflow with steps after the agent or chain that's handling the chat.
 - **When Last Node Finishes:** the Chat Trigger node returns the response code and the data output from the last node executed in the workflow.
 - **Using 'Respond to Webhook' Node:** the Chat Trigger node responds as defined in the [Respond to Webhook](#) node.
- **Require Button Click to Start Chat:** display a **New Conversation** button on the chat interface.
- **View screenshot**



-
- **Allowed Origin (CORS):** which origins can access the chat URL.

Embedded chat options#

- **Load Previous Session:** whether to load chat messages from a previous chat session. When you enable this, you must connect the Chat Trigger and the Agent you're using to a memory sub-node. The memory connector on the Chat Trigger appears when you set **Load Previous Session** to **From Memory**. n8n recommends connecting both the Chat Trigger and Agent to the same memory sub-node, as this ensures a single source of truth for both nodes.
- **View screenshot**



-
- **Response Mode:** use this when building a workflow with steps after the agent or chain that's handling the chat.
 - **When Last Node Finishes:** the Chat Trigger node returns the response code and the data output from the last node executed in the workflow.
 - **Using 'Respond to Webhook' Node:** the Chat Trigger node responds as defined in the [Respond to Webhook](#) node.
- **Allowed Origin (CORS):** which origins can access the chat URL.

Date & Time#

The Date & Time node manipulates date and time data and convert it to different formats.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Date & Time integrations](#) list.

Timezone settings

The node relies on the timezone setting. n8n uses either:

1. The workflow timezone, if set. Refer to [Workflow settings](#) for more information.
2. The n8n instance timezone, if the workflow timezone isn't set. The default is `America/New York` for self-hosted instances. n8n Cloud tries to detect the instance owner's timezone when they sign up, falling back to GMT as the default. Self-hosted users can change the instance setting using [Environment variables](#). Cloud admins can change the instance timezone in the [Admin dashboard](#).

Date and time in other nodes

You can work with data and time in the Code node, and in expressions in any node. n8n supports Luxon to help work with date and time in JavaScript. Refer to [Date and time with Luxon](#) for more information.

Operations#

- Add to a Date
- Extract Part of a Date
- Format a Date
- Get Current Date
- Get Time Between Dates
- Round a Date
- Subtract From a Date

Related resources#

View [example workflows and related content](#) on n8n's website.

The Date & Time node uses [Luxon](#). You can also use Luxon in the Code node and expressions. Refer to [Date and time with Luxon](#) for more information.

Supported date formats#

n8n supports all date formats [supported by Luxon](#).

Edit Fields (Set)#

Use the Edit Fields node to set workflow data. This node can set new data as well as overwrite data that already exists. This node is crucial in workflows which expect incoming data from previous nodes, such as when inserting values to Google Sheets or databases.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Edit Fields integrations](#) page.

Node parameters#

These are the settings and options available in the Edit Fields node.

Mode#

You can either use **Manual Mapping** to edit fields using the GUI, or **JSON Output** to write JSON that n8n adds to the input data.

Fields to Set#

If you select **Mode > Manual Mapping**, you can configure the fields by dragging and dropping values from **INPUT**.

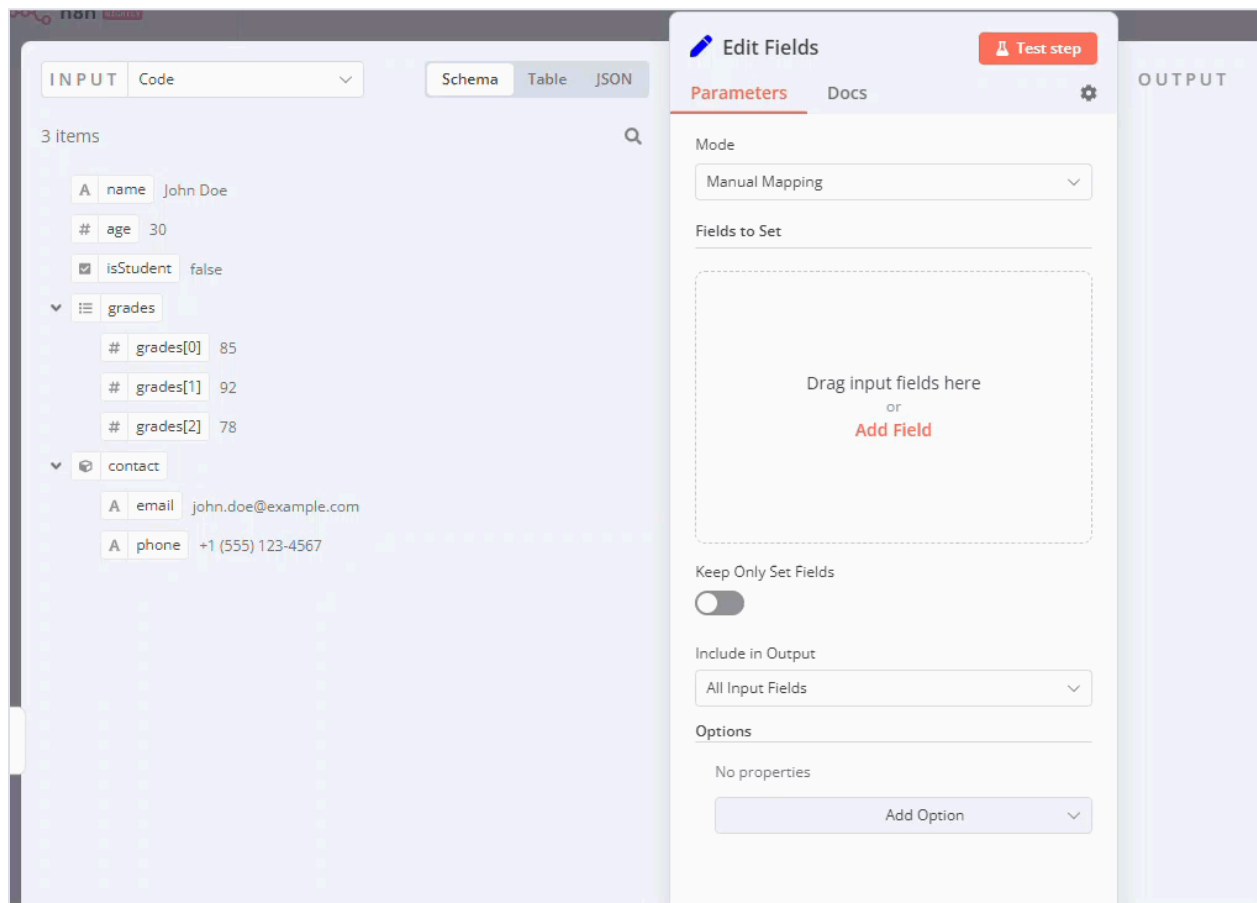
The default behavior when you drag a value is:

- n8n sets the value's name as the field name.
- The field value contains an expression which accesses the value.

If you don't want to use expressions:

1. Hover over a field. n8n displays the **Fixed | Expressions** toggle.
2. Select **Fixed**.

You can do this for both the name and value of the field.



Keep Only Set Fields#

Enable this to discard any input data that you don't use in **Fields to Set**.

Include in Output#

Choose which input data to include in the node's output data.

Options#

Customize the behavior of the node.

Include Binary Data#

If the input data includes binary data, choose whether to include it in the Edit Fields node's output data.

Ignore Type Conversion Errors#

Manual Mapping only.

Enabling this allows n8n to ignore some data type errors when mapping fields.

Support Dot Notation#

By default, n8n supports dot notation.

For example, when using manual mapping, the node follows the dot notation for the **Name** field. That means if you set the name in the **Name** field as `number.one` and the value in the **Value** field as `20`, the resulting JSON is:

```
{ "number": { "one": 20 } }
```

1

You can prevent this behavior by selecting **Add Option > Support Dot Notation**, and setting the **Dot Notation** field to off. Now the resulting JSON is:

```
{ "number.one": 20 }
```

1

Arrays and expressions in JSON

Output mode#

You can use arrays and expressions when creating your JSON Output.

For example, given this input data generated by the Customer Datastore node:

```
1  [
2    {
3      "id": "23423532",
4      "name": "Jay Gatsby",
5      "email": "gatsby@west-egg.com",
6      "notes": "Keeps asking about a green light??",
7      "country": "US",
8      "created": "1925-04-10"
9    },
10   {
11     "id": "23423533",
12     "name": "José Arcadio Buendía",
13     "email": "jab@macondo.co",
14     "notes": "Lots of people named after him. Very confusing",
15     "country": "CO",
16     "created": "1967-05-05"
17   },
18   {
19     "id": "23423534",
20     "name": "Max Sendak",
21     "email": "info@in-and-out-of-weeks.org",
22     "notes": "Keeps rolling his terrible eyes",
23     "country": "US",
24     "created": "1963-04-09"
25   },
26   {
27     "id": "23423535",
28     "name": "Zaphod Beeblebrox",
29     "email": "captain@heartofgold.com",
30     "notes": "Felt like I was talking to more than one person",
31     "country": null,
32     "created": "1979-10-12"
33   },
34   {
35     "id": "23423536",
36     "name": "Edmund Pevensie",
37     "email": "edmund@narnia.gov",
38     "notes": "Passionate sailor",
39     "country": "UK",
40     "created": "1950-10-16"
41   }
42 ]
```


Add the following JSON in the **JSON Output** field, with **Include in Output** set to **All Input Fields**:

```
1 {  
2   "newKey": "new value",  
3   "array": [{"{{ $json.id }}", "{{ $json.name }}"},  
4   "object": {  
5     "innerKey1": "new value",  
6     "innerKey2": "{{ $json.id }}",  
7     "innerKey3": "{{ $json.name }}",  
8   }  
9 }
```

You get this output:

```
1  [
2    {
3      "id": "23423532",
4      "name": "Jay Gatsby",
5      "email": "gatsby@west-egg.com",
6      "notes": "Keeps asking about a green light??",
7      "country": "US",
8      "created": "1925-04-10",
9      "newKey": "new value",
10     "array": [
11       23423532,
12       "Jay Gatsby"
13     ],
14     "object": {
15       "innerKey1": "new value",
16       "innerKey2": "23423532",
17       "innerKey3": "Jay Gatsby"
18     }
19   },
20   {
21     "id": "23423533",
22     "name": "José Arcadio Buendía",
23     "email": "jab@macondo.co",
24     "notes": "Lots of people named after him. Very confusing",
25     "country": "CO",
26     "created": "1967-05-05",
27     "newKey": "new value",
28     "array": [
29       23423533,
30       "José Arcadio Buendía"
31     ],
32     "object": {
33       "innerKey1": "new value",
34       "innerKey2": "23423533",
35       "innerKey3": "José Arcadio Buendía"
36     }
37   },
38   {
39     "id": "23423534",
40     "name": "Max Sendak",
41     "email": "info@in-and-out-of-weeks.org",
42     "notes": "Keeps rolling his terrible eyes",
```

```
42  "country": "US",
43  "created": "1963-04-09",
44  "newKey": "new value",
45  "array": [
46    23423534,
47    "Max Sendak"
48  ],
49  "object": {
50    "innerKey1": "new value",
51    "innerKey2": "23423534",
52    "innerKey3": "Max Sendak"
53  }
54 },
55 {
56   "id": "23423535",
57   "name": "Zaphod Beeblebrox",
58   "email": "captain@heartofgold.com",
59   "notes": "Felt like I was talking to more than one person",
60   "country": null,
61   "created": "1979-10-12",
62   "newKey": "new value",
63   "array": [
64     23423535,
65     "Zaphod Beeblebrox"
66   ],
67   "object": {
68     "innerKey1": "new value",
69     "innerKey2": "23423535",
70     "innerKey3": "Zaphod Beeblebrox"
71   }
72 },
73 {
74   "id": "23423536",
75   "name": "Edmund Pevensie",
76   "email": "edmund@narnia.gov",
77   "notes": "Passionate sailor",
78   "country": "UK",
79   "created": "1950-10-16",
80   "newKey": "new value",
81   "array": [
82     23423536,
83     "Edmund Pevensie"
```

```
83   },
84   "object": {
85     "innerKey1": "new value",
86     "innerKey2": "23423536",
87     "innerKey3": "Edmund Pevensie"
88   }
89 }
90 ]
91
92
```

Edit Image#

Use the Edit Image node to manipulate and edit images.

Dependencies

1. If you aren't running n8n on Docker, you need to install [GraphicsMagick](#).
2. You need to use a node such as the [Read/Write Files from Disk](#) node or the [HTTP Request](#) node to pass the image file as a data property to the Edit Image node.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Edit Image integrations](#) page.

Operations#

- Add a blur to the image to reduce sharpness
- Add a border to the image
- Create a new image
- Crop the image
- Composite an image on top of another image
- Draw on an image
- Get information about the image
- Rotate the image
- Change the size of the image
- Shear image along the X or Y axis
- Add text to the image

Options#

- **File Name:** specify the filename of the output file.
- **Format:** specify the image format of the output file:
 - BMP
 - GIF
 - JPEG
 - PNG
 - TIFF
 - WebP

Related resources#

View [example workflows and related content](#) on n8n's website.

HTML#

The HTML node provides operations to help you work with HTML in n8n.

HTML Extract node

The HTML node replaces the HTML Extract node from version 0.213.0 on. If you're using an older version of n8n, you can still view the [HTML Extract node documentation](#).

Cross-site scripting

When using the HTML node to generate an HTML template you can introduce [XSS \(cross-site scripting\)](#). This is a security risk. Be careful with un-trusted inputs.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [HTML integrations](#) page.

Operations#

- Generate HTML template
- Extract HTML content

Related resources#

View [example workflows and related content](#) on n8n's website.

Generate HTML template#

Create an HTML template. This allows you to take data from your workflow and output it as HTML. You can use [Expressions](#) in the template, including n8n's [Built-in methods and variables](#)

You can include:

- Standard HTML
- CSS in `<style>` tags.
- JavaScript in `<script>` tags. n8n doesn't execute the JavaScript.
- Expressions, wrapped in `{{}}`.

Extract HTML Content#

Extract contents from an HTML-formatted source. The source can be in JSON, or a binary file (.html).

- **Source Data:** choose the source type, JSON or binary.

- **JSON Property** or **Binary Property**: the name of the input containing the HTML you want to extract. The property can either contain a string or an array of strings.
- **Extraction Values**:
 - **Key**: the key to save the extracted value under.
 - **CSS Selector**: the CSS selector to search for.
 - **Return Value**: the data to return. In this dropdown list there are four options.
 - **Attribute**: get an attribute value like 'class' from an element.
 - **Attribute**: the name of the attribute to return the value of.
 - **HTML**: get the HTML that the element contains.
 - **Text**: get the text content of the element.
 - **Value**: get the value of an input, select, or text area.
 - **Return Array**: returns the values as an array so that if n8n finds multiple values, it returns them as individual items in an array. If you don't set this, n8n returns all values as a single string.
- **Options**:
 - **Trim Values**: removes all spaces and newlines from the beginning and end of the values.
-

HTTP Request#

The HTTP Request node is one of the most versatile nodes in n8n. It allows you to make HTTP requests to query data from any app or service with a REST API.

When using this node, you're creating a REST API call. You need some understanding of basic API terminology and concepts.

There are two ways to create an HTTP request: configure the [node fields](#) or [import a curl command](#).

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [HTTP Request integrations](#) page.

Node parameters#

Method#

Select the method to use for the request:

- DELETE
- GET
- HEAD
- OPTIONS
- PATCH
- POST
- PUT

URL#

Enter the endpoint you want to use.

Authentication#

n8n recommends using the **Predefined Credential Type** option when it's available. It offers an easier way to set up and manage credentials, compared to configuring generic credentials.

Predefined credentials#

Select **Predefined Credential Type**. This allows you to perform custom operations, without additional authentication setup. For example, n8n has an Asana node, and supports using your Asana credentials in the HTTP Request node. Refer to [Custom API operations](#) for more information.

Generic credentials#

Select **Generic Credential Type** to set up authentication using one of the following methods:

- Basic Auth
- Custom Auth
- Digest Auth
- Header Auth
- OAuth1 API
- OAuth2 API
- Query Auth

Refer to [HTTP request credentials](#) for more information setting up each credential type.

Parameters, headers, and body#

You can choose to send additional information with your request. The data you need to send depends on the API you're interacting with, and the type of request you're making. Refer to your service's API documentation for detailed guidance.

- **Send Query Parameters:** include query parameters. Query parameters are usually used as filters or searches on your query.
- **Send Headers:** include request headers. Headers contain metadata about your request.
- **Send Body:** send additional information in the body of your request.

Node options#

Select **Add Option** to view and select these options.

- **Batching:** control how to batch large responses.
- **Ignore SSL Issues:** download the response even if SSL validation isn't possible.
- **Redirects:** choose whether to follow redirects. Enabled by default.
- **Response:** provide settings about the expected API response.
- **Pagination:** handle query results that are too big for the API to return in a single call.
- **Proxy:** use this if you need to specify an HTTP proxy.
- **Timeout:** set a timeout for the request.

Pagination#

Use this option to paginate results.

Inspect the API data first

Some options for pagination require knowledge of the data returned by the API you're using. Before setting up pagination, either check the API documentation, or do an API call without pagination, to see the data it returns.

Understand pagination

Configure the pagination settings:

- **Pagination Mode:**
 - **Off:** turn off pagination.
 - **Update a Parameter in Each Request:** use this when you need to dynamically set parameters for each request.
 - **Response Contains Next URL:** use this when the API response includes the URL of the next page. Use an expression to set **Next URL**.

For example setups, refer to [HTTP Request node cookbook | Pagination](#).

n8n provides built-in variables for working with HTTP node requests and responses when using pagination:

Variable	Description
<code>\$pageCount</code>	The pagination count. Tracks how many pages the node has fetched.
<code>\$request</code>	The request object sent by the HTTP node.
<code>\$response</code>	The response object from the HTTP call. Includes <code>\$response.body</code> , <code>\$response.headers</code> , and <code>\$response.statusCode</code> . The contents of <code>body</code> and <code>headers</code> depend on the data sent by the API.

API differences

Different APIs implement pagination in different ways. Check the API documentation for the API you're using for details. You need to find out things like:

- Does the API provide the URL for the next page?

- Are there API-specific limits on page size or page number?
- The structure of the data that the API returns.

Import curl command#

[curl](#) is a command line tool and library for transferring data with URLs.

You can use curl to call REST APIs. If the API documentation of the service you want to use provides curl examples, you can copy them out of the documentation and into n8n to configure the HTTP Request node.

Import a curl command:

1. Select **Import cURL command**.
2. Paste in your curl command.
3. Select **Import**. n8n loads the request configuration into the node fields. This overwrites any existing configuration.

Local File trigger#

The Local File trigger node starts a workflow when it detects changes on the file system. These changes involve a file or folder getting added, changed or deleted.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Local File Trigger's integrations](#) page.

Parameters#

You can choose what event to watch for:

Trigger On:

- **Changes to a Specific File:** triggers when the specified file changes.
 - **File to Watch:** the path to the file to watch.
- **Changes Involving a Specific Folder:** triggers when a change occurs in the selected folder.
 - **Folder to Watch:** the path of the folder to watch.
 - **Watch for:** the type of change to watch for.

Options#

Use **Options** settings to include or exclude files and folders.

- **Include Linked Files/Folders:** also watch for changes to linked files or folders.
- **Ignore:** files or paths to ignore. n8n tests the whole path, not just the filename. Supports the [Anymatch](#) syntax.
- **Max Folder Depth:** how deep into the folder structure to watch for changes.

Examples for Ignore#

Ignore a single file:

```
**/<fileName>.<suffix>  
1 # For example, **/myfile.txt  
2
```

Ignore a sub-directory of a directory you're watching:

```
**/<directoryName>/**  
1 # For example, **/myDirectory/**  
2
```

No Operation, do nothing#

The No Operation, do nothing node is used when you don't want to perform any operations. The purpose of this node is to make the workflow easier to read and understand where the flow of data stops. This can sometimes help others with a better understanding of the workflow, visually.

Webhook#

Use the Webhook node to create [webhooks](#), which can receive data from apps and services when an event occurs. It's a trigger node, which means it can start an n8n workflow. This allows services to connect to n8n and run a workflow.

You can use the Webhook node as a trigger for a workflow when you want to receive data and run a workflow based on the data. The Webhook node also supports returning the data generated at the end of a workflow. This makes it useful for build a workflow to process data and return the results, like an API endpoint.

The webhook allows you to trigger workflows from services that don't have a dedicated app trigger node.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Webhook integrations](#) list.

Build and test workflows#

While building or testing a workflow, use a test webhook URL. Using a test webhook ensures that you can view the incoming data in the editor UI, which is useful for debugging. Select **Test step** to register the webhook before sending the data to the test webhook. The test webhook stays active for 120 seconds.

When using the Webhook node on the localhost, run n8n in tunnel mode: [npm with tunnel](#) or [Docker with tunnel](#).

Production workflows#

When your workflow is ready, switch to using the production webhook URL. You can then activate your workflow, and n8n runs it automatically when an external service calls the webhook URL.

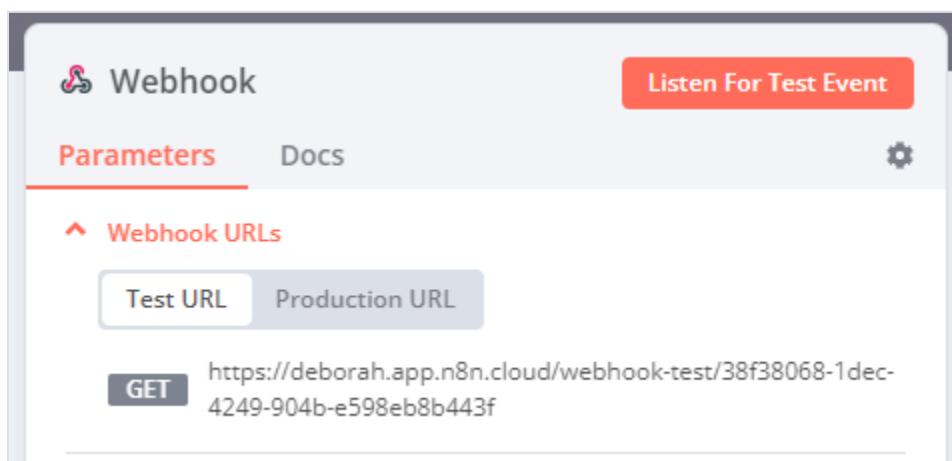
When working with a Production webhook, ensure that you have saved and activated the workflow. Data flowing through the webhook isn't visible in the editor UI with the production webhook.

Node parameters#

These are the main node configuration fields.

Webhook URLs#

The Webhook node has two URLs: test URL and production URL. n8n displays the URLs at the top of the node panel. Select **Test URL** or **Production URL** to toggle which URL n8n displays.



- **Test:** n8n registers a test webhook when you select **Listen for Test Event** or **Test workflow**, if the workflow isn't active. When you call the webhook URL, n8n displays the data in the workflow.
- **Production:** n8n registers a production webhook when you activate the workflow. When using the production URL, n8n doesn't display the data in the workflow. You can still view workflow data for a production execution: select the **Executions** tab in the workflow, then select the workflow execution you want to view.

Authentication#

You can require authentication for any service calling your webhook URL.

- **Basic Auth:** a method of authentication where calls to the webhook URL must include the username and password in the request header.
- **Header Auth:** a method of authentication where calls to the webhook URL must include the specified header parameter. For example, use this method when you want to authenticate using an API key or an access token.
- **Credential data can vary**
 - The **Credential Data** required for header auth credentials depends on the type used. For example, if you need to provide an `Authorization: Bearer <token>` header, the Credential Data Name will be `Authorization` and the Value will be `Bearer <token>`.

HTTP Method#

The Webhook node supports standard [HTTP Requests](#).

Path#

By default, this field contains a randomly generated webhook URL path, to avoid conflicts with other webhook nodes.

You can manually specify a URL path, including adding route parameters. For example, you may need to do this if you use n8n to prototype an API, and want consistent endpoint URLs.

The **Path** field can take the following formats:

- `/:variable`
- `/path/:variable`
- `/:variable/path`
- `/:variable1/path/:variable2`
- `/:variable1/:variable2`

Respond#

- **Immediately:** the Webhook node returns the response code and the message **Workflow got started**.
- **When Last Node Finishes:** the Webhook node returns the response code and the data output from the last node executed in the workflow.
- **Using 'Respond to Webhook' Node:** the Webhook node responds as defined in the [Respond to Webhook](#) node.

Response Code#

Customize the [HTTP response code](#) that the Webhook node returns upon successful execution.

Response Data#

Choose what data to include in the response body.

Node options#

Select **Add Option** to view more configuration options. The available options depend on your node parameters. Refer to the table for option availability.

- **Binary Property:** enabling this setting allows the Webhook node to receive binary data, such as an image or audio file.
- **Ignore Bots:** ignore requests from bots like link previewers and web crawlers.
- **No Response Body:** enable this to prevent n8n sending a body with the response.
- **Raw Body:** specify that the Webhook node will receive data in a raw format, such as JSON or XML.
- **Response Content-Type:** choose the format for the webhook body.
- **Response Data:** send custom data with the response.
- **Response Headers:** send additional headers in the Webhook response. Refer to [MDN Web Docs | Response header](#) to learn more about response headers.
- **Property Name:** by default, n8n returns all available data. You can choose to return a specific JSON key, so that n8n returns the value.
- **Allowed Origins (CORS):** set the permitted cross-origin domains.

Option	Required node configuration
Binary data	Either: HTTP Method > POST HTTP Method > PATCH HTTP Method > PUT
Ignore Bots	Any
No Response Body	Respond > Immediately
Raw Body	Any
Response Content-Type	Both:

Respond > When Last
Node Finishes

Response Data > First
Entry JSON

Response
Data

Respond > Immediately

Response
Headers

Any

Property
Name

Both:

Respond > When Last
Node Finishes

Response Data > First
Entry JSON

Allowed
Origins
(CORS)

Any

Use the HTTP Request node to trigger the Webhook node#

The [HTTP Request](#) node makes HTTP requests to the URL you specify.

1. Create a new workflow.
2. Add the HTTP Request node to the workflow.
3. Select a method from the **Request Method** dropdown list. For example, if you select GET as the **HTTP method** in your Webhook node, select GET as the request method in the HTTP Request node.
4. Copy the URL from the Webhook node, and paste it in the **URL** field in the HTTP Request node.
5. If using the test URL for the webhook node: execute the workflow with the Webhook node.
6. Execute the HTTP Request node.

Use curl to trigger the Webhook node#

You can use [curl](#) to make HTTP requests that trigger the Webhook node.

Note

In the examples, replace `<https://your-n8n.url/webhook/path>` with your webhook URL.

The examples make GET requests. You can use whichever HTTP method you set in **HTTP Method**.

Make an HTTP request without any parameters:

```
curl --request GET <https://your-n8n.url/webhook/path>
```

1

Make an HTTP request with a body parameter:

```
curl --request GET <https://your-n8n.url/webhook/path> --data 'key=value'
```

1

Make an HTTP request with header parameter:

```
curl --request GET <https://your-n8n.url/webhook/path> --header 'key=value'
```

1

Make an HTTP request to send a file:

```
curl --request GET <https://your-n8n.url/webhook/path> --from 'key=@/path/to/file'
```

1

Replace `/path/to/file` with the path of the file you want to send.

Send a response of type string#

By default, the response format is JSON or an array. To send a response of type string:

1. Select **Response Mode > When Last Node Finishes**.
2. Select **Response Data > First Entry JSON**.
3. Select **Add Option > Property Name**.
4. Enter the name of the property that contains the response. This defaults to data.
5. Connect an Edit Fields node to the Webhook node.
6. In the Edit Fields node, select **Add Value > String**.
7. Enter the name of the property in the **Name** field. The name should match the property name from step 4.
8. Enter the string value in the **Value** field.
9. Toggle **Keep Only Set** to on (green).

When you call the Webhook, it sends the string response from the Edit Fields node.

Respond to Webhook#

Use the Respond to Webhook node to control the response to incoming webhooks. This node works with the [Webhook](#) node.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Respond to Webhook integrations](#) list.

Runs once for the first data item

The Respond to Webhook node runs once, using the first incoming data item. Refer to [Return more than one data item](#) for more information.

How to use Respond to Webhook#

To use the Respond to Webhook node:

1. Add a [Webhook](#) node as the trigger node for the workflow.
2. In the Webhook node, set **Respond to Using 'Respond to Webhook' node**.

3. Add the Respond to Webhook node anywhere in your workflow. If you want it to return data from other nodes, place it after those nodes.

Node parameters#

Configure the node behavior using these parameters.

Respond With#

Choose what data to send in the webhook response.

- **All Incoming Items:** respond with all the JSON items from the input.
- **Binary:** respond with a binary file defined in **Response Data Source**.
- **First Incoming Item:** respond with the first incoming item's JSON.
- **JSON:** respond with a JSON object defined in **Response Body**.
- **No Data:** no response payload.
- **Redirect:** redirect to a URL set in **Redirect URL**.
- **Text:** respond with text set in **Response Body**.

Node options#

Select **Add Option** to view and set the options.

- **Response Code:** set the [response code](#) to use.

- **Response Headers:** define response headers to send.
- **Put Response in Field:** available when you respond with **All Incoming Items** or **First Incoming Item**. Set the field name for the field containing the response data.

Return more than one data item#

Deprecated in 1.22.0

n8n 1.22.0 added support for returning all data items using the **All Incoming Items** option. n8n recommends upgrading to the latest version of n8n, instead of using the workarounds described in this section.

The Respond to Webhook node runs once, using the first incoming data item. This includes when using [expressions](#). You can't force looping using the Loop node: the workflow will run, but the webhook response will still only contain the results of the first execution.

If you need to return more than one data item, you can either:

- Instead of using the Respond to Webhook node, use the **When Last Node Finishes** option in **Respond** in the Webhook node. Use this when you want to return the final data that the workflow outputs.
- Use the [Aggregate](#) node to turn multiple items into a single item before passing the data to the Respond to Webhook node. Set **Aggregate to All Item Data (Into a Single List)**.

Workflow behavior#

When using the Respond to Webhook node, workflows behave as follows:

- The workflow finishes without executing the Respond to Webhook node: it returns a standard message with a 200 status.
- The workflow errors before the first Respond to Webhook node executes: the workflow returns an error message with a 500 status.
- A second Respond to Webhook node executes after the first one: the workflow ignores it.
- A Respond to Webhook node executes but there was no webhook: the workflow ignores the Respond to Webhook node.

Tutorial: Build an AI workflow in n8n#

This tutorial introduces LangChain functionality in n8n. You can work through it with no prior knowledge of LangChain, AI, or n8n. However, if you've never used n8n before, you should also do the [Longer quickstart](#), which introduces key n8n concepts.

In this tutorial you will:

- Create a workflow from scratch. It uses the Chat Trigger to simulate and test chat interactions, ChatGPT to power the chat functionality, and a custom tool to connect to other n8n workflows.
- Understand key concepts, including:
 - The role of agents, models, and tools when building AI functionality.
 - Cluster nodes in n8n: groups of root and sub-node.

- Loading your own data into AI workflows.

Step two: Create a new workflow#

When you open n8n, you'll see the **Workflows** list.

To create a new workflow for this tutorial:

- If you have no workflows, select **Start from scratch**.
- If you already built a workflow, select **Add workflow**.

Step three: Add a trigger node#

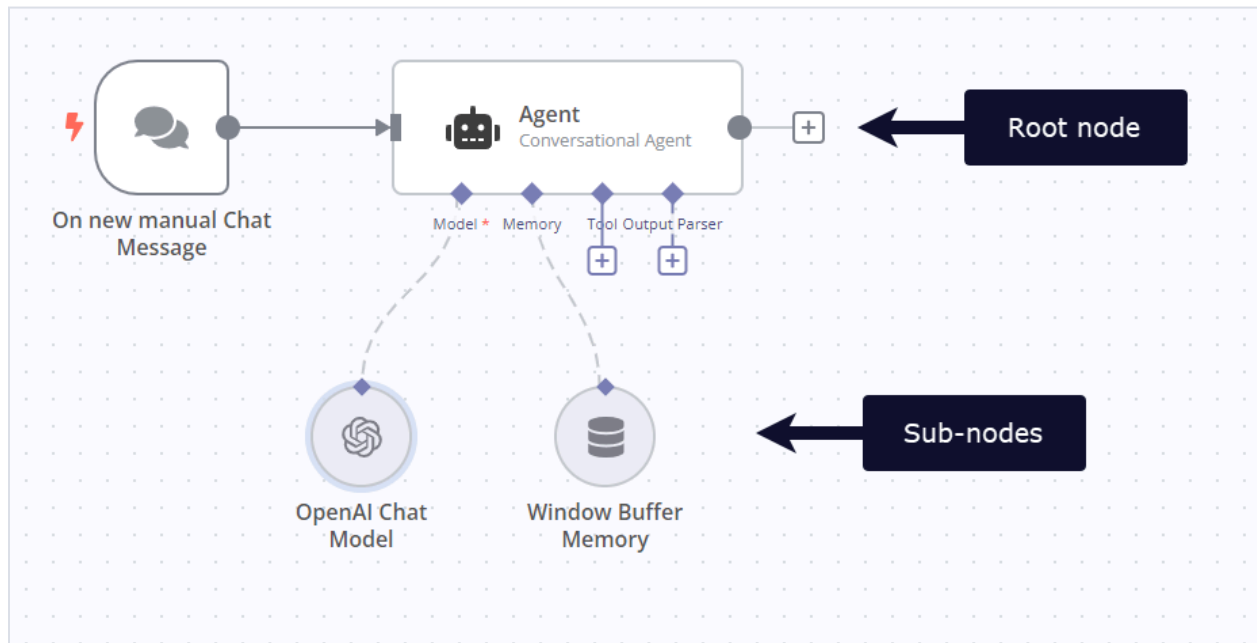
A trigger node starts a workflow. For this tutorial, use the [Chat Trigger](#). This provides a chat interface that you can use to test the workflow.


1. Select **Add first step**.
2. Search for **Chat Trigger**. n8n shows a list of nodes that match the search.
3. Select **Chat Trigger** to add the node to the canvas. n8n opens the node.
4. Close the node details view to return to the canvas.

Step four: Add an agent#

An agent takes an input, such as a user query, then uses a collection of tools to respond. In n8n the AI Agent node is a root node: a node whose behavior you can configure using sub-nodes.

Cluster nodes: root and sub-nodes



1. Select the **Add node**  connector. n8n opens the nodes panel.
2. Search for **Agent**. n8n shows a list of nodes that match the search.
3. Select **AI Agent**. n8n adds the node to the canvas and opens it.
4. Close the node details view to return to the canvas.

Step five: Chat functionality#

To provide chat interactions, the workflow needs:

- A chat model, to process chat inputs and generate responses.
- Memory, so that the workflow can remember and use previous sections of the chat.

Use the connectors on the bottom of the AI Agent node to connect the following:

- **Model:** OpenAI Chat Model. Set up the credentials for this node.
- **Credential setup steps**
 - a.
 - b.
 - c.
 - d.
 - e.
 - f.
- **Memory:** Window Buffer Memory

Step six: Get data from another n8n workflow#

One unique feature of AI in n8n is the ability to pull in data from other n8n workflows using the [Custom n8n Workflow Tool node](#). This means you can:

- Load your own custom data
- Create custom functionality in another workflow

Because n8n can connect to any service with a public API, this is a powerful tool.

This example generates some fake data in a workflow, and loads it in to the AI workflow. The AI workflow passes a parameter to the workflow generating the data, to ensure it only returns data that's intended to be public.

1. Create a new workflow, then copy in this workflow JSON:

```

1  {
2      "name": "Workflow tool example",
3      "nodes": [
4          {
5              "parameters": {
6                  "jsCode": "return [{\"fruit\": \"apple\", \"color\": \"green\",
7                  \"dataPrivacyLevel\": \"public\"},{\"fruit\": \"banana\", \"color\": \"yellow\",
8                  \"dataPrivacyLevel\": \"public\"},{\"fruit\": \"tomato\", \"color\": \"red\",
9                  \"dataPrivacyLevel\": \"private\"}]"
10             },
11             "id": "4a30b74c-ed27-4668-8e68-c2e3a630ecd9",
12             "name": "Code",
13             "type": "n8n-nodes-base.code",
14             "typeVersion": 2,
15             "position": [
16                 1180,
17                 500
18             ],
19             {
20                 "parameters": {
21                     "conditions": {
22                         "string": [
23                             {
24                                 "value1": "={{ $json.dataPrivacyLevel }}",
25                                 "value2": "={{ $('Execute Workflow
26 Trigger').item.json.visibility }}"
27                             }
28                         ]
29                     }
30                 },
31                 "id": "eb479841-8099-4537-b89b-4b9867446688",
32                 "name": "Filter",
33                 "type": "n8n-nodes-base.filter",
34                 "typeVersion": 1,
35                 "position": [
36                     1380,
37                     500
38                 ],
39                 },
40                 {
41                     "parameters": {

```

```

42         "aggregate": "aggregateAllItemData",
43         "include": "specifiedFields",
44         "fieldsToInclude": "fruit, color",
45         "options": {}
46     },
47     "id": "6d6c847d-7417-4780-8a8b-b99996cd6166",
48     "name": "Aggregate",
49     "type": "n8n-nodes-base.aggregate",
50     "typeVersion": 1,
51     "position": [
52         1600,
53         500
54     ],
55     },
56     {
57         "parameters": {},
58         "id": "1ec54b09-7533-4280-8bf2-54407d6bf134",
59         "name": "Execute Workflow Trigger",
60         "type": "n8n-nodes-base.executeWorkflowTrigger",
61         "typeVersion": 1,
62         "position": [
63             1000,
64             500
65         ]
66     },
67     ],
68     "pinData": {},
69     "connections": {
70         "Code": {
71             "main": [
72                 [
73                     {
74                         "node": "Filter",
75                         "type": "main",
76                         "index": 0
77                     }
78                 ]
79             ],
80             "Filter": {
81                 "main": [
82                     [

```

```

83         {
84             "node": "Aggregate",
85             "type": "main",
86             "index": 0
87         }
88     ]
89 },
90 "Execute Workflow Trigger": {
91     "main": [
92         [
93             {
94                 "node": "Code",
95                 "type": "main",
96                 "index": 0
97             }
98         ]
99     ]
100 }
    }
}

```

2.

The workflow uses the Code node to generate a sample data set, containing a list of fruits, their colors, and whether this information should be public or private.

3. Copy the workflow ID from workflow URL. The ID is the group of random numbers and letters at the end of the URL.
4. In the AI workflow, select the **Tool** output on the **AI Agent**. n8n opens the nodes panel.
5. Select **Custom n8n Workflow Tool**. n8n adds the node to the canvas and opens it.

6. Configure the node parameters:

- **Name:** Fruit
- **Description:** Call this tool to get a list of fruit.
- **Workflow ID:** Paste in the workflow ID that you copied from the example workflow URL.
- **Response Property Name:** data. This is the name of the item created in the Aggregate node in the example workflow.

7. Select **Add Value** in **Workflow Values** to pass information to the workflow you're calling. Configure the value as follows:

- **Name:** visibility
- **Type:** String
- **Value:** public

8. n8n makes this value available to the workflow in the output data of the trigger node in the workflow you're calling. In this example, to access the value in an expression, use `{{ $('Execute Workflow Trigger').item.json.visibility }}`. You can see this in practice in the Filter node in the sample workflow.

Step seven: Test the workflow#

You can now try out the workflow: close any open nodes, then select **Chat** to start talking with the AI. Ask `Please give me a list of fruits`. The AI Agent node executes the workflow you specified in **Workflow ID**, passing in the value of `visibility` from the **Workflow Values** field. The workflow generates some sample data, filters it to remove any items that aren't set to `public`, and returns the resulting data.

LangChain in n8n#

n8n provides a collection of nodes that implement LangChain's functionality. The LangChain nodes are configurable, meaning you can choose your preferred agent, LLM, memory, and so on. Alongside the LangChain nodes, you can connect any n8n node as normal: this means you can integrate your LangChain logic with other data sources and services.

- [Learning resources](#): n8n's documentation for LangChain assumes you're familiar with AI and LangChain concepts. This page provides links to learning resources.
- [LangChain concepts and features in n8n](#): how n8n represents LangChain concepts and features.

LangChain concepts in n8n#

This page explains how LangChain concepts and features map to n8n nodes.

This page includes lists of the LangChain-focused nodes in n8n. You can use any n8n node in a workflow where you interact with LangChain, to link LangChain to other services. The LangChain features uses n8n's [Cluster nodes](#).

n8n implements LangChain JS

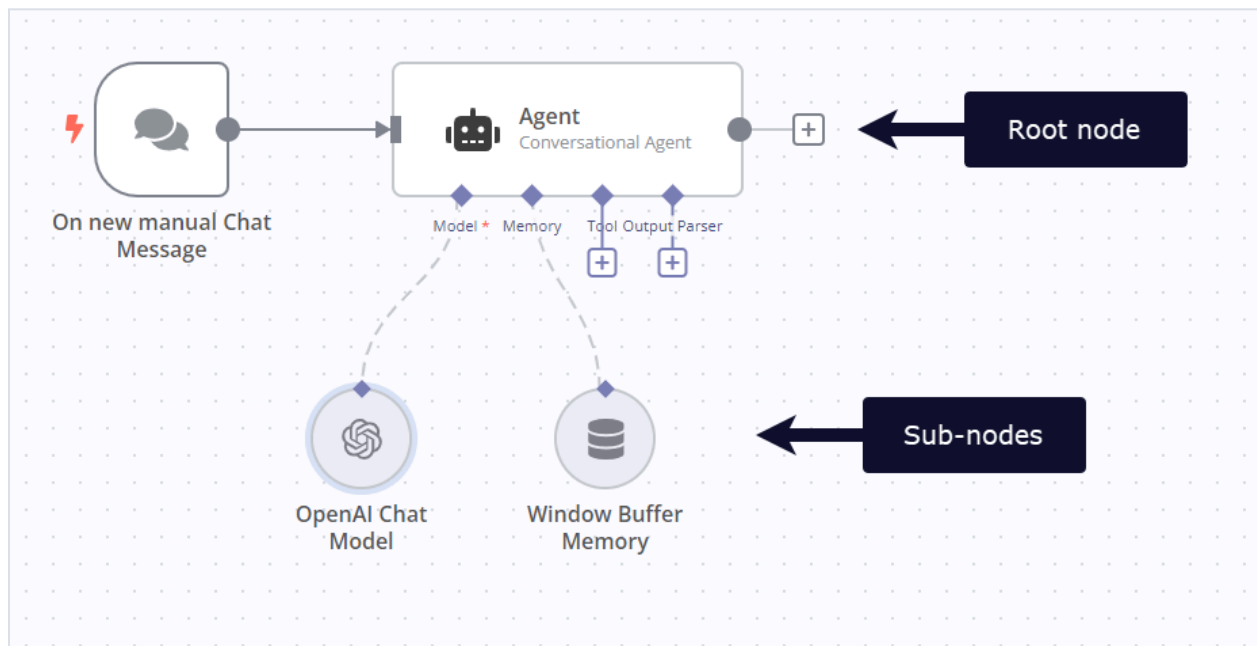
This feature is n8n's implementation of [LangChain's JavaScript framework](#).

Trigger nodes#

Chat Trigger

Cluster nodes#

Cluster nodes are node groups that work together to provide functionality in an n8n workflow. Instead of using a single node, you use a root node and one or more sub-nodes. This allows extensive configuration of node functionality.



Root nodes#

Each cluster starts with one root node.

Chains#

A chain is a series of LLMs, and related tools, linked together to support functionality that can't be provided by a single LLM alone.

Available nodes:

- [Basic LLM Chain](#)
- [Retrieval Q&A Chain](#)
- [Summarization Chain](#)

Learn more about [Chains in LangChain](#).

Agents#

An agent has access to a suite of tools, and determines which ones to use depending on the user input. Agents can use multiple tools, and use the output of one tool as the input to the next. [Source](#)

Available nodes:

- [Agent](#)

Learn more about [Agents in LangChain](#).

Vector stores#

Vector stores store embedded data, and perform vector searches on it.

- [In Memory Vector Store](#)
- [Pinecone Vector Store](#)
- [Qdrant Vector Store](#)
- [Supabase Vector Store](#)
- [Zep Vector Store](#)

Learn more about [Vector stores in LangChain](#).

Miscellaneous#

Utility nodes.

[LangChain Code](#): import LangChain. This means if there is functionality you need that n8n hasn't created a node for, you can still use it.

Sub-nodes#

Each root node can have one or more sub-nodes attached to it.

Document loaders#

Document loaders add data to your chain as documents. The data source can be a file or web service.

Available nodes:

- [Default Document Loader](#)
- [GitHub Document Loader](#)

Learn more about [Document loaders in LangChain](#).

Language models#

LLMs (large language models) are programs that analyze datasets. They're the key element of working with AI.

Available nodes:

- [Anthropic Chat Model](#)
- [AWS Bedrock Chat Model](#)

- [Cohere Model](#)
- [Google PaLM Chat Model](#)
- [Google PaLM Language Model](#)
- [Hugging Face Inference Model](#)
- [Mistral Cloud Chat Model](#)
- [Ollama Chat Model](#)
- [Ollama Model](#)
- [OpenAI Chat Model](#)
- [OpenAI Model](#)

Learn more about [Language models in LangChain](#).

Memory#

Memory retains information about previous queries in a series of queries. For example, when a user interacts with a chat model, it's useful if your application can remember and call on the full conversation, not just the most recent query entered by the user.

Available nodes:

- [Motorhead](#)
- [Redis Chat Memory](#)
- [Window Buffer Memory](#)
- [Xata](#)
- [Zep](#)

Learn more about [Memory in LangChain](#).

Output parsers#

Output parsers take the text generated by an LLM and format it to match the structure you require.

Available nodes:

- [Auto-fixing Output Parser](#)
- [Item List Output Parser](#)
- [Structured Output Parser](#)

Learn more about [Output parsers in LangChain](#).

Retrievers#

- [Contextual Compression Retriever](#)
- [MultiQuery Retriever](#)
- [Vector Store Retriever](#)
- [Workflow Retriever](#)

Text splitters#

Text splitters break down data (documents), making it easier for the LLM to process the information and return accurate results.

Available nodes:

- [Character Text Splitter](#)
- [Recursive Character Text Splitter](#)
- [Token Splitter](#)

n8n's text splitter nodes implements parts of [LangChain's text_splitter API](#).

Tools#

Utility tools.

- [Calculator](#)
- [Code Tool](#)
- [SerpAPI](#)
- [Wikipedia](#)
- [Wolfram|Alpha](#)
- [Workflow Tool](#)

Embeddings#

Embeddings capture the "relatedness" of text, images, video, or other types of information. ([source](#))

Available nodes:

- [Embeddings AWS Bedrock](#)
- [Embeddings Cohere](#)
- [Embeddings Google PaLM](#)
- [Embeddings Hugging Face Inference](#)
- [Embeddings Mistral Cloud](#)
- [Embeddings Ollama](#)
- [Embeddings OpenAI](#)

Learn more about [Text embeddings in LangChain](#).

Miscellaneous#

- [Chat Memory Manager](#)

What is a tool in AI?#

In AI, 'tools' has a specific meaning. Tools act like addons that your AI can use to access extra context or resources.

Here are a couple of other ways of expressing it:

Tools are interfaces that an agent can use to interact with the world ([source](#))

We can think of these tools as being almost like functions that your AI model can call ([source](#))

Tools in n8n#

n8n provides tool sub-nodes that you can connect to your AI agent. As well as providing some popular tools, such as [Wikipedia](#) and [SerpAPI](#), n8n provides two especially powerful tools:

- [Custom n8n Workflow Tool](#): use this to load any n8n workflow as a tool.
- [Custom Code Tool](#): write code that your agent can run.

The next three examples highlight the Custom n8n Workflow Tool:

- [Chat with Google Sheets](#)
- [Call an API to fetch data](#)
- [Set up a human fallback](#)

Agent#

Use the Agent node to set which agent type you want to use.

On this page, you'll find the node parameters for the Agent node, and links to more resources.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Agent integrations](#) page.

Connect a tool

You must connect at least one tool sub-node.

n8n provides several agents. The conversational agent is the default. n8n recommends using this for most use cases: it's the easiest to configure and will handle most scenarios.

Conversational Agent#

This agent is optimised for conversation allowing it to chat with the user.

You can use this agent with the [On new manual Chat Message](#) node. Attach a memory sub-node so that users can have an ongoing conversation with multiple queries. Memory doesn't persist between sessions.

Parameters#

Text#

The input from the chat. This is the user's query, also known as the prompt.

Options#

Human Message#

Tell the agent about the tools it can use, and add context to the user's input.

You must include:

- `{tools}`: a LangChain expression. Provides a string of the tools you've connected to the Agent.
- `{format_instructions}`: a LangChain expression. Provides the schema or format from the output parser node you've connected.
- `{{input}}`: a LangChain variable. The user's prompt. Populated with the value of the **Text** parameter.

Example:

```
1  TOOLS
2  -----
3  Assistant can ask the user to use tools to look up information that may be helpful in
4  answering the users original question. The tools the human can use are:
5  {tools}
6
7  {format_instructions}
8
9  USER'S INPUT
10 -----
11 Here is the user's input (remember to respond with a markdown code snippet of a
12 json blob with a single action, and NOTHING else):

{{input}}
```


System Message#

Send a message to the agent before the conversation starts. Use this to guide the agent's decision-making.

Max Iterations#

How many times the model should run to try and generate a good answer for the user's prompt.

OpenAI Functions Agent#

Use the OpenAI Functions Agent node to use an [OpenAI functions model](#). These are models that detect when a function should be called and respond with the inputs that should be passed to the function.

You can use this agent with the [On new manual Chat Message](#) node. Attach a memory sub-node so that users can have an ongoing conversation with multiple queries. Memory doesn't persist between sessions.

You must use the [OpenAI Chat Model](#) with this agent.

Parameters#

Text#

The input from the chat. This is the user's query, also known as the prompt.

ReAct Agent#

The ReAct Agent node implements [ReAct](#) logic. ReAct (reasoning and action) brings together the reasoning powers of chain-of-thought prompting and action plan generation.

No memory

The ReAct agent doesn't support memory sub-nodes. This means it can't recall previous prompts, or simulate an ongoing conversation.

Parameters#

Text#

The input from the chat. This is the user's query, also known as the prompt.

Options#

Use the options to create a message to send to the agent at the start of the conversation. The message type depends on the model you're using.

- Chat models: these models have the concept of three components interacting (AI, system, and human). They can receive system messages and human messages (prompts).

- Instruct models: these models don't have the concept of separate AI, system, and human components. They receive one body of text, the instruct message.

Human Message Template#

Use this to extend the user prompt. This is a way for the agent to pass information from one iteration to the next.

Available LangChain expressions:

- {input}: contains the user prompt.
- {agent_scratchpad}: information to remember for the next iteration.

Prefix Message#

Add text to prefix the tools list at the start of the conversation. You don't need to add the list of tools. LangChain automatically adds this.

Suffix Message for Chat Model#

The final part of the system message. Sent before the user prompt.

Suffix Message for Regular Model#

The final part of the message. Sent before the user prompt.

SQL Agent#

The SQL Agent uses a SQL database as a data source. The agent builds a SQL query based on the natural language query in the prompt.

Postgres and MySQL Agents

If you are using Postgres or MySQL this doesn't support the tunnel options you can set in the credential.

Data Source#

Options:

- MySQL
- SQLite
- Postgres

SQLite

To use SQLite you will need to use a [Read/Write File From Disk](#) node before the Agent to read your SQLite file.

Prompt#

The query to run on the data.

Options#

Use the options to refine the agent's behavior.

- Ignored Tables: comma-separated list of tables to ignore from the database. If empty, no tables are ignored.

- Include Sample Rows: number of sample rows to include in the prompt to the agent. It helps the agent to understand the schema of the database but it also increases the amount of tokens used.
- Included Tables: comma-separated list of tables to include in the database. If empty, all tables are included.
- Prefix Prompt: set the initial message.
- Suffix Prompt: set the final part of the message. Includes the user input and agent scratchpad.
- Top K: number of database results the agent should keep in its context.

You can view prompt examples in the node.

Related resources#

View [example workflows and related content](#) on n8n's website.

Refer to [LangChain's documentation on agents](#) for more information about the service.

View n8n's [Advanced AI](#) documentation.

Custom n8n Workflow Tool#

The Workflow Tool node is a tool that allows an agent to run another n8n workflow and fetch its output data.

On this page, you'll find the node parameters for the Workflow Tool node, and links to more resources.

Examples and templates

For usage examples and templates to help you get started, refer to n8n's [Custom n8n Workflow Tool integrations](#) page.

Parameter resolution in sub-nodes

Sub-nodes behave differently to other nodes when processing multiple items using an expression.

Most nodes, including root nodes, take any number of items as input, process these items, and output the results. You can use expressions to refer to input items, and the node resolves the expression for each item in turn. For example, given an input of five `name` values, the expression `{{ $json.name }}` resolves to each name in turn.

In sub-nodes, the expression always resolves to the first item. For example, given an input of five `name` values, the expression `{{ $json.name }}` always resolves to the first name.

Node parameters#

Name#

Give your custom code a name. It can't contain whitespace or special characters.

Description#

Give your custom code a description. This tells the agent when to use this tool. For example:

Call this tool to get a random color. The input should be a string with comma separated names of colors to exclude.

Source#

Tell n8n which workflow to call. You can choose either:

- **Database**, then enter a workflow ID.
- **Parameter**, then copy in a complete [workflow JSON](#).

Response Property Name#

This must match the name of the output property in the workflow you're calling.

Workflow Values#

Set values to pass to the workflow you're calling.

These values appear in the output data of the trigger node in the workflow you call. You can access these values in expressions in the workflow. For example, if you have:

- **Workflow Values** with a **Name** of myCustomValue
- A workflow with an Execute Workflow Trigger node as its trigger

The expression to access the value of myCustomValue is `{{ $('Execute Workflow Trigger').item.json.myCustomValue }}`.

Related resources#

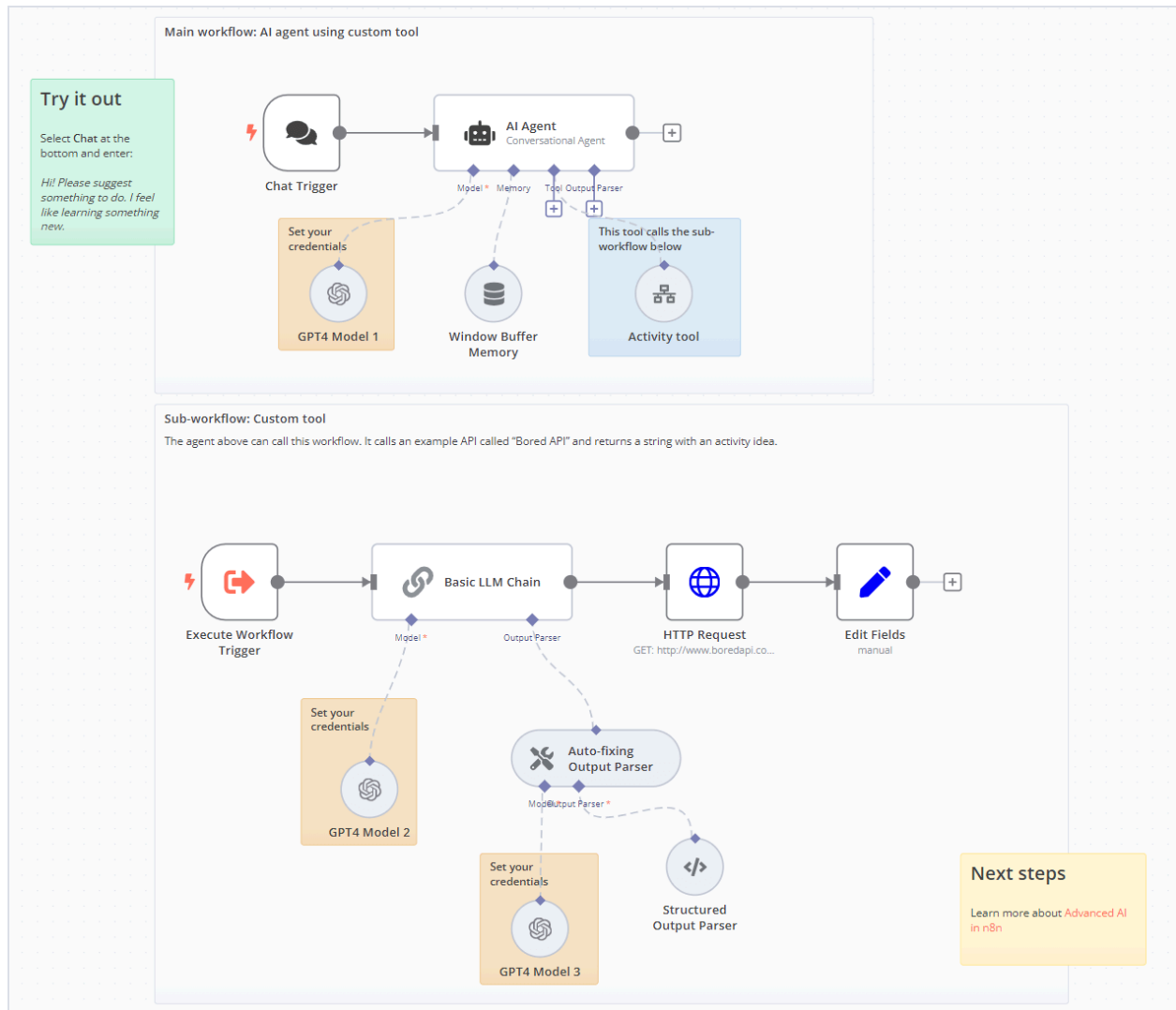
View [example workflows and related content](#) on n8n's website.

Refer to [LangChain's documentation on tools](#) for more information about tools in LangChain.

View n8n's [Advanced AI](#) documentation.

Call an API to fetch data#

Use n8n to bring data from any API to your AI. This workflow uses the [Chat Trigger](#) to provide the chat interface, and the [Custom n8n Workflow Tool](#) to call a second workflow that calls the API. The second workflow uses AI functionality to refine the API request based on the user's query.



[Download the example workflow](#)


Key features#

This workflow uses:

- [Chat Trigger](#): start your workflow and respond to user chat interactions. The node provides a customizable chat interface.
- [Agent](#): the key piece of the AI workflow. The Agent interacts with other components of the workflow and makes decisions about what tools to use.
- [Custom n8n Workflow Tool](#): plug in n8n workflows as custom tools. In AI, a tool is an interface the AI can use to interact with the world (in this case, the data provided by your workflow). It allows the AI model to access information beyond its built-in dataset.
- A [Basic LLM Chain](#) with an [Auto-fixing Output Parser](#) and [Structured Output Parser](#) to read the user's query and set parameters for the API call based on the user input.

Using the example#

To load the template into your n8n instance:

1. [Download the workflow JSON file](#).
2. Open a new workflow in your n8n instance.
3. Copy in the JSON, or select **Workflow menu**  > **Import from file....**

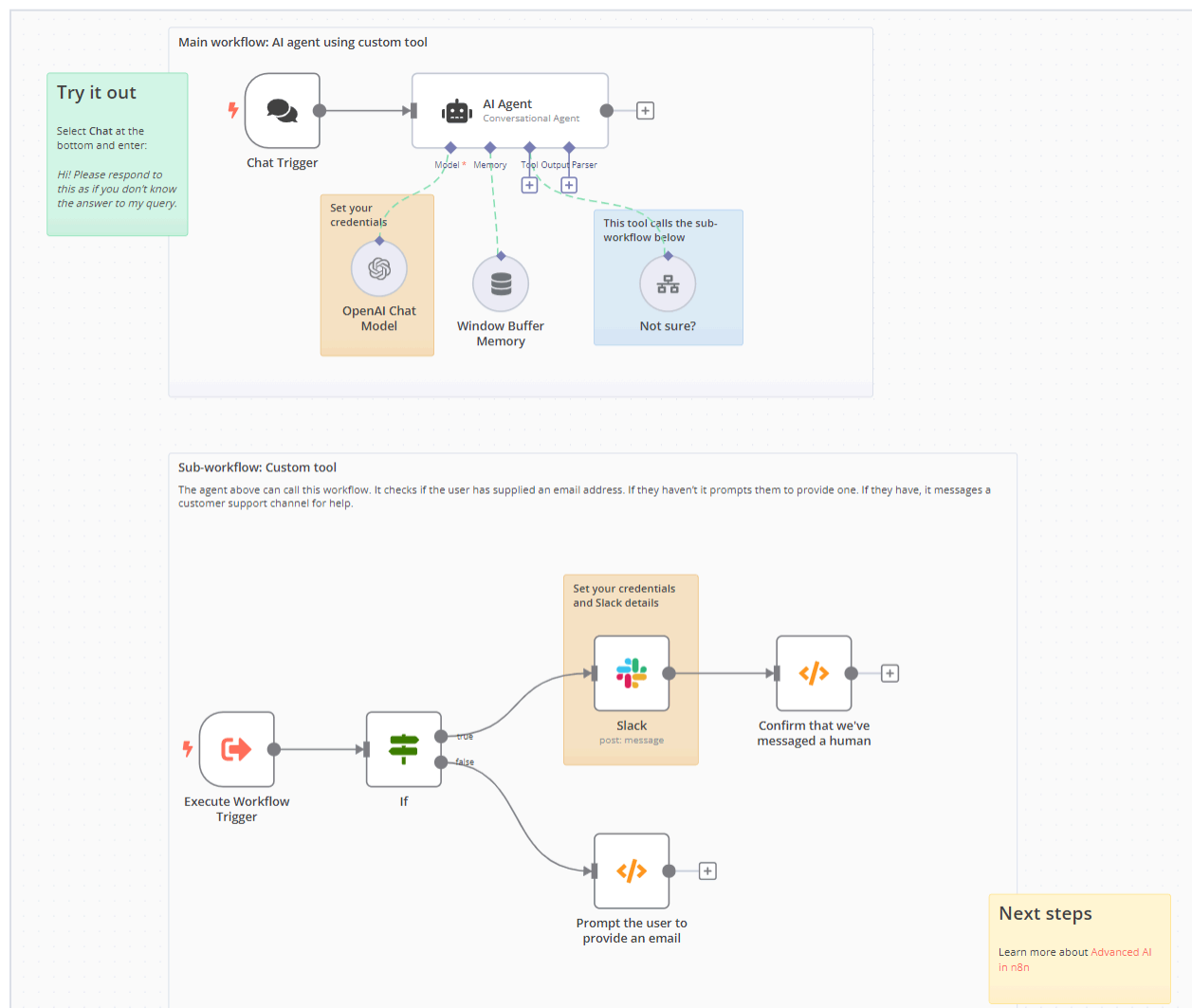
The example workflows use Sticky Notes to guide you:

- Yellow: notes and information.
- Green: instructions to run the workflow.
- Orange: you need to change something to make the workflow work.
- Blue: draws attention to a key feature of the example.

Have a human fallback for AI workflows#

This is a workflow that tries to answer user queries using the standard GPT-4 model. If it can't answer, it sends a message to Slack to ask for human help. It prompts the user to supply an email address.

This workflow uses the [Chat Trigger](#) to provide the chat interface, and the [Custom n8n Workflow Tool](#) to call a second workflow that handles checking for email addresses and sending the Slack message.



[Download the example workflow](#)


Key features#

This workflow uses:

- [Chat Trigger](#): start your workflow and respond to user chat interactions. The node provides a customizable chat interface.
- [Agent](#): the key piece of the AI workflow. The Agent interacts with other components of the workflow and makes decisions about what tools to use.
- [Custom n8n Workflow Tool](#): plug in n8n workflows as custom tools. In AI, a tool is an interface the AI can use to interact with the world (in this case, the data provided by your workflow). It allows the AI model to access information beyond its built-in dataset.

Using the example#

To load the template into your n8n instance:

1. [Download the workflow JSON file](#).
2. Open a new workflow in your n8n instance.
3. Copy in the JSON, or select **Workflow menu**  > **Import from file....**

The example workflows use Sticky Notes to guide you:

- Yellow: notes and information.
- Green: instructions to run the workflow.
- Orange: you need to change something to make the workflow work.
- Blue: draws attention to a key feature of the example.

Code in n8n#

n8n is a low-code tool. This means you can do a lot without code, then add code when needed.

Code in your workflows#

There are two places in your workflows where you can use code:

- **Expressions**

Use expressions to transform [data](#) in your nodes. You can use JavaScript in expressions, as well as n8n's [Built-in methods and variables](#) and [Data transformation functions](#).

[Expressions](#)

- **Code node**

Use the Code node to add JavaScript or Python to your workflow.

[Code node](#)

Expressions#

In version 1.9.0 n8n changed the templating language for expressions

If you have issues with expressions in 1.9.0:

- Please report the issue on the [forums](#).
- Self-hosted users can switch back to RiotTmpl: set `N8N_EXPRESSION_EVALUATOR` to `tmpl`. Refer to [Environment variables](#) for more information on configuring self-hosted n8n.

Expressions allow you to set node parameters dynamically based on data from:

- Previous nodes
- The workflow
- Your n8n environment

You can execute JavaScript within an expression.

n8n created and uses a templating language called [Tournament](#), and extends it with [custom methods and variables](#) and [data transformation functions](#) that help with common tasks, such as retrieving data from other nodes, or accessing metadata.

n8n supports two libraries:

- [Luxon](#), for working with data and time.
- [JMESPath](#), for querying JSON.

No Python support

Expressions must use JavaScript.

Data in n8n

When writing expressions, it's helpful to understand data structure and behavior in n8n. Refer to [Data](#) for more information on working with data in your workflows.

Writing expressions#

To use an expression to set a parameter value:

1. Hover over the parameter where you want to use an expression.
2. Select **Expressions** in the **Fixed/Expression** toggle.
3. Write your expression in the parameter, or select **Open expression editor**  to open the expressions editor. If you use the expressions editor, you can browse the available data in the **Variable selector**. All expressions have the format `{{ your expression here }}`.

Example: Get data from webhook body#

Consider the following scenario: you have a webhook trigger that receives data through the webhook body. You want to extract some of that data for use in the workflow.

Your webhook data looks similar to this:

```
1  [  
2    {  
3      "headers": {  
4        "host": "n8n.instance.address",  
5        ...  
6      },  
7      "params": {},  
8      "query": {},  
9      "body": {  
10       "name": "Jim",  
11       "age": 30,  
12       "city": "New York"  
13     }  
14   }  
15 ]
```

In the next node in the workflow, you want to get just the value of `city`. You can use the following expression:

```
{{ $json.body.city }}
```

1

This expression:

1. Accesses the incoming JSON-formatted data using n8n's custom `$json` variable.
2. Finds the value of `city` (in this example, "New York"). Note that this example uses JMESPath syntax to query the JSON data. You can also write this expression as `{{ $json['body']['city'] }}`.

Example: Writing longer JavaScript#

An expression contains one line of JavaScript. This means you can do things like variable assignments or multiple standalone operations.

To understand the limitations of JavaScript in expressions, and start thinking about workarounds, look at the following two pieces of code. Both code examples use the Luxon date and time library to find the time between two dates in months, and encloses the code in handlebar brackets, like an expression.

However, the first example isn't a valid n8n expression:

```
1 // This example is split over multiple lines for readability
2 // It's still invalid when formatted as a single line
3 {{
4   function example() {
5     let end = DateTime.fromISO('2017-03-13');
6     let start = DateTime.fromISO('2017-02-13');
7     let diffInMonths = end.diff(start, 'months');
8     return diffInMonths.toObject();
9   }
10  example();
11 }}
```

While the second example is valid:

```
1 {{DateTime.fromISO('2017-03-13').diff(DateTime.fromISO('2017-02-13'),
   'months').toObject()}}
```

Data transformation functions#

Data transformation functions are helper functions to make data transformation easier in expressions.

JavaScript in expressions

You can use any JavaScript in expressions. Refer to [Expressions](#) for more information.

For a list of available functions, refer to the page for your data type:

- [Arrays](#)
- [Dates](#)
- [Numbers](#)
- [Objects](#)
- [Strings](#)

Usage#

Data transformation functions are available in the expressions editor.

The syntax is:

```
{{ dataItem.function() }}
```

1

For example, to check if a string is an email:

```
1 {{ "example@example.com".isEmail() }}  
2  
3 // Returns true
```

Arrays#

A reference document listing built-in convenience functions to support data transformation in expressions for arrays.

JavaScript in expressions

You can use any JavaScript in expressions. Refer to [Expressions](#) for more information.

average () : Number #

Returns the value of elements in an array

chunk (size: Number) : Array #

Splits arrays into chunks with a length of size

Function parameters#

sizeREQUIREDNUMBER

The size of each chunk.

compact() : Array #

Removes empty values from the array.

difference(arr: Array) : Array #

Compares two arrays. Returns all elements in the base array that aren't present in arr.

Function parameters#

arrREQUIREDARRAY

The array to compare to the base array.

intersection(arr: Array): Array #

Compares two arrays. Returns all elements in the base array that are present in arr.

Function parameters#

arrREQUIREDARRAY

The array to compare to the base array.

first(): Array item #

Returns the first element of the array.

isEmpty() : Boolean #

Checks if the array doesn't have any elements.

isNotEmpty() : Boolean #

Checks if the array has elements.

last() : Array item #

Returns the last element of the array.

max() : Number #

Returns the highest value in an array.

merge(arr: Array) : Array #

Merges two Object-arrays into one array by merging the key-value pairs of each element.

Function parameters#

arrREQUIREDARRAY

The array to merge into the base array.

min() : Number #

Gets the minimum value from a number-only array.

pluck(fieldName?: String) : Array #

Returns an array of Objects where keys equal the given field names.

Function parameters#

fieldnameOPTIONALSTRING

The key(s) you want to retrieve. You can enter as many keys as you want, as comma-separated strings.

randomItem() : Array item #

Returns a random element from an array.

removeDuplicates (key?: String) : Array #

Removes duplicates from an array.

Function parameters#

keyOPTIONALSTRING

A key, or comma-separated list of keys, to check for duplicates.

**renameKeys (from: String, to: String) :
Array #**

Renames all matching keys in the array. You can rename more than one key by entering a series of comma separated strings, in the pattern oldKeyName, newKeyName.

Function parameters#

fromREQUIREDSTRING

The key you want to rename.

toREQUIREDSTRING

The new name.

smartJoin(keyField: String, nameField: String): Array #

Operates on an array of objects where each object contains key-value pairs. Creates a new object containing key-value pairs, where the key is the value of the first pair, and the value is the value of the second pair. Removes non-matching and empty values and trims any whitespace before joining.

Function parameters#

keyfieldREQUIREDSTRING

The key to join.

namefieldREQUIREDSTRING

The value to join.

Example

BASIC USAGE

```
1 // Input
2 {{ [{"type": "fruit", "name": "apple"}, {"type": "vegetable", "name": "carrot"}
3 ].smartJoin("type", "name") }}
4 // Output
5 [Object: {"fruit": "apple", "vegetable": "carrot"}]
```

sum() : Number #

Returns the total sum all the values in an array of parsable numbers.

union(arr: Array) : Array #

Concatenates two arrays and then removes duplicate.

Function parameters#

arrREQUIREDARRAY

The array to compare to the base array.

unique(key?: String) : Array #

Remove duplicates from an array.

Function parameters#

keyOPTIONALSTRING

A key, or comma-separated list of keys, to check for duplicates.

Dates#

A reference document listing built-in convenience functions to support data transformation in expressions for dates.

JavaScript in expressions

You can use any JavaScript in expressions. Refer to [Expressions](#) for more information.

beginningOf (unit?: DurationUnit): Date

#

Transforms a Date to the start of the given time period. Returns either a JavaScript Date or Luxon Date, depending on input.

Function parameters#

unitOPTIONALSTRING ENUM

A valid string specifying the time unit.

Default: week

One of: second, minute, hour, day, week, month, year

endOfMonth () : Date #

Transforms a Date to the end of the month.

extract (datePart?: DurationUnit) :

Number #

Extracts the part defined in datePart from a Date. Returns either a JavaScript Date or Luxon Date, depending on input.

Function parameters#

datepartOPTIONALSTRING ENUM

A valid string specifying the time unit.

Default: week

One of: second, minute, hour, day, week, month, year

format(fmt: TimeFormat): String #

Formats a Date in the given structure

Function parameters#

fmtREQUIREDSTRING ENUM

A valid string specifying the time format. Refer to [Luxon | Table of tokens](#) for formats.

```
isBetween(date1: Date | DateTime,  
date2: Date | DateTime): Boolean #
```

Checks if a Date is between two given dates.

Function parameters**#**

date1**REQUIRED**DATE OR DATETIME

The first date in the range.

date2**REQUIRED**DATE OR DATETIME

The last date in the range.

isDst() : Boolean #

Checks if a Date is within Daylight Savings Time.

**isInLast(n?: Number, unit?:
DurationUnit) : Boolean #**

Checks if a Date is within a given time period.

Function parameters#

nOPTIONALNUMBER

The number of units. For example, to check if the date is in the last nine weeks, enter 9.

Default: 0

unitOPTIONALSTRING ENUM

A valid string specifying the time unit.

Default: minutes

One of: second, minute, hour, day, week, month, year

isWeekend() : Boolean #

Checks if the Date falls on a Saturday or Sunday.

**minus(n: Number, unit?: DurationUnit) :
Date #**

Subtracts a given time period from a Date. Returns either a JavaScript Date or Luxon Date, depending on input.

Function parameters#

nREQUIREDNUMBER

The number of units. For example, to subtract nine seconds, enter 9 here.

unitOPTIONALSTRING ENUM

A valid string specifying the time unit.

Default: milliseconds

One of: second, minute, hour, day, week, month, year

plus(n: Number, unit?: DurationUnit):

Date #

Adds a given time period to a Date. Returns either a JavaScript Date or Luxon Date, depending on input.

Function parameters#

nREQUIREDNUMBER

The number of units. For example, to add nine seconds, enter 9 here.

unitOPTIONALSTRING ENUM

A valid string specifying the time unit.

Default: milliseconds

One of: second, minute, hour, day, week, month, year

Numbers#

A reference document listing built-in convenience functions to support data transformation in expressions for numbers.

JavaScript in expressions

You can use any JavaScript in expressions. Refer to [Expressions](#) for more information.

ceil() : Number #

Rounds up a number to a whole number.

floor() : Number #

Rounds down a number to a whole number.

**format(locales?: LanguageCode,
options?: FormatOptions): String #**

This is a wrapper around [Intl.NumberFormat\(\)](#). Returns a formatted string of a number based on the given LanguageCode and FormatOptions. When no arguments are given, transforms the number in a like format 1.234.

Function parameters#

localesOPTIONALSTRING

An IETF BCP 47 language tag.

Default: en-US

`options` **OPTIONAL** OBJECT

Configure options for number formatting. Refer to [MDN | Intl.NumberFormat\(\)](#) for more information.

`isEven()` : Boolean <#>

Returns true if the number is even. Only works on whole numbers.

`isOdd()` : Boolean <#>

Returns true if the number is odd. Only works on whole numbers.

round(decimalPlaces?: Number): Number #

Returns the value of a number rounded to the nearest whole number, unless a decimal place is specified.

Function parameters#

decimalplacesOPTIONALNUMBER

How many decimal places to round to.

Default: 0

Objects#

A reference document listing built-in convenience functions to support data transformation in expressions for objects.

JavaScript in expressions

You can use any JavaScript in expressions. Refer to [Expressions](#) for more information.

isEmpty() : Boolean #

Checks if the Object has no key-value pairs.

merge(object: Object) : Object #

Merges two Objects into a single Object using the first as the base Object. If a key exists in both Objects, the key in the base Object takes precedence.

Function parameters#

objectREQUIREDOBJECT

The Object to merge with the base Object.

hasField(fieldName: String): Boolean #

Checks if the Object has a given field. Only top-level keys are supported.

Function parameters#

fieldnameREQUIREDSTRING

The field to search for.

removeField(key: String): Object #

Removes a given field from the Object

Function parameters#

keyREQUIREDSTRING

The field key of the field to remove.

removeFieldsContaining(value: String) :
Object #

Removes fields with a given value from the Object.

Function parameters#

valueREQUIREDSTRING

The field value of the field to remove.

keepFieldsContaining(value: String) :
Object #

Removes fields that do not match the given value from the Object.

Function parameters#

valueREQUIREDSTRING

The field value of the field to keep.

compact() : Object #

Removes empty values from an Object.

urlEncode() : String #

Transforms an Object into a URL parameter list. Only top-level keys are supported.

Strings#

A reference document listing built-in convenience functions to support data transformation in expressions for strings.

JavaScript in expressions

You can use any JavaScript in expressions. Refer to [Expressions](#) for more information.

`extractDomain(): String #`

Extracts a domain from a string containing a valid URL. Returns undefined if none is found.

`extractEmail(): String #`

Extracts an email from a string. Returns undefined if none is found.

`extractUrl(): String #`

Extracts a URL from a string. Returns undefined if none is found.

hash(algo?: Algorithm): String #

Returns a string hashed with the given algorithm.

Function parameters#

algoOPTIONALSTRING ENUM

Which hashing algorithm to use.

Default: md5

One of: md5, base64, sha1, sha224, sha256, sha384, sha512, sha3, ripemd160

isDomain(): Boolean #

Checks if a string is a domain.

isEmail() : Boolean #

Checks if a string is an email.

isEmpty() : Boolean #

Checks if a string is empty.

isNotEmpty() : Boolean #

Checks if a string has content.

isNumeric() : Boolean #

Checks if a string only contains digits.

isUrl() : Boolean #

Checks if a string is a valid URL.

quote(mark?: String) : String #

Returns a string wrapped in the quotation marks. Default quotation is ".

Function parameters#

markOPTIONALSTRING

Which quote mark style to use.

Default: "

removeMarkdown() : String #

Removes Markdown formatting from a string.

replaceSpecialChars() : String #

Replaces non-ASCII characters in a string with an ASCII representation.

removeTags() : String #

Remove tags, such as HTML or XML, from a string.

toDate() : Date #

Converts a string to a date.

toDecimalNumber() : Number #

See [toFloat](#)

toFloat() : Number #

Converts a string to a decimal number.

toInt() : Number #

Converts a string to an integer.

toSentenceCase() : String #

Formats a string to sentence case.

toSnakeCase() : String #

Formats a string to snake case.

toTitleCase(): String #

Formats a string to title case. Will not change already uppercase letters to prevent losing information from acronyms and trademarks such as iPhone or FAANG.

toWholeNumber(): Number #

Converts a string to a whole number.

**urlDecode(entireString?: Boolean):
String #**

Decodes a URL-encoded string. It decodes any percent-encoded characters in the input string, and replaces them with their original characters.

Function parameters#

entirestringOPTIONALBOOLEAN

Whether to decode characters that are part of the URI syntax (true) or not (false).

urlEncode (entireString?: Boolean) :
String #

Encodes a string to be used/included in a URL.

Function parameters#

entirestringOPTIONALBOOLEAN

Whether to encode characters that are part of the URI syntax (true) or not (false).

Pagination in the HTTP Request node#

The HTTP Request node supports pagination. This page provides some example configurations, including using the [HTTP node variables](#).

Refer to [HTTP Request](#) for more information on the node.

API differences

Different APIs implement pagination in different ways. Check the API documentation for the API you're using for details. You need to find out things like:

- Does the API provide the URL for the next page?
- Are there API-specific limits on page size or page number?
- The structure of the data that the API returns.

An example API#

You can try these examples with any API that supports pagination. If you need a free option, [Punk API](#) is free with rate limits, and doesn't require authorization.

Enable pagination#

In the HTTP Request node, select **Add Option > Pagination**.

Use a URL from the response to get the next page using `$response#`

If the API returns the URL of the next page in its response:

1. Set **Pagination Mode** to **Response Contains Next URL**. n8n displays the parameters for this option.
2. In **Next URL**, use an expression to set the URL. The exact expression depends on the data returned by your API. For example, if the API includes a parameter called `next-page` in the response body:

```
{{ $response.body["next-page"] }}
```

Get the next page by number using `$pageCount#`

If the API you're using allows you to target a particular page by number:

1. Set **Pagination Mode** to **Update a Parameter in Each Request**.
2. Set **Type** to **Query**.
3. Enter the **Name** of the query parameter. This depends on your API. For example, Punk API uses a query parameter named `page` to set the page. So **Name** would be `page`.

4. Hover over **Value** and toggle **Expression** on.
5. Enter `{{ $pageCount + 1 }}`

`$pageCount` is the number of pages the HTTP Request node has fetched. It starts at zero. Most API pagination counts from one (the first page is page one). This means that adding +1 to `$pageCount` means the node fetches page one on its first loop, page two on its second, and so on.

Set the page size in the query#

If the API you're using supports choosing the page size in the query:

1. Select **Add Parameter** in **Query Parameters**. This is in the main node parameters, not in the option settings.
2. Enter the **Name** of the query parameter. This depends on your API. For example, Punk API uses a query parameter named `per_page` to set page size. So **Name** would be `per_page`.
3. In **Value**, enter your page size.