Create a Coda account.

Supported authentication methods

- API access token

Related resources

Refer to Coda's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- An API **Access Token**: Generate an API access token in your Coda **Account settings**.

Cohere credentials

You can use these credentials to authenticate the following nodes:

- Cohere
- Cohere Chat
- Reranker Cohere
- Embeddings Cohere

Prerequisites

Create a Cohere account.

You'll need an account with the following access:

- For the Trial API, you need User or Owner permissions.
- For Production API, you need Owner permissions.

Refer to Cohere Teams and Roles documentation for more information.

Supported authentication methods

- API key

Related resources

Refer to Cohere's documentation for more information about the service.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

To configure this credential, you'll need:

- An **API Key**: To generate a Cohere API key, go to the API Keys section of your Cohere dashboard.

Contentful credentials

You can use these credentials to authenticate the following nodes:

- Contentful

Prerequisites

- Create a Contentful account.
- Create a Contentful space.

Supported authentication methods

- API access token

Related resources

Refer to Contentful's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- Your Contentful **Space ID**: The Space ID displays as you generate the tokens; You can also refer to the Contentful Find space ID documentation to view the Space ID.
- A **Content Delivery API Access Token**: Required if you want to use the Content Delivery API. Leave blank if you don't intend to use this API.
- A **Content Preview API Access Token**: Required if you want to use the Content Preview API. Leave blank if you don't intend to use this API.

View and generate access tokens in Contentful in **Settings > API keys**. Contentful generates tokens for both Content Delivery API and Content Preview API as part of a single key. Refer to the Contentful API authentication documentation for detailed instructions.

ConvertAPI credentials

–8<– “_snippets/integrations/builtin/credentials/cred-only-statement.md”

Supported authentication methods

- API Token

Related resources

Refer to ConvertAPI’s API documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n’s website.

Using API Token

To configure this credential, you’ll need a ConvertAPI account and:

- An **API Token** to authenticate requests to the service.

Refer to ConvertAPI’s API documentation for more information about authenticating to the service.

ConvertKit credentials

You can use these credentials to authenticate the following nodes:

- ConvertKit
- ConvertKit Trigger

Prerequisites

Create a ConvertKit account.

Supported authentication methods

- API key

Related resources

Refer to ConvertKit’s API documentation for more information about the service.

Using API key

To configure this credential, you’ll need:

- An **API Secret**: Access your ConvertKit API key in **Account**

Settings > Advanced. Add this key as the **API Secret** in n8n.

Copper credentials

You can use these credentials to authenticate the following nodes:

- Copper
- Copper Trigger

Prerequisites

Create a Copper account at the **Professional** or **Business** plan level.

Supported authentication methods

- API key

Related resources

Refer to Copper's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to the Copper Generating an API key documentation for information on generating an API key.
- An **Email** address: Use the API key creator's email address

---

Cortex credentials

You can use these credentials to authenticate the following nodes:

- Cortex

Prerequisites

Install Cortex on your server.

Supported authentication methods

- API key

Related resources

Refer to Cortex's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to the Cortex API Authentication documentation for detailed instructions on generating API keys.
- The URL/Server Address for your **Cortex Instance** (defaults to `http://<your_server_address>:9001/`)

CrateDB credentials

You can use these credentials to authenticate the following nodes:

- CrateDB

Prerequisites

An available instance of CrateDB.

Supported authentication methods

- account connection

Related resources

Refer to CrateDB's documentation for more information about the service.

Using account connection

To configure this credential, you'll need:

- Your **Host** name
- Your **Database** name
- A **User** name
- A user **Password**
- To set the **SSL** parameter. Refer to the CrateDB Secured Communications (SSL/TLS) documentation for more information. The options n8n supports are:
  - Allow
  - Disable
  - Require
- A **Port** number

Refer to the Connect to a CrateDB cluster documentation for detailed instructions on these fields and their default values.

---

crowd.dev credentials

You can use these credentials to authenticate the following nodes:

- crowd.dev
- crowd.dev Trigger

Prerequisites

Create a working instance of crowd.dev.

Supported authentication methods

- API key

Related resources

Refer to crowd.dev's documentation for more information about the service, and their API documentation for working with the API.

Using API key

To configure this credential, you'll need:

- A **URL**:
  - If your crowd.dev instance is hosted on crowd.dev, keep the default of `https://app.crowd.dev`.
  - If your crowd.dev instance is self-hosted, use the URL you use to access your crowd.dev instance.
- Your crowd.dev **Tenant ID**: Displayed in the **Settings** section of the crowd.dev app
- An API **Token**: Displayed in the **Settings** section of the crowd.dev app

Refer to the crowd.dev API documentation for more detailed instructions.

CrowdStrike credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a CrowdStrike account.

Authentication methods

- OAuth2

Related resources

Refer to CrowdStrike's documentation for more information about the service. Their documentation is behind a log in, so you must log in to your account on their website to access the API documentation.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using OAuth2

To configure this credential, you'll need:

- The **URL** of your CrowdStrike instance
- A **Client ID**: Generated by creating a new API Client in Crowdstrike in **Support > API Clients and Keys**.
- A **Client Secret**: Generated by creating a new API Client in Crowdstrike in **Support > API Clients and Keys**.

When setting up your API client, grant it the `usermgmt:read` scope. n8n relies on this to test that the credential is working.

A broad outline of the appropriate steps is available publicly at the CrowdStrike blog: Getting Access to the CrowdStrike API. CrowdStrike's full documentation is behind a log in, so you must log in to your account to access the full API documentation.

Customer.io credentials

You can use these credentials to authenticate the following nodes with Customer.io.

- Customer.io
- Customer.io Trigger

Prerequisites

Create a Customer.io account.

Supported authentication methods

- API Key

Related resources

Refer to Customer.io's summary API documentation for more information about the service.

For detailed API reference documentation for each API, refer to the Track API documentation and the App API documentation.

Using API key

To configure this credential, you'll need:

- A **Tracking API Key**: For use with the Track API at `https://track.customer.io/api/v1/`. See the FAQs below for more details.
- Your **Region**: Customer.io uses different API subdomains depending on the region you select. Options include:
  - **Global region**: Keeps the default URLs for both APIs; for use in all non-EU countries/regions.
  - **EU region**: Adjusts the Track API subdomain to `track-eu` and the App API subdomain to `api-eu`; only use this if you are in the EU.
- A **Tracking Site ID**: Required with your **Tracking API Key**
- An **App API Key**: For use with the App API at `https://api.customer.io/v1/api/`. See the FAQs below for more details.

Refer to the Customer.io Finding and managing your API credentials documentation for instructions on creating both Tracking API and App API keys.

Why you need a Tracking API Key and an App API Key

Customer.io has two different API endpoints and generates and stores the keys for each slightly differently:

- The Track API at `https://track.customer.io/api/v1/`
- The App API at `https://api.customer.io/v1/api/`

The Track API requires a Tracking Site ID; the App API doesn't.

Based on the operation you want to perform, n8n uses the correct API key and its corresponding endpoint.

Datadog credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Datadog account.

Related resources

Refer to Datadog's API documentation for more information about authenticating with the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API Key

To configure this credential, you'll need:

- Your Datadog instance **Host**
- An **API Key**
- An **App Key**

Refer to Authentication on Datadog's website for more information.

DeepL credentials

You can use these credentials to authenticate the following nodes:

- DeepL

Prerequisites

Create a DeepL developer account. n8n works with both Free and Pro API Plans.

Supported authentication methods

- API key

Related resources

Refer to DeepL's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to DeepL's Authentication documentation for more information on getting your API key.
- To identify which **API Plan** you're on. DeepL has different API endpoints for each plan, so be sure you select the correct one:
  - Pro Plan
  - Free Plan

DeepSeek credentials

You can use these credentials to authenticate the following nodes:

- Chat DeepSeek

Prerequisites

Create a DeepSeek account.

Supported authentication methods

- API key

Related resources

Refer to DeepSeek's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**

To generate your API Key:

1. Login to your DeepSeek account or create an account.
2. Open your API keys page.
3. Select **Create new secret key** to create an API key, optionally naming the key.
4. Copy your key and add it as the **API Key** in n8n.

Refer to the Your First API Call page for more information.

Demio credentials

You can use these credentials to authenticate the following nodes:

- Demio

Prerequisites

Create a Demio account.

Supported authentication methods

- API key

Related resources

Refer to Demio's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**
- An **API Secret**

You must have Owner status in Demio to generate API keys and secrets. To view and generate API keys and secrets, go to **Account Settings > API**. Refer to the Demio Account Owner Settings documentation for more detailed steps.

DFIR-IRIS credentials

–8<– “_snippets/integrations/builtin/credentials/cred-only-statement.md”

Prerequisites

An accessible instance of DFIR-IRIS.

Related resources

Refer to DFIR-IRIS’s API documentation for more information about authenticating with the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n’s website.

Using API Key

To configure this credential, you’ll need:

- An **API Key**: Refer to DFIR-IRIS’s API documentation for instructions on getting your API key.
- The **Base URL** of your DFIR-IRIS instance.

DHL credentials

You can use these credentials to authenticate the following nodes:

- DHL

Supported authentication methods

- API key

Related resources

Refer to DHL’s Developer documentation for more information about the service.

Using API key

To configure this credential, you'll need a DHL Developer account and:

- An **API Key**

To get an API key, create an app:

1. In the DHL Developer portal, select the user icon to open your User Apps.
2. Select **+ Create App**.
3. Enter an **App name**, like `n8n integration`.
4. Enter a **Machine name**, like `n8n_integration`.
5. In **SELECT APIs**, select **Shipment Tracking - Unified**. The API is added to the **Add API to app** section.
6. In the **Add API to app** section, select the **+** next to the **Shipment Tracking - Unified** API.
7. Select **Create App**. The **Apps** page opens, displaying the app you just created.
8. Select the app you just created to view its details.
9. Select **Show key** next to **API Key**.
10. Copy the **API Key** and enter it in your n8n credential.

Refer to How to create an app? for more information.

Discord credentials

You can use these credentials to authenticate the following nodes:

- Discord

Prerequisites

- Create a Discord account.
- For Bot and OAuth2 credentials:
  - Set up your local developer environment.
  - Create an application and a bot user.
- For webhook credentials, create a webhook.

Supported authentication methods

- Bot
- OAuth2
- Webhook

Not sure which method to use? Refer to Choose an authentication method for more guidance.

Related resources

Refer to Discord's Developer documentation for more information about the service.

Using bot

Use this method if you want to add the bot to your Discord server using a bot token rather than OAuth2.

To configure this credential, you'll need:

- A **Bot Token**: Generated once you create an application with a bot.

To create an application with a bot and generate the **Bot Token**:

1. If you don't have one already, create an app in the developer portal.
2. Enter a **Name** for your app.
3. Select **Create**.
4. Select **Bot** from the left menu.
5. Under **Token**, select **Reset Token** to generate a new bot token.
6. Copy the token and add it to your n8n credential.
7. In **Bot > Privileged Gateway Intents**, add any privileged intents you want your bot to have. Refer to Configuring your bot for more information on privileged intents.
    - n8n recommends activating **SERVER MEMBERS INTENT: Required for your bot to receive events listed under GUILD_MEMBERS**.
8. In **Installation > Installation Contexts**, select the installation contexts you want your bot to use:
    - Select **Guild Install** for server-installed apps. (Most common for n8n users.)
    - Select **User Install** for user-installed apps. (Less common for n8n users, but may be useful for testing.)
    - Refer to Discord's Choosing installation contexts documentation for more information about these installation contexts.
9. In **Installation > Install Link**, select **Discord Provided Link** if it's not already selected.
10. Still on the **Installation** page, in the **Default Install Settings** section, select `applications.commands` and `bot` scopes. Refer to Discord's Scopes documentation for more information about these and other scopes.
11. Add permissions on the **Bot > Bot Permissions** page. Refer to Discord's Permissions documentation for more information. n8n recommends selecting these permissions for the Discord node:
    - Manage Roles
    - Manage Channels
    - Read Messages/View Channels
    - Send Messages
    - Create Public Threads
    - Create Private Threads
    - Send Messages in Threads
    - Send TTS Messages
    - Manage Messages
    - Manage Threads
    - Embed Links
    - Attach Files
    - Read Message History
    - Add Reactions
12. Add the app to your server or test server:
    1. Go to **Installation > Install Link** and copy the link listed there.

2. Paste the link in your browser and hit Enter.
3. Select **Add to server** in the installation prompt.
4. Once your app's added to your server, you'll see it in the member list.

These steps outline the basic functionality needed to set up your n8n credential. Refer to the Discord Creating an App guide for more information on creating an app, especially:

- Fetching your credentials for getting your app's credentials into your local developer environment.
- Handling interactivity for information on setting up public endpoints for interactive `/slash` commands.

Using OAuth2

Use this method if you want to add the bot to Discord servers using the OAuth2 flow, which simplifies the process for those installing your app.

To configure this credential, you'll need:

- A **Client ID**
- A **Client Secret**
- Choose whether to send **Authentication** in the **Header** or **Body**
- A **Bot Token**

For details on creating an application with a bot and generating the token, follow the same steps as in Using bot above.

Then:

1. Copy the **Bot Token** you generate and add it into the n8n credential.
2. Open the **OAuth2** page in your Discord application to access your **Client ID** and generate a **Client Secret**. Add these to your n8n credential.
3. From n8n, copy the **OAuth Redirect URL** and add it into the Discord application in **OAuth2 > Redirects**. Be sure you save these changes.

Using webhook

To configure this credential, you'll need:

- A **Webhook URL**: Generated once you create a webhook.

To get a Webhook URL, you create a webhook and copy the URL that gets generated:

1. Open your Discord **Server Settings** and open the **Integrations** tab.
2. Select **Create Webhook** to create a new webhook.
3. Give your webhook a **Name** that makes sense.
4. Select the **avatar** next to the **Name** to edit or upload a new avatar.
5. In the **CHANNEL** dropdown, select the channel the webhook should post to.
6. Select **Copy Webhook URL** to copy the Webhook URL. Enter this

URL in your n8n credential.

Refer to the Discord Making a Webhook documentation for more information.

Choose an authentication method

The simplest installation is a **webhook**. You create and add webhooks to a single channel on a Discord server. Webhooks can post messages to a channel. They don't require a bot user or authentication. But they can't listen or respond to user requests or commands. If you need a straightforward way to send messages to a channel without the need for interaction or feedback, use a webhook.

A **bot** is an interactive step up from a webhook. You add bots to the Discord server (referred to as a `guild` in the Discord API documentation) or to user accounts. Bots added to the server can interact with users on all the server's channels. They can manage channels, send and retrieve messages, retrieve the list of all users, and change their roles. If you need to build an interactive, complex, or multi-step workflow, use a bot.

OAuth2 is basically a **bot** that uses an OAuth2 flow rather than just the bot token. As with bots, you add these to the Discord server or to user accounts. These credentials offer the same functionalities as bots, but they can simplify the installation of the bot on your server.

Discourse credentials

You can use these credentials to authenticate the following nodes:

- Discourse

Prerequisites

- Host an instance of Discourse
- Create an account on your hosted instance and make sure that you are an admin

Supported authentication methods

- API key

Related resources

Refer to Discourse's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- The **URL** of your Discourse instance, for example `https://community.n8n.io`
- An **API Key**: Create an API key through the Discourse admin panel. Refer to the Discourse create and configure an API key documentation for instructions on creating an API key and specifying a username.
- A **Username**: Use your own name, `system`, or another user.

Refer to the Authentication section of the Discourse API documentation for examples.

Disqus credentials

You can use these credentials to authenticate the following nodes:

- Disqus

Prerequisites

- Create a Disqus account.
- Register an API application.

Supported authentication methods

- API access token

Related resources

Refer to Disqus's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- An **Access Token**: Once you've registered an API application, copy the **API Key** and add it to n8n as the **Access Token**.

Drift credentials

You can use these credentials to authenticate the following nodes:

- Drift

Prerequisites

- Create a Drift account.
- Create a Drift app.

Supported authentication methods

- API personal access token
- OAuth2

Related resources

Refer to Drift's API documentation for more information about the service.

Using API personal access token

To configure this credential, you'll need:

- A **Personal Access Token**: To get a token, create a Drift app. Install the app to generate an OAuth Access token. Add this to the n8n credential as your **Personal Access Token**.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you need to configure OAuth2 from scratch or need more detail on what's happening in the OAuth web flow, refer to the instructions in the Drift Authentication and Scopes documentation to set up OAuth for your app.

Dropbox credentials

You can use these credentials to authenticate the following nodes:

- Dropbox

Supported authentication methods

- API access token: Dropbox recommends this method for testing with your user account and granting a limited number of users access.
- OAuth2: Dropbox recommends this method for production or for testing with more than 50 users.

Related resources

Refer to Dropbox's Developer documentation for more information about the service.

Using access token

To configure this credential, you'll need a Dropbox developer account and:

- An **Access Token**: Generated once you create a Dropbox app.
- An **App Access Type**

To set up the credential, create a Dropbox app:

1. Open the App Console within the Dropbox developer portal.
2. Select **Create app**.
3. In **Choose an API**, select **Scoped access**.
4. In **Choose the type of access you need**, choose whichever option best fits your use of the Dropbox node:
   - **App Folder** grants access to a single folder created specifically for your app.
   - **Full Dropbox** grants access to all files and folders in your user's Dropbox.
   - Refer to the DBX Platform developer guide for more information.
5. In **Name your app**, enter a name for your app, like `n8n integration`.
6. Check the box to agree to the **Dropbox API Terms and Conditions**.
7. Select **Create app**. The app's **Settings** open.
8. In the **OAuth 2** section, in **Generated access token**, select **Generate**.
9. Copy the access token and enter it as the **Access Token** in your n8n credential.
10. In n8n, select the same **App Access Type** you selected for your app.

Refer to the Dropbox App Console Settings documentation for more information.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

Cloud users need to select the **App Access Type**:

- **App Folder** grants access to a single folder created specifically for your app.
- **Full Dropbox** grants access to all files and folders in your user's Dropbox.
- Refer to the DBX Platform developer guide for more information.

If you're self-hosting n8n, you'll need to configure OAuth2 manually:

1. Open the App Console within the Dropbox developer portal.
2. Select **Create app**.
3. In **Choose an API**, select **Scoped access**.
4. In **Choose the type of access you need**, choose whichever option best fits your use of the Dropbox node:
   - **App Folder** grants access to a single folder created specifically for your app.
   - **Full Dropbox** grants access to all files and folders in your user's Dropbox.
   - Refer to the DBX Platform developer guide for more

information.
5. In **Name your app**, enter a name for your app, like `n8n integration`.
6. Check the box to agree to the **Dropbox API Terms and Conditions**.
7. Select **Create app**. The app's **Settings** open.
8. Copy the **App key** and enter it as the **Client ID** in your n8n credential.
9. Copy the **Secret** and enter it as the **Client Secret** in your n8n credential.
10. In n8n, copy the **OAuth Redirect URL** and enter it in the Dropbox **Redirect URIs**.
11. In n8n, select the same **App Access Type** you selected for your app.

Refer to the instructions in the Dropbox Implementing OAuth documentation for more information.

For internal tools and limited usage, you can keep your app private. But if you'd like your app to be used by more than 50 users or you want to distribute it, you'll need to complete Dropbox's production approval process. Refer to **Production Approval** in the DBX Platform developer guide for more information.

Dropcontact credentials

You can use these credentials to authenticate the following nodes:

- Dropcontact

Prerequisites

Create a developer account in Dropcontact.

Supported authentication methods

- API key

Related resources

Refer to Dropcontact's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: To view your API key in Dropcontact, go to **API**. Refer to the Dropcontact API key documentation for more information.

Dynatrace credentials

–8<– “_snippets/integrations/builtin/credentials/cred-only-statement.md”

Prerequisites

Create a Dynatrace account.

Related resources

Refer to Dynatrace’s API documentation for more information about authenticating with the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n’s website.

Using Access Token

To configure this credential, you’ll need:

- An **Access Token**

Refer to Access Tokens on Dynatrace’s website for more information.

E-goi credentials

You can use these credentials to authenticate the following nodes:

- E-goi

Prerequisites

Create an E-goi account.

Supported authentication methods

- API key

Related resources

Refer to E-goi’s API documentation for more information about the service.

Using API key

To configure this credential, you’ll need:

- An **API Key**: Refer to E-goi's API key documentation for instructions on generating and viewing an API key.

Elasticsearch credentials

You can use these credentials to authenticate the following nodes:

- Elasticsearch

Supported authentication methods

- Basic auth

Related resources

Refer to Elasticsearch's documentation for more information about the service.

Using basic auth

To configure this credential, you'll need an Elasticsearch account with a deployment and:

- A **Username**
- A **Password**
- Your Elasticsearch application's **Base URL** (also known as the Elasticsearch application endpoint)

To set up the credential:

1. Enter your Elasticsearch **Username**.
2. Enter your Elasticsearch **Password**.
3. In Elasticsearch, go to **Deployments**.
4. Select your deployment.
5. Select **Manage this deployment**.
6. In the **Applications** section, copy the endpoint of the **Elasticsearch** application.
7. Enter this in n8n as the **Base URL**.
8. By default, n8n connects only if SSL certificate validation succeeds. If you'd like to connect even if SSL certificate validation fails, turn on **Ignore SSL Issues**.

Elastic Security credentials

You can use these credentials to authenticate the following nodes:

- Elastic Security

Prerequisites

- Create an Elastic Security account.
- Deploy an application.

Supported authentication methods

- Basic auth
- API Key

Related resources

Refer to Elastic Security's documentation for more information about the service.

Using basic auth

To configure this credential, you'll need:

- A **Username**: For the user account you log into Elasticsearch with.
- A **Password**: For the user account you log into Elasticsearch with.
- Your Elasticsearch application's **Base URL** (also known as the Elasticsearch application endpoint):
    1. In Elasticsearch, select the option to **Manage this deployment**.
    2. In the **Applications** section, copy the endpoint of the **Elasticsearch** application.
    3. Add this in n8n as the **Base URL**.

Using API key

To configure this credential, you'll need:

- An **API Key**: For the user account you log into Elasticsearch with. Refer to Elasticsearch's Create API key documentation for more information.
- Your Elasticsearch application's **Base URL** (also known as the Elasticsearch application endpoint):
    1. In Elasticsearch, select the option to **Manage this deployment**.
    2. In the **Applications** section, copy the endpoint of the **Elasticsearch** application.
    3. Add this in n8n as the **Base URL**.

Emelia credentials

You can use these credentials to authenticate the following nodes:

- Emelia
- Emelia Trigger

Prerequisites

Create an Emelia account.

Supported authentication methods

- API key

Related resources

Refer to Emelia's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: To generate an API Key in Emelia, access your **API Keys** by selecting the avatar in the top right (your **Settings**). Refer to the Authentication section of Emelia's API documentation for more information.

ERPNext credentials

You can use these credentials to authenticate the following nodes:

- ERPNext

Prerequisites

- Create an ERPNext account.

Supported authentication methods

- API key

Related resources

Refer to ERPNext's documentation for more information about the service.

Refer to ERPNext's developer documentation for more information about working with the framework.

Using API key

To configure this credential, you'll need:

- An **API Key**: Generate this from your own ERPNext user account in **Settings > My Settings > API Access**.
- An **API Secret**: Generated with the API key.
- Your ERPNext **Environment**:
  - For **Cloud-hosted**:
    - Your ERPNext **Subdomain**: Refer to the FAQs
    - Your **Domain**: Choose between `erpnext.com` and `frappe.cloud`.
  - For **Self-hosted**:
    - The fully qualified **Domain** where you host ERPNext
- Choose whether to **Ignore SSL Issues**: When selected, n8n will connect even if SSL certificate validation is unavailable.

If you are an ERPNext System Manager, you can also generate API keys and secrets for other users. Refer to the ERPNext Adding Users documentation for more information.

How to find the subdomain of an ERPNext cloud-hosted account

You can find your ERPNext subdomain by reviewing the address bar of your browser. The string between `https://` and either `.erpnext.com` or `frappe.cloud` is your subdomain.

For example, if the URL in the address bar is `https://n8n.erpnext.com`, the subdomain is `n8n`.

Eventbrite credentials

You can use these credentials to authenticate the following nodes:

- Eventbrite Trigger

Prerequisites

Create an Eventbrite account.

Supported authentication methods

- API private key
- OAuth2

Related resources

Refer to Eventbrite's API documentation for more information about the service.

Using API private key

To configure this credential, you'll need:

- A **Private Key**: Refer to the Eventbrite API Authentication Get a Private Token documentation for detailed steps to generate a Private Token. Use this private token as the **Private Key** in the n8n credential.

Using OAuth2

–8<– “_snippets/integrations/builtin/credentials/cloud-oauth-button.md”

If you need to configure OAuth2 from scratch or need more detail on what’s happening in the OAuth web flow, refer to the instructions in the Eventbrite API authentication For App Partners documentation to set up OAuth.

F5 Big-IP credentials

–8<– “_snippets/integrations/builtin/credentials/cred-only-statement.md”

Prerequisites

Create an F5 Big-IP account.

Authentication methods

- Account login

Related resources

Refer to F5 Big-IP’s API documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n’s website.

Using account login

To configure this credential, you’ll need:

- A **Username**: Use the username you use to log in to F5 Big-IP.
- A **Password**: Use the user password you use to log in to F5 Big-IP.

Facebook App credentials

You can use these credentials to authenticate the following nodes:

- Facebook Trigger

Supported authentication methods

- App access token

Related resources

Refer to Meta's Graph API documentation for more information about the service.

Using app access token

To configure this credential, you'll need a Meta for Developers account and:

- An app **Access Token**
- An optional **App Secret**: Used to verify the integrity and origin of the payload.

There are five steps in setting up your credential:

1. Create a Meta app with the Webhooks product.
2. Generate an App Access Token for that app.
3. Configure the Facebook trigger.
4. Optional: Add an app secret.
5. App Review: Only required if your app's users don't have roles on the app itself. If you're creating the app for your own internal purposes, this isn't necessary.

Refer to the detailed instructions below for each step.

Create a Meta app

To create a Meta app:

1. Go to the Meta Developer App Dashboard and select **Create App**.
2. If you have a business portfolio and you're ready to connect the app to it, select the business portfolio. If you don't have a business portfolio or you're not ready to connect the app to the portfolio, select **I don't want to connect a business portfolio yet** and select **Next**. The **Use cases** page opens.
3. Select **Other**, then select **Next**.
4. Select **Business** and **Next**.
5. Complete the essential information:
    - Add an **App name**.
    - Add an **App contact email**.
    - Here again you can connect to a business portfolio or skip it.
6. Select **Create app**.
7. The **Add products to your app** page opens.
8. Select **App settings > Basic** from the left menu.
9. Enter a **Privacy Policy URL**. (Required to take the app "Live.")
10. Select **Save changes**.
11. At the top of the page, toggle the **App Mode** from **Development** to **Live**.
12. In the left menu, select **Add Product**.
13. The **Add products to your app** page appears. Select **Webhooks**.
14. The **Webhooks** product opens.

Refer to Meta's Create an app documentation for more information on creating an app, required fields like the Privacy Policy URL, and adding products.

For more information on the app modes and switching to **Live** mode, refer to App Modes and Publish | App Types.

Generate an App Access Token

Next, create an app access token to be used by your n8n credential and the Webhooks product:

1. In a separate tab or window, open the Graph API explorer.
2. Select the **Meta App** you just created in the **Access Token** section.
3. In **User or Page**, select **Get App Token**.
4. Select **Generate Access Token**.
5. The page prompts you to log in and grant access. Follow the on-screen prompts.
6. Copy the token and enter it in your n8n credential as the **Access Token**. Save this token somewhere else, too, since you'll need it for the Webhooks configuration.
7. Save your n8n credential.

Refer to the Meta instructions for Your First Request for more information on generating the token.

Configure the Facebook Trigger

Now that you have a token, you can configure the Facebook Trigger node:

1. In your Meta app, copy the **App ID** from the top navigation bar.
2. In n8n, open your Facebook Trigger node.
3. Paste the **App ID** into the **APP ID** field.
4. Select **Execute step** to shift the trigger into listening mode.
5. Return to the tab or window where your Meta app's **Webhooks** product configuration is open.
6. **Subscribe** to the objects you want to receive Facebook Trigger notifications about. For each subscription:
    1. Copy the **Webhook URL** from n8n and enter it as the **Callback URL** in your Meta App.
    2. Enter the **Access Token** you copied above as the **Verify token**.
    3. Select **Verify and save**. (This step fails if you don't have your n8n trigger listening.)
    4. Some webhook subscriptions, like **User**, prompt you to subscribe to individual events. Subscribe to the events you're interested in.
    5. You can send some **Test** events from Meta to confirm things are working. If you send a test event, verify its receipt in n8n.

Refer to the Facebook Trigger node documentation for more information.

Optional: Add an App Secret

For added security, Meta recommends adding an **App Secret**. This signs all API calls with the `appsecret_proof` parameter. The app secret proof is a sha256 hash of your access token, using your app secret as the key.

To generate an App Secret:

1. In Meta while viewing your app, select **App settings > Basic** from the left menu.
2. Select **Show** next to the **App secret** field.
3. The page prompts you to re-enter your Facebook account credentials. Once you do so, Meta shows the App Secret.
4. Highlight it to select it, copy it, and paste this into your n8n credential as the **App Secret**.
5. **Save** your n8n credential.

Refer to the App Secret documentation for more information.

App review

App Review requires Business Verification.

Your app must go through App Review if it will be used by someone who:

- Doesn't have a role on the app itself.
- Doesn't have a role in the Business that has claimed the app.

If your only app users are users who have a role on the app itself, App Review isn't required.

As part of the App Review process, you may need to request advanced access for your webhook subscriptions.

Refer to Meta's App Review and Advanced Access documentation for more information.

Common issues

Unverified apps limit

Facebook only lets you have a developer or administrator role on a maximum of 15 apps that aren't already linked to a Meta Verified Business Account.

Refer to Limitations | Create an app if you're over that limit.

Facebook Graph API credentials

You can use these credentials to authenticate the following nodes:

- Facebook Graph API

Supported authentication methods

- App access token

Related resources

Refer to Meta's Graph API documentation for more information about the service.

Using app access token

To configure this credential, you'll need a Meta for Developers account and:

- An app **Access Token**

There are two steps in setting up your credential:

1. Create a Meta app with the products you need to access.
2. Generate an App Access Token for that app.

Refer to the detailed instructions below for each step.

Create a Meta app

To create a Meta app:

1. Go to the Meta Developer App Dashboard and select **Create App**.
2. If you have a business portfolio and you're ready to connect the app to it, select the business portfolio. If you don't have a business portfolio or you're not ready to connect the app to the portfolio, select **I don't want to connect a business portfolio yet** and select **Next**. The **Use cases** page opens.
3. Select the **Use case** that aligns with how you wish to use the Facebook Graph API. For example, for products in Meta's **Business** suite (like Messenger, Instagram, WhatsApp, Marketing API, App Events, Audience Network, Commerce API, Fundraisers, Jobs, Threat Exchange, and Webhooks), select **Other**, then select **Next**.
4. Select **Business** and **Next**.
5. Complete the essential information:
    - Add an **App name**.
    - Add an **App contact email**.
    - Here again you can connect to a business portfolio or skip it.
6. Select **Create app**.
7. The **Add products to your app** page opens.
8. Select **App settings > Basic** from the left menu.
9. Enter a **Privacy Policy URL**. (Required to take the app "Live.")
10. Select **Save changes**.
11. At the top of the page, toggle the **App Mode** from **Development** to **Live**.
12. In the left menu, select **Add Product**.
13. The **Add products to your app** page appears. Select the products that make sense for your app and configure them.

Refer to Meta's Create an app documentation for more information on creating an app, required fields like the Privacy Policy URL, and adding products.

For more information on the app modes and switching to **Live** mode, refer to App Modes and Publish | App Types.

Generate an App Access Token

Next, create an app access token to use with your n8n credential and the products you selected:

1. In a separate tab or window, open the Graph API explorer.
2. Select the **Meta App** you just created in the **Access Token** section.
3. In **User or Page**, select **Get App Token**.
4. Select **Generate Access Token**.
5. The page prompts you to log in and grant access. Follow the on-screen prompts.
6. Copy the token and enter it in your n8n credential as the **Access Token**.

Refer to the Meta instructions for Your First Request for more information on generating the token.

Facebook Lead Ads credentials

You can use these credentials to authenticate the following nodes:

- Facebook Lead Ads trigger

Supported authentication methods

- OAuth2

Related resources

Refer to Facebook Lead Ads' documentation for more information about the service.

View example workflows and related content on n8n's website.

Using OAuth2

To configure this credential, you'll need a Meta for Developers account and:

- A **Client ID**
- A **Client Secret**

To get both, create a Meta app with either the Facebook Login product or the Facebook Login for Business product.

To create your app and set up the credential with **Facebook Login for Business**:

1. Go to the Meta Developer App Dashboard and select **Create App**.
2. If you have a business portfolio and you're ready to connect the app to it, select the business portfolio. If you don't have a business portfolio or you're not ready to connect the app to the portfolio, select **I don't want to connect a business portfolio yet** and select **Next**. The **Use cases** page opens.
3. Select **Other**, then select **Next**.
4. Select **Business** and **Next**.
5. Complete the essential information:
    - Add an **App name**.
    - Add an **App contact email**.
    - Here again you can connect to a business portfolio or skip it.
6. Select **Create app**. The **Add products to your app** page opens.
7. Select **Facebook Login for Business**. The **Settings** page for this product opens.
8. Copy the **OAuth Redirect URL** from your n8n credential.
9. In your Meta app settings in **Client OAuth settings**, paste that URL as the **Valid OAuth Redirect URIs**.
10. Select **App settings > Basic** from the left menu.
11. Copy the **App ID** and enter it as the **Client ID** within your n8n credential.
12. Copy the **App Secret** and enter it as the **Client Secret** within your n8n credential.

Your credential should successfully connect now, but you'll need to go through the steps to take your Meta app live before you can use it with the Facebook Lead Ads trigger. Here's a summary of what you'll need to do:

1. In your Meta app, select **App settings > Basic** from the left menu.
2. Enter a **Privacy Policy URL**. (Required to take the app "Live.")
3. Select **Save changes**.
4. At the top of the page, toggle the **App Mode** from **Development** to **Live**.
5. Facebook Login for Business requires Advanced Access for `public_profile`. To add it, go to **App Review > Permissions and Features**.
6. Search for `public_profile` and select **Request advanced access**.
7. Complete the steps for business verification.
8. Use the Lead Ads Testing Tool to trigger some demo form submissions and test your workflow.

Refer to Meta's Create an app documentation for more information on creating an app, required fields like the Privacy Policy URL, and adding products.

For more information on the app modes and switching to **Live** mode, refer to App Modes and Publish | App Types.

Figma credentials

You can use these credentials to authenticate the following nodes:

- Figma Trigger (Beta)

Prerequisites

Create a Figma account. You need an admin or owner level account.

Supported authentication methods

- API key

Related resources

Refer to Figma's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- A Personal **Access Token** (PAT): Refer to the Figma API Access Tokens documentation for instructions on generating a Personal **Access Token**.

FileMaker credentials

You can use these credentials to authenticate the following nodes:

- FileMaker

Prerequisites

- Create a user account on a FileMaker Server with the `fmrest` extended privilege to Access the FileMaker Data API.
- Ensure the FileMaker Server can use the FileMaker Data API:
  1. Prepare your database for FileMaker Data API access using FileMaker Pro. You can create a database or prepare an existing database.
     - Refer to Prepare databases for FileMaker Data API access for more information.
  2. Write code that calls FileMaker Data API methods to find, create, edit, duplicate, and delete records in a hosted database.
     - Refer to Write FileMaker Data API calls for more information.
  3. Host your solution with FileMaker Data API access enabled.
     - Refer to Host a FileMaker Data API solution for more information.
  4. Test that FileMaker Data API access is working.
     - Refer to Test the FileMaker Data API solution for more information.

5. Monitor your hosted solution using Admin Console.
    - Refer to Monitor FileMaker Data API solutions for more information.

Supported authentication methods

- Database connection

Related resources

Refer to FileMaker's Data API Guide for more information about the service.

Using database connection

To configure this credential:

1. Enter the **Host** name or IP address of your FileMaker Server.
2. Enter the **Database** name. This should match the database name as it appears in the **Databases** list within FileMaker.
3. Enter the user account **Login** for the account with the `fmrest` extended privilege. Refer to the previous Prerequisites section for more information.
4. Enter the **Password** for that user account.

Filescan credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Filescan account.

Related resources

Refer to Filescan's API documentation for more information about authenticating with the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API key

To configure this credential, you'll need:

- An **API Key**: Generate your API key from your **profile settings > API Key**. Refer to the Filescan FAQ for more information.

Flow credentials

You can use these credentials to authenticate the following nodes:

- Flow
- Flow Trigger

Prerequisites

Create a Flow account.

Supported authentication methods

- API key

Related resources

Refer to Flow's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- Your numeric **Organization ID**
- An **Access Token**

Refer to the Flow API Getting Started documentation for instructions on generating your Access Token and viewing your Organization ID.

Form.io Trigger credentials

You can use these credentials to authenticate the following nodes:

- Form.io Trigger

Supported authentication methods

- Basic auth

Related resources

Refer to Form.io's API documentation for more information about the service.

Using basic auth

To configure this credential, you'll need a Form.io account and:

- Your **Environment**
- Your login **Email address**
- Your **Password**

To set up the credential:

1. Select your **Environment**:
    - Choose **Cloud hosted** if you aren't hosting Form.io yourself.
    - Choose **Self-hosted** if you're hosting Form.io yourself. Then add:
        - Your **Self-Hosted Domain**. Use only the domain itself. For example, if you view a form at `https://yourserver.com/yourproject/manage/view`, the Self-Hosted Domain is `https://yourserver.com`.
2. Enter the **Email address** you use to log in to Form.io.
3. Enter the **Password** you use to log in to Form.io.

Formstack Trigger credentials

You can use these credentials to authenticate the following nodes:

- Formstack Trigger

Prerequisites

Create a Formstack account.

Supported authentication methods

- API access token
- OAuth2

Related resources

Refer to Formstack's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- An API **Access Token**: To generate an Access Token, create a new application in Formstack using the following details:
    - **Redirect URI**: For cloud n8n instances, enter `https://oauth.n8n.cloud/oauth2/callback`.
        - For self-hosted n8n instances, enter the OAuth callback URL for your n8n instance in the format `https://<n8n_url>/rest/oauth2-credential/callback`. For example `https://localhost:5678/rest/oauth2-credential/callback`.
    - **Platform**: Select **Website**.

Once you've created the application, copy the access token either from the applications list or by selecting the application to view its details.

Refer to Formstack's API Authorization documentation for more detailed instructions.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**
- A **Client Secret**

To generate both of these, create a new application in Formstack using the following details:

- **Redirect URI**: Copy the **OAuth Redirect URL** from the n8n credential to enter here.
  - For self-hosted n8n instances, enter the OAuth callback URL for your n8n instance in the format `https://<n8n_url>/rest/oauth2-credential/callback`. For example `https://localhost:5678/rest/oauth2-credential/callback`.
- **Platform**: Select **Website**.

Once you've created the application, select it from the applications list to view the **Application Details**. Copy the **Client ID** and **Client Secret** and add them to n8n. Once you've added both, select the **Connect my account** button to begin the OAuth2 flow and authorization process.

Refer to Formstack's API Authorization documentation for more detailed instructions.

Fortinet FortiGate credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Fortinet FortiGate account.

Supported authentication methods

- API access token

Related resources

Refer to Fortinet FortiGate's API documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API access token

To configure this credential, you'll need:

- An API **Access Token**: To generate an access token, create a REST API administrator.

Refer to the Fortinet FortiGate Using APIs documentation for more information about token-based authentication in FortiGate.

Freshdesk credentials

You can use these credentials to authenticate the following nodes:

- Freshdesk

Prerequisites

Create a Freshdesk account.

Supported authentication methods

- API key

Related resources

Refer to Freshdesk's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to the Freshdesk API authenticaton documentation for detailed instructions on getting your API key.
- A Freshdesk **Domain**: Use the subdomain of your Freshdesk account. This is part of the URL, for example `https://<subdomain>.freshdesk.com`. So if you access Freshdesk through `https://n8n.freshdesk.com`, enter `n8n` as your **Domain**.

Freshservice credentials

You can use these credentials to authenticate the following nodes:

- Freshservice

Prerequisites

Create a Freshservice account.

Supported authentication methods

- API key

Related resources

Refer to Freshservice's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to the Freshservice API authenticaton documentation for detailed instructions on getting your API key.
- Your Freshservice **Domain**: Use the subdomain of your Freshservice account. This is part of the URL, for example `https://<subdomain>.freshservice.com`. So if you access Freshservice through `https://n8n.freshservice.com`, enter `n8n` as your **Domain**.

Freshworks CRM credentials

You can use these credentials to authenticate the following nodes:

- Freshworks CRM

Prerequisites

Create a Freshworks CRM account.

Supported authentication methods

- API key

Related resources

Refer to Freshworks CRM's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to the Freshworks CRM API authenticaton documentation for detailed instructions on getting your API key.
- Your Freshworks CRM **Domain**: Use the subdomain of your Freshworks CRM account. This is part of the URL, for example `https://<subdomain>.myfreshworks.com`. So if you access Freshworks CRM through `https://n8n.myfreshworks.com`, enter `n8n` as your **Domain**.

FTP credentials

You can use these credentials to authenticate the following nodes:

- FTP

Prerequisites

Create an account on a File Transfer Protocol (FTP) server like JSCAPE, OpenSSH, or FileZilla Server.

Supported authentication methods

- **FTP account**: Use this method if your FTP server doesn't support SSH tunneling or encrypted connections.
- **SFTP account**: Use this method if your FTP server supports SSH tunneling and encrypted connections.

Related resources

File Transfer Protocol (FTP) and Secure Shell File Transfer Protocol (SFTP) are protocols for transferring files directly between an FTP/SFTP client and server.

Using FTP account

Use this method if your FTP server doesn't support SSH tunneling or encrypted connections.

To configure this credential, you'll need to:

1. Enter the name or IP address of your FTP server's **Host**.
2. Enter the **Port** number the connection should use.
3. Enter the **Username** the credential should connect as.
4. Enter the user's **Password**.

Review your FTP server provider's documentation for instructions on getting the information you need.

Using SFTP account

Use this method if your FTP server supports SSH tunneling and encrypted connections.

To configure this credential, you'll need to:

1. Enter the name or IP address of your FTP server's **Host**.
2. Enter the **Port** number the connection should use.
3. Enter the **Username** the credential should connect as.
4. Enter the user's **Password**.
5. For the **Private Key**, enter a string for either key-based or host-based user authentication
   - Enter your Private Key in OpenSSH format. This is most often generated using the ssh-keygen `-o` parameter, for example: `ssh-keygen -o -a 100 -t ed25519`.
6. If the **Private Key** is encrypted, enter the **Passphrase** used to decrypt it.
   - If the **Private Key** doesn't use a passphrase, leave this field blank.

Review your FTP server provider's documentation for instructions on getting the information you need.

GetResponse credentials

You can use these credentials to authenticate the following nodes:

- GetResponse
- GetResponse Trigger

Prerequisites

Create a GetResponse account.

Supported authentication methods

- API key
- OAuth2

Related resources

Refer to GetResponse's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: To view or generate an API key, go to **Integrations and API > API**. Refer to the GetResponse Help Center for more detailed instructions.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated when you register your application.
- A **Client Secret**: Generated when you register your application as the **Client Secret Key**.

When you register your application, copy the **OAuth Redirect URL** from n8n and add it as the **Redirect URL** in GetResponse.

Configure OAuth2 credentials for a local environment

GetResponse doesn't accept the localhost callback URL. Follow the steps below to configure the OAuth credentials for a local environment: 1. Use ngrok to expose the local server running on port `5678` to the internet. In your terminal, run the following command:

```
ngrok http 5678
```

2. Run the following command in a new terminal. Replace `<YOUR-NGROK-URL>` with the URL that you got from the previous step.

```
export WEBHOOK_URL=<YOUR-NGROK-URL>
```

3. Follow the Using OAuth2 instructions to configure your credentials, using this URL as your **Redirect URL**.

Ghost credentials

You can use these credentials to authenticate the following nodes:

- Ghost

Prerequisites

Create a Ghost account.

Supported authentication methods

- Admin API key
- Content API key

The keys are generated following the same steps, but the authorization flows and key format are different, so n8n stores the credentials separately. The Content API uses an API key; the Admin API uses an API key to generate a token for authentication.

Related resources

Refer to Ghost's Admin API documentation for more information about the Admin API service. Refer to Ghost's Content API documentation for more information about the Content API service.

Using Admin API key

To configure this credential, you'll need:

- The **URL** of your Ghost admin domain. Your admin domain can be different to your main domain and may include a subdirectory. All Ghost(Pro) blogs have a `*.ghost.io` domain as their admin domain and require https.
- An **API Key**: To generate a new API key, create a new Custom Integration. Refer to the Ghost Admin API Token Authentication Key documentation for more detailed instructions. Copy the **Admin API Key** and use this as the **API Key** in the Ghost Admin n8n credential.

Using Content API key

To configure this credential, you'll need:

- The **URL** of your Ghost admin domain. Your admin domain can be different to your main domain and may include a subdirectory. All Ghost(Pro) blogs have a `*.ghost.io` domain as their admin domain and require https.
- An **API Key**: To generate a new API key, create a new Custom Integration. Refer to the Ghost Content API Key documentation for more detailed instructions. Copy the **Content API Key** and use this as the **API Key** in the Ghost Content n8n credential.

Git credentials

You can use these credentials to authenticate the following nodes:

- Git

Prerequisites

Create an account on GitHub, GitLab, or similar platforms for use with Git.

Supported authentication methods

- Basic auth

Related resources

Refer to Git's documentation for more information about the service.

Using basic auth

To configure this credential, you'll need:

- A **Username** for GitHub, GitLab, or a similar platform
- A **Password** for GitHub, GitLab, or a similar platform

GitHub credentials

You can use these credentials to authenticate the following nodes:

- GitHub
- GitHub Trigger
- GitHub Document Loader: this node doesn't support OAuth.

Prerequisites

Create a GitHub account.

Supported authentication methods

- API access token: Use this method with any GitHub nodes.
- OAuth2: Use this method with GitHub and GitHub Trigger nodes only; don't use with GitHub Document Loader.

Related resources

Refer to GitHub's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need a GitHub account.

There are two steps to setting up this credential:

1. Generate a GitHub personal access token.
2. Set up the credential.

Refer to the sections below for detailed instructions.

Generate personal access token

To generate your personal access token:

1. If you haven't done so already, verify your email address with GitHub. Refer to Verifying your email address for more information.
2. Open your GitHub profile Settings.
3. In the left navigation, select **Developer settings**.
4. In the left navigation, under **Personal access tokens**, select **Tokens (classic)**.
5. Select **Generate new token > Generate new token (classic)**.
6. Enter a descriptive name for your token in the **Note** field, like `n8n integration`.
7. Select the **Expiration** you'd like for the token, or select **No**

expiration.
8. Select **Scopes** for your token. For most of the n8n GitHub nodes, add the `repo` scope.
    - A token without assigned scopes can only access public information.
    - Refer to
9. Select **Generate token**.
10. Copy the token.

Refer to Creating a personal access token (classic) for more information. Refer to Scopes for OAuth apps for more information on GitHub scopes.

Set up the credential

Then, in your n8n credential:

1. If you aren't using GitHub Enterprise Server, don't change the **GitHub server** URL.
    - If you're using GitHub Enterprise Server, update **GitHub server** to match the URL for your server.
2. Enter your **User** name as it appears in your GitHub profile.
3. Enter the **Access Token** you generated above.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you're self-hosting n8n, create a new GitHub OAuth app:

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

Refer to the GitHub Authorizing OAuth apps documentation for more information on the authorization process.

GitLab credentials

You can use these credentials to authenticate the following nodes:

- GitLab

- GitLab Trigger

Supported authentication methods

- API access token
- OAuth2 (Recommended)

Related resources

Refer to GitLab's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need a GitLab account and:

- The URL of your **GitLab Server**
- An **Access Token**

To set up the credential:

1. In GitLab, select your avatar, then select **Edit profile**.
2. In the left sidebar, select **Access tokens**.
3. Select **Add new token**.
4. Enter a **Name** for the token, like `n8n integration`.
5. Enter an **expiry date** for the token. If you don't enter an expiry date, GitLab automatically sets it to 365 days later than the current date.
    - The token expires on that expiry date at midnight UTC.
6. Select the desired **Scopes**. For the GitLab node, use the `api` scope to easily grant access for all the node's functionality. Or refer to Personal access token scopes to select scopes for the functions you want to use.
7. Select **Create personal access token**.
8. Copy the access token this creates and enter it in your n8n credential as the **Access Token**.
9. Enter the URL of your **GitLab Server** in your n8n credential.

Refer to GitLab's Create a personal access token documentation for more information.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you're self-hosting n8n, you'll need a GitLab account. Then create a new GitLab application:

1. In GitLab, select your avatar, then select **Edit profile**.
2. In the left sidebar, select **Applications**.
3. Select **Add new application**.
4. Enter a **Name** for your application, like `n8n integration`.
5. In n8n, copy the **OAuth Redirect URL**. Enter it as the GitLab **Redirect URI**.

6. Select the desired **Scopes**. For the GitLab node, use the `api` scope to easily grant access for all the node's functionality. Or refer to Personal access token scopes to select scopes for the functions you want to use.
7. Select **Save application**.
8. Copy the **Application ID** and enter it as the **Client ID** in your n8n credential.
9. Copy the **Secret** and enter it as the **Client Secret** in your n8n credential.

Refer to GitLab's Configure GitLab as an OAuth 2.0 authentication identity provider documentation for more information. Refer to the GitLab OAuth 2.0 identity provider API documentation for more information on OAuth2 and GitLab.

Gong credentials

You can use these credentials to authenticate the following nodes:

- Gong

Supported authentication methods

- API access token
- OAuth2

Related resources

Refer to Gong's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need a Gong account and:

- An **Access Key**
- An **Access Key Secret**

You can create both of these items on the Gong API Page (you must be a technical administrator in Gong to access this resource).

Refer to Gong's API documentation for more information about authenticating to the service.

Using OAuth2

To configure this credential, you'll need a Gong account, a Gong developer account and:

- A **Client ID**: Generated when you create an Oauth app for Gong.
- A **Client Secret**: Generated when you create an Oauth app for Gong.

If you're self-hosting n8n, you'll need to create an app to configure OAuth2. Refer to Gong's OAuth documentation for more information about setting up OAuth2.

Google credentials

This section contains:

- OAuth2 single service: Create an OAuth2 credential for a specific service node, such as the Gmail node.
- OAuth2 generic: Create an OAuth2 credential for use with custom operations.
- Service Account: Create a Service Account credential for some specific service nodes.
- Google PaLM and Gemini: Get a Google Gemini/Google PaLM API key.

OAuth2 and Service Account

There are two authentication methods available for Google services nodes:

- OAuth2: Recommended because it's more widely available and easier to set up.
- Service Account: Refer to the Google documentation: Understanding service accounts for guidance on when you need a service account.

–8<– "_snippets/integrations/managed-google-oauth.md"

Compatible nodes

Once configured, you can use your credentials to authenticate the following nodes. Most nodes are compatible with OAuth2 authentication. Support for Service Account authentication is limited.

Node	OAuth	Service Account
Google Ads	:white_check_mark:	:x:
Gmail	:white_check_mark:	:warning:
Google Analytics	:white_check_mark:	:x:
Google BigQuery	:white_check_mark:	:white_check_mark:
Google Books	:white_check_mark:	:white_check_mark:
Google Calendar	:white_check_mark:	:x:
Google Chat	:x:	:white_check_mark:
Google Cloud Storage	:white_check_mark:	:x:

Google Contacts	:white_check_mark:	:x:
Google Cloud Firestore	:white_check_mark:	:white_check_mark:
Google Cloud Natural Language	:white_check_mark:	:x:
Google Cloud Realtime Database	:white_check_mark:	:x:
Google Docs	:white_check_mark:	:white_check_mark:
Google Drive	:white_check_mark:	:white_check_mark:
Google Drive Trigger	:white_check_mark:	:white_check_mark:
Google Perspective	:white_check_mark:	:x:
Google Sheets	:white_check_mark:	:white_check_mark:
Google Slides	:white_check_mark:	:white_check_mark:
Google Tasks	:white_check_mark:	:x:
Google Translate	:white_check_mark:	:white_check_mark:
Google Workspace Admin	:white_check_mark:	:x:
YouTube	:white_check_mark:	:x:

Google: OAuth2 single service

This document contains instructions for creating a Google credential for a single service. They're also available as a video.

–8<– "_snippets/integrations/managed-google-oauth.md"

Prerequisites

- Create a Google Cloud account.

Set up OAuth

There are five steps to connecting your n8n credential to Google services:

1. Create a Google Cloud Console project.
2. Enable APIs.
3. Configure your OAuth consent screen.
4. Create your Google OAuth client credentials.
5. Finish your n8n credential.

Create a Google Cloud Console project

First, create a Google Cloud Console project. If you already have a project, jump to the next section:

–8<– "_snippets/integrations/builtin/credentials/google/create-google-cloud-project.md"

Enable APIs

With your project created, enable the APIs you'll need access to:

–8<– "_snippets/integrations/builtin/credentials/google/enable-apis.md"

Configure your OAuth consent screen

If you haven't used OAuth in your Google Cloud project before, you'll need to configure the OAuth consent screen:

1. Access your Google Cloud Console - Library. Make sure you're in the correct project.

   [Image: The project dropdown in the Google Cloud top navigation]
   Check the project dropdown in the Google Cloud top navigation

2. Open the left navigation menu and go to **APIs & Services > OAuth consent screen**. Google will redirect you to the Google Auth Platform overview page.
3. Select **Get started** on the **Overview** tab to begin configuring OAuth consent.
4. Enter an **App name** and **User support email** to include on the Oauth screen. Select **Next** to continue.
5. For the **Audience**, select **Internal** for user access within your organization's Google workspace or **External** for any user with a Google account. Refer to Google's User type documentation for more information on user types. Select **Next** to continue.
6. Select the **Email addresses** Google should use to contact you about changes to your project. Select **Next** to continue.
7. Read and accept the Google's User Data Policy. Select **Continue** and then select **Create**.
8. In the left-hand menu, select **Branding**.
9. In the **Authorized domains** section, select **Add domain**:
   - If you're using n8n's Cloud service, add `n8n.cloud`
   - If you're self-hosting, add the domain of your n8n instance.
10. Select **Save** at the bottom of the page.

Create your Google OAuth client credentials

Next, create the OAuth client credentials in Google:

1. Access your Google Cloud Console. Make sure you're in the correct project.
2. In the **APIs & Services** section, select **Credentials**.
3. Select **+ Create credentials > OAuth client ID**.
4. In the **Application type** dropdown, select **Web application**.
5. Google automatically generates a **Name**. Update the **Name** to something you'll recognize in your console.
6. **If you're self-hosting:** From your n8n credential, copy the **OAuth Redirect URL**. Paste it into the **Authorized redirect URIs** in Google Console. If you're using **n8n cloud**, you can leave this field empty as the OAuth setup is pre-configured, and the callback URL is fixed for that configuration.
7. Select **Create**.

Finish your n8n credential

With the Google project and credentials fully configured, finish the n8n credential: - If **self-hosted:**

1. From Google's **OAuth client created** modal, copy the **Client ID**. Enter this in your n8n credential.
2. From the same Google modal, copy the **Client Secret**. Enter this in your n8n credential.

- **If n8n cloud:**

1. In n8n, select **Sign in with Google** to complete your Google authentication.
2. **Save** your new credentials.

Video

Troubleshooting

Google hasn't verified this app

–8<– "_snippets/integrations/builtin/credentials/google/unverified-app.md"

Google Cloud app becoming unauthorized

–8<– "_snippets/integrations/builtin/credentials/google/app-becoming-unauthorized.md"

Google: OAuth2 generic

This document contains instructions for creating a generic OAuth2 Google credential for use with custom operations.

–8<– "_snippets/integrations/managed-google-oauth.md"

Prerequisites

- Create a Google Cloud account.

Set up OAuth

There are five steps to connecting your n8n credential to Google services:

1. Create a Google Cloud Console project.
2. Enable APIs.
3. Configure your OAuth consent screen.
4. Create your Google OAuth client credentials.
5. Finish your n8n credential.

Create a Google Cloud Console project

First, create a Google Cloud Console project. If you already have a project, jump to the next section:

–8<– "_snippets/integrations/builtin/credentials/google/create-google-cloud-project.md"

Enable APIs

With your project created, enable the APIs you'll need access to:

–8<– "_snippets/integrations/builtin/credentials/google/enable-apis.md"

Configure your OAuth consent screen

If you haven't used OAuth in your Google Cloud project before, you'll need to configure the OAuth consent screen:

1. Access your Google Cloud Console - Library. Make sure you're in the correct project.

   > [Image: The project dropdown in the Google Cloud top navigation]
   > Check the project dropdown in the Google Cloud top navigation

2. Open the left navigation menu and go to **APIs & Services > OAuth consent screen**. Google will redirect you to the Google Auth Platform overview page.
3. Select **Get started** on the **Overview** tab to begin configuring OAuth consent.
4. Enter an **App name** and **User support email** to include on the Oauth screen. Select **Next** to continue.
5. For the **Audience**, select **Internal** for user access within your organization's Google workspace or **External** for any user with a Google account. Refer to Google's User type documentation for more information on user types. Select **Next** to continue.
6. Select the **Email addresses** Google should use to contact you about changes to your project. Select **Next** to continue.
7. Read and accept the Google's User Data Policy. Select **Continue** and then select **Create**.
8. In the left-hand menu, select **Branding**.

9. In the **Authorized domains** section, select **Add domain**:
    - If you're using n8n's Cloud service, add `n8n.cloud`
    - If you're self-hosting, add the domain of your n8n instance.
10. Select **Save** at the bottom of the page.

Create your Google OAuth client credentials

Next, create the OAuth client credentials in Google:

1. Access your Google Cloud Console. Make sure you're in the correct project.
2. In the **APIs & Services** section, select **Credentials**.
3. Select **+ Create credentials** > **OAuth client ID**.
4. In the **Application type** dropdown, select **Web application**.
5. Google automatically generates a **Name**. Update the **Name** to something you'll recognize in your console.
6. From your n8n credential, copy the **OAuth Redirect URL**. Paste it into the **Authorized redirect URIs** in Google Console.
7. Select **Create**.

Finish your n8n credential

With the Google project and credentials fully configured, finish the n8n credential:

1. From Google's **OAuth client created** modal, copy the **Client ID**. Enter this in your n8n credential.
2. From the same Google modal, copy the **Client Secret**. Enter this in your n8n credential.
3. You must provide the scopes for this credential. Refer to Scopes for more information. Enter multiple scopes in a space-separated list, for example: `https://www.googleapis.com/auth/gmail.labels https://www.googleapis.com/auth/gmail.addons.current.action.compose`
4. In n8n, select **Sign in with Google** to complete your Google authentication.
5. **Save** your new credentials.

Video

The following video demonstrates the steps described above:

Scopes

Google services have one or more possible access scopes. A scope limits what a user can do. Refer to OAuth 2.0 Scopes for Google APIs for a list of scopes for all services.

n8n doesn't support all scopes. When creating a generic Google OAuth2 API credential, you can enter scopes from the **Supported scopes** list below. If you enter a scope that n8n doesn't already support, it won't work.

??? Details "Supported scopes" | Service | Available scopes | | ———————————————–| ———————————————————————————————

———––| | Gmail |

- https://www.googleapis.com/auth/gmail.labels
- https://www.googleapis.com/auth/gmail.addons.current.action.compose
- https://www.googleapis.com/auth/gmail.addons.current.message.action
- https://mail.google.com/
- https://www.googleapis.com/auth/gmail.modify
- https://www.googleapis.com/auth/gmail.compose

| | Google Ads |

- https://www.googleapis.com/auth/adwords

```
|
| Google Analytics                     | <ul>
<li>`https://www.googleapis.com/auth/analytics`</li>
<li>`https://www.googleapis.com/auth/analytics.readonly`</li></ul>
|
| Google BigQuery                      | <ul>
<li>`https://www.googleapis.com/auth/bigquery`</li></ul>
|
| Google Books                         | <ul>
<li>`https://www.googleapis.com/auth/books`</li></ul>
|
| Google Calendar                      | <ul>
<li>`https://www.googleapis.com/auth/calendar`</li>
<li>`https://www.googleapis.com/auth/calendar.events`</li></ul>
|
| Google Cloud<br> Natural Language    | <ul>
<li>`https://www.googleapis.com/auth/cloud-language`</li>
<li>`https://www.googleapis.com/auth/cloud-platform`</li></ul>
|
| Google Cloud<br>Storage              | <ul>
<li>`https://www.googleapis.com/auth/cloud-platform`</li>
<li>`https://www.googleapis.com/auth/cloud-platform.read-only`</li>
<li>`https://www.googleapis.com/auth/devstorage.full_control`</li>
<li>`https://www.googleapis.com/auth/devstorage.read_only`</li>
<li>`https://www.googleapis.com/auth/devstorage.read_write`</li>
</ul>                                              |
| Google Contacts                      | <ul>
<li>`https://www.googleapis.com/auth/contacts`</li></ul>
|
| Google Docs                          | <ul>
<li>`https://www.googleapis.com/auth/documents`</li>
<li>`https://www.googleapis.com/auth/drive`</li>
<li>`https://www.googleapis.com/auth/drive.file`</li></ul>
|
| Google Drive                         | <ul>
<li>`https://www.googleapis.com/auth/drive`</li>
<li>`https://www.googleapis.com/auth/drive.appdata`</li>
<li>`https://www.googleapis.com/auth/drive.photos.readonly`</li>
</ul>
|
| Google Firebase<br>Cloud Firestore   | <ul>
<li>`https://www.googleapis.com/auth/datastore`</li>
<li>`https://www.googleapis.com/auth/firebase`</li></ul>
```

```
|
| Google Firebase<br>Realtime Database | <ul>
<li>`https://www.googleapis.com/auth/userinfo.email`</li>
<li>`https://www.googleapis.com/auth/firebase.database`</li>
<li>`https://www.googleapis.com/auth/firebase`</li></ul>
|
| Google Perspective                   | <ul>
<li>`https://www.googleapis.com/auth/userinfo.email`</li></ul>
|
| Google Sheets                        | <ul>
<li>`https://www.googleapis.com/auth/drive.file`</li>
<li>`https://www.googleapis.com/auth/spreadsheets`</li></ul>
|
| Google Slide                         | <ul>
<li>`https://www.googleapis.com/auth/drive.file`</li>
<li>`https://www.googleapis.com/auth/presentations`</li></ul>
|
| Google Tasks                         | <ul>
<li>`https://www.googleapis.com/auth/tasks`</li></ul>
|
| Google Translate                     | <ul>
<li>`https://www.googleapis.com/auth/cloud-translation`</li></ul>
|
| GSuite Admin                         | <ul>
<li>`https://www.googleapis.com/auth/admin.directory.group`</li>
<li>`https://www.googleapis.com/auth/admin.directory.user`</li>
<li>`https://www.googleapis.com/auth/admin.directory.domain.readonly`
<li>`https://www.googleapis.com/auth/admin.directory.userschema.reado
</ul>
|
```

Troubleshooting

Google hasn’t verified this app

–8<– “_snippets/integrations/builtin/credentials/google/unverified-app.md”

Google Cloud app becoming unauthorized

–8<– “_snippets/integrations/builtin/credentials/google/app-becoming-unauthorized.md”

Google: Service Account

Using service accounts is more complex than OAuth2. Before you begin:

- Check if your node is compatible with Service Account.
- Make sure you need to use Service Account. For most use cases, OAuth2 is a better option.
- Read the Google documentation on Creating and managing service accounts.

Prerequisites

- Create a Google Cloud account.

Set up Service Account

There are four steps to connecting your n8n credential to a Google Service Account:

1. Create a Google Cloud Console project.
2. Enable APIs.
3. Set up Google Cloud Service Account.
4. Finish your n8n credential.

Create a Google Cloud Console project

First, create a Google Cloud Console project. If you already have a project, jump to the next section:

–8<– “_snippets/integrations/builtin/credentials/google/create-google-cloud-project.md”

Enable APIs

With your project created, enable the APIs you’ll need access to:

–8<– “_snippets/integrations/builtin/credentials/google/enable-apis.md”

Set up Google Cloud Service Account

1. Access your Google Cloud Console - Library. Make sure you’re in the correct project.

   [Image: The project dropdown in the Google Cloud top navigation]

   Check the project dropdown in the Google Cloud top navigation

2. Open the left navigation menu and go to **APIs & Services > Credentials**. Google takes you to your **Credentials** page.
3. Select **+ Create credentials > Service account**.
4. Enter a name in **Service account name** and an ID in **Service account ID**. Refer to Creating a service account for more information.
5. Select **Create and continue**.
6. Based on your use-case, you may want to **Select a role** and **Grant users access to this service account** using the corresponding sections.
7. Select **Done**.

8. Select your newly created service account under the **Service Accounts** section. Open the **Keys** tab.
9. Select **Add key > Create new key**.
10. In the modal that appears, select **JSON**, then select **CREATE**. Google saves the file to your computer.

Finish your n8n credential

With the Google project and credentials fully configured, finish the n8n credential:

1. Open the downloaded JSON file.
2. Copy the `client_email` and enter it in your n8n credential as the **Service Account Email**.
3. Copy the `private_key`. Don't include the surrounding `"` marks. Enter this as the **Private Key** in your n8n credential.
4. **Optional**: Choose if you want to **Impersonate a User** (turned on).
    1. To use this option, you must Enable domain-wide delegation for the service account as a Google Workspace super admin.
    2. Enter the **Email** of the user you want to impersonate.
5. If you plan to use this credential with the HTTP Request node, turn on **Set up for use in HTTP Request node**.
    1. With this setting turned on, you'll need to add **Scope(s)** for the node. n8n prepopulates some scopes. Refer to OAuth 2.0 Scopes for Google APIs for more information.
6. **Save** your credentials.

Video

Troubleshooting

Service Account can't access Google Drive files

A Service Account can't access Google Drive files and folders that weren't shared with its associated user email.

1. Access your Google Cloud Console and copy your Service Account email.
2. Access your Google Drive and go to the designated file or folder.
3. Right-click on the file or folder and select **Share**.
4. Paste your Service Account email into **Add People and groups**.
5. Select **Editor** for read-write access or **Viewer** for read-only access.

Enable domain-wide delegation

To impersonate a user with a service account, you must enable domain-wide delegation for the service account.

To delegate domain-wide authority to a service account, you must be a super administrator for the Google Workspace domain. Then:

1. From your Google Workspace domain's Admin console, select the hamburger menu, then select **Security > Access and data control > API Controls**.
2. In the **Domain wide delegation** pane, select **Manage Domain Wide Delegation**.
3. Select **Add new**.
4. In the **Client ID** field, enter the service account's **Client ID**. To get the Client ID:
   - Open your Google Cloud Console project, then open the Service Accounts page.
   - Copy the **OAuth 2 Client ID** and use this as the **Client ID** for the **Domain Wide Delegation**.
5. In the **OAuth scopes** field, enter a list of comma-separate scopes to grant your application access. For example, if your application needs domain-wide full access to the Google Drive API and the Google Calendar API, enter:
   `https://www.googleapis.com/auth/drive`, `https://www.googleapis.com/auth/calendar`.
6. Select **Authorize**.

It can take from 5 minutes up to 24 hours before you can impersonate all users in your Workspace.

Google Gemini(PaLM) credentials

You can use these credentials to authenticate the following nodes:

- Embeddings Google Gemini
- Google Gemini
- Google Gemini Chat Model
- Embeddings Google PaLM

Prerequisites

- Create a Google Cloud account.
- Create a Google Cloud Platform project.

Supported authentication methods

- Gemini(PaLM) API key

Related resources

Refer to Google's Gemini API documentation for more information about the service.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using Gemini(PaLM) API key

To configure this credential, you'll need:

- The API **Host** URL: Both PaLM and Gemini use the default `https://generativelanguage.googleapis.com`.
- An **API Key**: Create a key in Google AI Studio.

To create an API key:

1. Go to the API Key page in Google AI Studio: https://aistudio.google.com/apikey.
2. Select **Create API Key**.
3. You can choose whether to **Create API key in new project** or search for an existing Google Cloud project to **Create API key in existing project**.
4. Copy the generated API key and add it to your n8n credential.

Gotify credentials

You can use these credentials to authenticate the following nodes:

- Gotify

Prerequisites

Install Gotify on your server.

Supported authentication methods

- API token

Related resources

Refer to Gotify's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- An **App API Token**: Only required if you'll use this credential to create messages. To generate an App API token, create an application from the **Apps** menu. Refer to Gotify's Push messages documentation for more information.
- A **Client API Token**: Required for all actions other than creating messages (such as deleting or retrieving messages). To generate a Client API token, create a client from the **Clients** menu.
- The **URL** of the Gotify host

GoTo Webinar credentials

You can use these credentials to authenticate the following nodes:

- GoToWebinar

Prerequisites

Create a GoToWebinar account with Developer Center access.

Supported authentication methods

- OAuth2

Related resources

Refer to GoToWebinar's API documentation for more information about authenticating with the service.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Provided once you create an OAuth client
- A **Client Secret**: Provided once you create an OAuth client

Refer to the Create an OAuth client documentation for detailed instructions on creating an OAuth client. Copy the **OAuth Callback URL** from n8n to use as the **Redirect URI** in your OAuth client. The Client ID and Client secret are provided once you've finished setting up your client.

Grafana credentials

You can use these credentials to authenticate the following nodes:

- Grafana

Prerequisites

- Create a Grafana account.

Supported authentication methods

- API key

Related resources

Refer to Grafana's API documentation for more information about authenticating with the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to the Create an API key documentation for detailed instructions on creating an API key.
- The **Base URL** for your Grafana instance, for example: `https://n8n.grafana.net`.

Grist credentials

You can use these credentials to authenticate the following nodes:

- Grist

Prerequisites

Create a Grist account.

Supported authentication methods

- API key

Related resources

Refer to Grist's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to the Grist API authentication documentation for instructions on creating an API key.
- To select your Grist **Plan Type**. Options include:
  - Free
  - Paid: If selected, provide your Grist **Custom Subdomain**. This is the portion that comes before `.getgrist.com`. For example, if our full Grist domain was `n8n.getgrist.com`, we'd enter `n8n` here.
  - Self-Hosted: If selected, provide your Grist **Self-Hosted URL**. This should be the full URL.

Groq credentials

You can use these credentials to authenticate the following nodes:

- Groq Chat Model

Prerequisites

Create a Groq account.

Supported authentication methods

- API key

Related resources

Refer to Groq's documentation for more information about the service.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

To configure this credential, you'll need:

- An **API Key**

To get your API key:

1. Go to the API Keys page of your Groq console.
2. Select **Create API Key**.
3. Enter a **display name** for the key, like `n8n integration`, and select **Submit**.
4. Copy the key and paste it into your n8n credential.

Refer to Groq's API Keys documentation for more information.

Gumroad credentials

You can use these credentials to authenticate the following nodes:

- Gumroad Trigger

Prerequisites

Create a Gumroad account.

Supported authentication methods

- API access token

Related resources

Refer to Gumroad's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- An API **Access Token**: Create an application to generate an access token. Refer to the Gumroad Create an application for the API documentation for detailed instructions on creating a new application and generating an access token.

HaloPSA credentials

You can use these credentials to authenticate the following nodes:

- HaloPSA

Prerequisites

Create a HaloPSA account.

Supported authentication methods

- API key

Related resources

Refer to HaloPSA's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- To select your **Hosting Type**:
  - **On Premise Solution**: Choose this option if you're hosting the Halo application on your own server
  - **Hosted Solution Of Halo**: Choose this option if your application is hosted by Halo. If this option is selected, you'll need to provide your **Tenant**.
- The **HaloPSA Authorisation Server URL**: Your Authorisation Server URL is displayed within HaloPSA in **Configuration > Integrations > Halo API** in API Details.
- The **Resource Server** URL: Your Resource Server is displayed within HaloPSA in **Configuration > Integrations > Halo API** in API Details.
- A **Client ID**: Obtained by registering the application in the Halo API settings. Refer to HaloPSA's Authorisation documentation for detailed instructions. n8n recommends using these settings:
  - Choose `Client Credentials` as your **Authentication Method**.

    - Use the `all` permission.
- A **Client Secret**: Obtained by registering the application in the Halo API settings.
- Your **Tenant** name: If **Hosted Solution of Halo** is selected as the **Hosting Type**, you must provide your tenant name. Your tenant name is displayed within HaloPSA in **Configuration > Integrations > Halo API** in API Details.

HaloPSA uses both the application permissions and the agent's permissions to determine API access.

Harvest credentials

You can use these credentials to authenticate the following nodes:

- Harvest

Prerequisites

Create a Harvest account.

Supported authentication methods

- API access token
- OAuth2

Related resources

Refer to Harvest's API documentation for more information about the service.

Using API Access Token

To configure this credential, you'll need:

- A Personal **Access Token**: Refer to the Harvest Personal Access Token Authentication documentation for instructions on creating a personal access token.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you need to configure OAuth2 from scratch or need more detail on what's happening in the OAuth web flow, refer to the instructions in the Harvest OAuth2 documentation to set up OAuth.

Help Scout credentials

You can use these credentials to authenticate the following nodes:

- Help Scout
- Help Scout Trigger

Prerequisites

Create a Help Scout account.

Supported authentication methods

- OAuth2

Related resources

Refer to Help Scout's API documentation for more information about the service.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you need to configure OAuth2 from scratch or need more detail on what's happening in the OAuth web flow, you'll need to create a Help Scout app. Refer to the instructions in the Help Scout OAuth documentation for more information.

HighLevel credentials

You can use these credentials to authenticate the following nodes:

- HighLevel node

Prerequisites

Create a HighLevel developer account.

Supported authentication methods

- API key: Use with API v1
- OAuth2: Use with API v2

Related resources

Refer to HighLevel's API 2.0 documentation for more information about the service.

For existing integrations with the API v1.0, refer to HighLevel's API 1.0 documentation.

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to the HighLevel API 1.0 Welcome documentation for instructions on getting your API key.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**
- A **Client Secret**

To generate both, create an app in **My Apps > Create App**. Use these settings:

1. Set **Distribution Type** to **Sub-Account**.
2. Add these **Scopes**:
    - `locations.readonly`
    - `contacts.readonly`
    - `contacts.write`
    - `opportunities.readonly`
    - `opportunities.write`
    - `users.readonly`
3. Copy the **OAuth Redirect URL** from n8n and add it as a **Redirect URL** in your HighLevel app.
4. Copy the **Client ID** and **Client Secret** from HighLevel and add them to your n8n credential.
5. Add the same scopes added above to your n8n credential in a space-separated list. For example:

    ```
    locations.readonly contacts.readonly contacts.write
    opportunities.readonly opportunities.write users.readonly
    ```

Refer to HighLevel's API Authorization documentation for more details. Refer to HighLevel's API Scopes documentation for more information about available scopes.

Home Assistant credentials

You can use these credentials to authenticate the following nodes:

- Home Assistant

Supported authentication methods

- API access token

Related resources

Refer to Home Assistant's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need to Install Home Assistant, create a Home Assistant account, and have:

- Your **Host**
- The **Port**
- A Long-Lived **Access Token**

To generate an access token and set up the credential:

1. To generate your **Access Token**, log in to Home Assistant and open your User profile.
2. In the **Long-Lived Access Tokens** section, generate a new token.
3. Copy this token and enter it in n8n as your **Access Token**.
4. Enter the URL or IP address of your Home Assistant **Host**, without the `http://` or `https://` protocol, for example `your.awesome.home`.
5. For the **Port**, enter the appropriate port:
   - If you've made no port changes and access Home Assistant at `http://`, keep the default of `8123`.
   - If you've made no port changes and access Home Assistant at `https://`, enter `443`.
   - If you've configured Home Assistant to use a specific port, enter that port.
6. If you've enabled SSL in Home Assistant in the config.yml map key, turn on the **SSL** toggle in n8n. If you're not sure, it's best to turn this setting on if you access your home assistant UI using `https://` instead of `http://`.

HTTP Request credentials

You can use these credentials to authenticate the following nodes:

- HTTP Request
- HTTP Request Tool (legacy)

Prerequisites

You must use the authentication method required by the app or service you want to query.

If you need to secure the authentication with an SSL certificate, refer to Provide an SSL certificate for the information you'll need.

Supported authentication methods

- Predefined credential type
- Basic auth (generic credential type)
- Custom auth (generic credential type)
- Digest auth (generic credential type)
- Header auth (generic credential type)
- Bearer auth (generic credential type)
- OAuth1 (generic credential type)
- OAuth2 (generic credential type)
- Query auth (generic credential type)

Refer to HTTP authentication for more information relating to generic credential types.

Using predefined credential type

–8<– “_snippets/integrations/predefined-credential-type-how-to.md”

Refer to Custom API operations for more information.

–8<– “_snippets/integrations/builtin/credentials/generic-auth/basic-auth.md”

–8<– “_snippets/integrations/builtin/credentials/generic-auth/digest-auth.md”

–8<– “_snippets/integrations/builtin/credentials/generic-auth/header-auth.md”

–8<– “_snippets/integrations/builtin/credentials/generic-auth/bearer-auth.md”

Using OAuth1

Use this generic authentication if your app or service supports OAuth1 authentication.

To configure this credential, enter:

- An **Authorization URL**: Also known as the Resource Owner Authorization URI. This URL typically ends in `/oauth1/authorize`. The temporary credentials are sent here to prompt a user to complete authorization.
- An **Access Token URL**: This is the URI used for the initial request for temporary credentials. This URL typically ends in `/oauth1/request` or `/oauth1/token`.
- A **Consumer Key**: Also known as the client key, like a username. This specifies the `oauth_consumer_key` to use for the call.
- A **Consumer Secret**: Also known as the client secret, like a password.
- A **Request Token URL**: This is the URI used to switch from temporary credentials to long-lived credentials after authorization. This URL typically ends in `/oauth1/access`.
- Select the **Signature Method** the auth handshake uses. This specifies the `oauth_signature_method` to use for the call. Options include:
  - **HMAC-SHA1**

    - **HMAC-SHA256**
    - **HMAC-SHA512**

For most OAuth1 integrations, you'll need to configure an app, service, or integration to generate the values for most of these fields. Use the **OAuth Redirect URL** in n8n as the redirect URL or redirect URI for such a service.

Read more about OAuth1 and the OAuth1 authorization flow.

Using OAuth2

Use this generic authentication if your app or service supports OAuth2 authentication.

Requirements to configure this credential depend on the **Grant Type** selected. Refer to OAuth Grant Types for more information on each grant type.

For most OAuth2 integrations, you'll need to configure an app, service, or integration. Use the **OAuth Redirect URL** in n8n as the redirect URL or redirect URI for such a service.

Read more about OAuth2.

Authorization Code grant type

Use Authorization Code grant type to exchange an authorization code for an access token. The auth flow uses the redirect URL to return the user to the client. Then the application gets the authorization code from the URL and uses it to request an access token. Refer to Authorization Code Request for more information.

To configure this credential, select **Authorization Code** as the **Grant Type**.

Then enter:

- An **Authorization URL**
- An **Access Token URL**
- A **Client ID**: The ID or username to log in with.
- A **Client Secret**: The secret or password used to log in with.
- *Optional:* Enter one or more **Scope**s for the credential. If unspecified, the credential will request all scopes available to the client.
- *Optional:* Some services require more query parameters. If your service does, add them as **Auth URI Query Parameters**.
- An **Authentication** type: Select the option that best suits your use case. Options include:
    - **Header**: Send the credentials as a basic auth header.
    - **Body**: Send the credentials in the body of the request.
- *Optional:* Choose whether to **Ignore SSL Issues**. If turned on, n8n will connect even if SSL validation fails.

Client Credentials grant type

Use the Client Credentials grant type when applications request an access token to access their own resources, not on behalf of a user. Refer to Client Credentials for more information.

To configure this credential, select **Client Credentials** as the **Grant Type**.

Then enter:

- An **Access Token URL**: The URL to hit to begin the OAuth2 flow. Typically this URL ends in `/token`.
- A **Client ID**: The ID or username to use to log in to the client.
- A **Client Secret**: The secret or password used to log in to the client.
- *Optional:* Enter one or more **Scope**s for the credential. Most services don't support scopes for Client Credentials grant types; only enter scopes here if yours does.
- An **Authentication** type: Select the option that best suits your use case. Options include:
  - **Header**: Send the credentials as a basic auth header.
  - **Body**: Send the credentials in the body of the request.
- *Optional:* Choose whether to **Ignore SSL Issues**. If turned on, n8n will connect even if SSL validation fails.

PKCE grant type

Proof Key for Code Exchange (PKCE) grant type is an extension to the Authorization Code flow to prevent CSRF and authorization code injection attacks.

To configure this credential, select **PKCE** as the **Grant Type**.

Then enter:

- An **Authorization URL**
- An **Access Token URL**
- A **Client ID**: The ID or username to log in with.
- A **Client Secret**: The secret or password used to log in with.
- *Optional:* Enter one or more **Scope**s for the credential. If unspecified, the credential will request all scopes available to the client.
- *Optional:* Some services require more query parameters. If your service does, add them as **Auth URI Query Parameters**.
- An **Authentication** type: Select the option that best suits your use case. Options include:
  - **Header**: Send the credentials as a basic auth header.
  - **Body**: Send the credentials in the body of the request.
- *Optional:* Choose whether to **Ignore SSL Issues**. If turned on, n8n will connect even if SSL validation fails.

Using query auth

Use this generic authentication if your app or service supports passing authentication as a single key/value query parameter. (For multiple query parameters, use Custom Auth.)

To configure this credential, enter:

- A query parameter key or **Name**

- A query parameter **Value**

Using custom auth

Use this generic authentication if your app or service supports passing authentication as multiple key/value query parameters or you need more flexibility than the other generic auth options.

The **Custom Auth** credential expects JSON data to define your credential. You can use `headers`, `qs`, `body` or a mix. Review the examples below to get started.

Sending two headers

```
{
    "headers": {
        "X-AUTH-USERNAME": "username",
        "X-AUTH-PASSWORD": "password"
    }
}
```

Body

```
{
     "body" : {
        "user": "username",
        "pass": "password"
    }
}
```

Query string

```
{
    "qs": {
        "appid": "123456",
        "apikey": "my-api-key"
    }
}
```

Sending header and query string

```
{
    "headers": {
        "api-version": "202404"
    },
    "qs": {
        "apikey": "my-api-key"
    }
}
```

Provide an SSL certificate

You can send an SSL certificate with your HTTP request. Create the SSL certificate as a separate credential for use by the node:

1. In the HTTP Request node **Settings**, turn on **SSL Certificates**.

2. On the **Parameters** tab, add an existing SSL Certificate credential to **Credential for SSL Certificates** or create a new one.

To configure your SSL Certificates credential, you'll need to add:

- The Certificate Authority **CA** bundle
- The **Certificate** (CRT): May also appear as a Public Key, depending on who your issuing CA was and how they format the cert
- The **Private Key** (KEY)
- *Optional:* If the **Private Key** is encrypted, enter a **Passphrase** for the private key.

If your SSL certificate is in a single file (such as a `.pfx` file), you'll need to open the file to copy details from it to paste into the appropriate fields:

- Enter the Public Key/CRT as the **Certificate**
- Enter the **Private Key**/KEY in that field

HubSpot credentials

You can use these credentials to authenticate the following nodes:

- HubSpot
- HubSpot Trigger

Supported authentication methods

- App token: Use with the HubSpot node.
- Developer API key: Use with the HubSpot Trigger node.
- OAuth2: Use with the HubSpot node.

Related resources

Refer to HubSpot's API documentation for more information about the service. The HubSpot Trigger node uses the Webhooks API; refer to HubSpot's Webhooks API documentation for more information about that service.

Using App token

To configure this credential, you'll need a HubSpot account or HubSpot developer account and:

- An **App Token**

To generate an app token, create a private app in HubSpot:

1. In your HubSpot account, select the **settings icon** in the main navigation bar.
2. In the left sidebar menu, go to **Integrations > Private Apps**.
3. Select **Create private app**.
4. On the **Basic Info** tab, enter your app's **Name**.
5. Hover over the **placeholder logo** and select the upload icon to

upload a square image that will serve as the logo for your app.
6. Enter a **Description** for your app.
7. Open the **Scopes** tab and add the appropriate scopes. Refer to Required scopes for HubSpot node for a complete list of scopes you should add.
8. Select **Create app** to finish the process.
9. In the modal, review the info about your app's access token, then select **Continue creating**.
10. Once your app's created, open the **Access token card** and select **Show token** to reveal the token.
11. Copy this token and enter it in your n8n credential.

Refer to the HubSpot Private Apps documentation for more information.

Using Developer API key

To configure this credential, you'll need a HubSpot developer account and:

- A **Client ID**: Generated once you create a public app.
- A **Client Secret**: Generated once you create a public app.
- A **Developer API Key**: Generated from your Developer Apps dashboard.
- An **App ID**: Generated once you create a public app.

To create the public app and set up the credential:

1. Log into your HubSpot app developer account.
2. Select **Apps** from the main navigation bar.
3. Select **Get HubSpot API key**. You may need to select the option to **Show key**.
4. Copy the key and enter it in n8n as the **Developer API Key**.
5. Still on the HubSpot **Apps** page, select **Create app**.
6. On the **App Info** tab, add an **App name**, **Description**, **Logo**, and any support contact info you want to provide. Anyone encountering the app would see these.
7. Open the **Auth** tab.
8. Copy the **App ID** and enter it in n8n.
9. Copy the **Client ID** and enter it in n8n.
10. Copy the **Client Secret** and enter it in n8n.
11. In the **Scopes** section, select **Add new scope**.
12. Add all the scopes listed in Required scopes for HubSpot Trigger node to your app.
13. Select **Update**.
14. Copy the n8n **OAuth Redirect URL** and enter it as the **Redirect URL** in your HubSpot app.
15. Select **Create app** to finish creating the HubSpot app.

Refer to the HubSpot Public Apps documentation for more detailed instructions.

Required scopes for HubSpot Trigger node

If you're creating an app for use with the HubSpot Trigger node, n8n recommends starting with these scopes:

Element	Object	Permission	Scope name

n/a	n/a	n/a	`oauth`
CRM	Companies	Read	`crm.objects.companies.read`
CRM	Companies schemas	Read	`crm.schemas.companies.read`
CRM	Contacts	Read	`crm.objects.contacts.read`
CRM	Contacts schemas	Read	`crm.schemas.contacts.read`
CRM	Deals	Read	`crm.objects.deals.read`
CRM	Deals schemas	Read	`crm.schemas.deals.read`

Using OAuth2

–8<– “_snippets/integrations/builtin/credentials/cloud-oauth-button.md”

If you’re self-hosting n8n, you’ll need to configure OAuth2 from scratch by creating a new public app:

1. Log into your HubSpot app developer account.
2. Select **Apps** from the main navigation bar.
3. Select **Create app**.
4. On the **App Info** tab, add an **App name**, **Description**, **Logo**, and any support contact info you want to provide. Anyone encountering the app would see these.
5. Open the **Auth** tab.
6. Copy the **App ID** and enter it in n8n.
7. Copy the **Client ID** and enter it in n8n.
8. Copy the **Client Secret** and enter it in n8n.
9. In the **Scopes** section, select **Add new scope**.
10. Add all the scopes listed in Required scopes for HubSpot node to your app.
11. Select **Update**.
12. Copy the n8n **OAuth Redirect URL** and enter it as the **Redirect URL** in your HubSpot app.
13. Select **Create app** to finish creating the HubSpot app.

Refer to the HubSpot Public Apps documentation for more detailed instructions. If you need more detail on what’s happening in the OAuth web flow, refer to the HubSpot Working with OAuth documentation.

Required scopes for HubSpot node

If you’re creating an app for use with the HubSpot node, n8n recommends starting with these scopes:

Element	Object	Permission	Scope name(s)
n/a	n/a	n/a	`oauth`
n/a	n/a	n/a	`forms`
n/a	n/a	n/a	`tickets`
CRM	Companies	Read Write	`crm.objects.companies.read` `crm.objects.companies.write`

CRM	Companies schemas	Read	`crm.schemas.companies.read`
CRM	Contacts schemas	Read	`crm.schemas.contacts.read`
CRM	Contacts	Read Write	`crm.objects.contacts.read` `crm.objects.contacts.write`
CRM	Deals	Read Write	`crm.objects.deals.read` `crm.objects.deals.write`
CRM	Deals schemas	Read	`crm.schemas.deals.read`
CRM	Owners	Read	`crm.objects.owners.read`
CRM	Lists	Write	`crm.lists.write`

Hugging Face credentials

You can use these credentials to authenticate the following nodes:

- Hugging Face Inference
- Embeddings Hugging Face Inference

Supported authentication methods

- API key

Related resources

Refer to Hugging Face's documentation for more information about the service.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

To configure this credential, you'll need a Hugging Face account and:

- An **API Key**: Hugging Face calls these API tokens.

To get your API token:

1. Open your Hugging Face profile and go to the **Tokens** section.
2. Copy the token listed there. It should begin with `hf_`.
3. Enter this API token as your n8n credential **API Key**.

Refer to Get your API token for more information.

Humantic AI credentials

You can use these credentials to authenticate the following nodes:

- Humantic AI

Prerequisites

Create a Humantic AI account.

You can also try out an API key as a free trial at the Humantic AI API page.

Supported authentication methods

- API key

Related resources

Refer to Humantic AI's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Get an API key from the Humantic AI API page.

Hunter credentials

You can use these credentials to authenticate the following nodes:

- Hunter

Prerequisites

Create a Hunter account.

Supported authentication methods

- API key

Related resources

Refer to Hunter's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Generate an API key from your profile in the dashboard. Refer to the Hunter API Authentication documentation for more information.

Hybrid Analysis credentials

–8<– “_snippets/integrations/builtin/credentials/cred-only-statement.md”

Prerequisites

Create a Hybrid Analysis account.

Supported authentication methods

- API key

Related resources

Refer to Hybrid Analysis’ API documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n’s website.

Using API key

To configure this credential, you’ll need:

- An **API Key**: Refer to the Hybrid Analysis’ API documentation for instructions on generating an API key.

IMAP credentials

You can use these credentials to authenticate the following nodes:

- IMAP Email

Prerequisites

Create an email account on a service with IMAP support.

Supported authentication methods

- User account

Related resources

Internet Message Access Protocol (IMAP) is a standard protocol for receiving email. Most email providers offer instructions on setting up their service with IMAP; refer to your provider's IMAP instructions.

Using user account

To configure this credential, you'll need:

- A **User** name: The email address you're retrieving email for.
- A **Password**: Either the password you use to check email or an app password. Your provider will tell you whether to use your own password or to generate an app password.
- A **Host**: The IMAP host address for your email provider, often formatted as `imap.<provider>.com`. Check with your provider.
- A **Port** number: The default is port `993`. Use this port unless your provider or email administrator tells you to use something different.

Choose whether to use **SSL/TLS** and whether to **Allow Self-Signed Certificates**.

Provider instructions

Refer to the quickstart guides for these common email providers.

Gmail

Refer to Gmail.

Outlook.com

Refer to Outlook.com.

Yahoo

Refer to Yahoo.

My provider isn't listed

If your email provider isn't listed here, search for their `IMAP settings` or `IMAP instructions`.

Gmail IMAP credentials

Follow these steps to configure the IMAP credentials with a Gmail account.

Prerequisites

To follow these instructions, you must first:

1. Enable 2-step Verification on your Gmail account.
2. Generate an app password.

Enable 2-step Verification

–8<– “_snippets/integrations/builtin/credentials/email/gmail-two-step-verification.md”

Generate an app password

–8<– “_snippets/integrations/builtin/credentials/email/gmail-app-password.md”

Set up the credential

To set up the IMAP credential with a Gmail account, use these settings:

1. Enter your Gmail email address as the **User**.
2. Enter the app password you generated above as the **Password**.
3. Enter `imap.gmail.com` as the **Host**.
4. For the **Port**, keep the default port number of `993`. Check with your email administrator if this port doesn’t work.
5. Turn on the **SSL/TLS** toggle.
6. Check with your email administrator about whether to **Allow Self-Signed Certificates**.

Refer to Add Gmail to another client for more information. You may need to **Enable IMAP** if you’re using a personal Google account before June 2024.

Outlook.com IMAP credentials

Follow these steps to configure the IMAP credentials with an Outlook.com account.

Set up the credentials

To set up the IMAP credential with Outlook.com account, use these settings:

1. Enter your Outlook.com email address as the **User**.
2. Enter your Outlook.com password as the **Password**.
3. Enter `outlook.office365.com` as the **Host**.
4. For the **Port**, keep the default port number of `993`.
5. Turn on the **SSL/TLS** toggle.
6. Check with your email administrator about whether to **Allow Self-Signed Certificates**.

Refer to Microsoft's POP, IMAP, and SMTP settings for Outlook.com documentation for more information.

Connection errors

You may receive a connection error if you configured your Outlook.com account as IMAP in multiple email clients. Microsoft is working on a fix for this. For now, try this workaround:

1. Go to account.live.com/activity and sign in using the email address and password of the affected account.
2. Under **Recent activity**, find the **Session Type** event that matches the most recent time you received the connection error. Select it to expand the details.
3. Select **This was me** to approve the IMAP connection.
4. Retest your n8n credential.

Refer to What is the Recent activity page? for more information on using this page.

The source for these instructions is Outlook.com IMAP connection errors. Refer to that documentation for more information.

Use an app password

–8<– "_snippets/integrations/builtin/credentials/email/outlook-app-password.md"

Yahoo IMAP credentials

Follow these steps to configure the IMAP credentials with a Yahoo account.

Prerequisites

To follow these instructions, you must first generate an app password:

–8<– "_snippets/integrations/builtin/credentials/email/yahoo-app-password.md"

Set up the credential

To set up the IMAP credential with a Yahoo Mail account, use these settings:

1. Enter your Yahoo email address as the **User**.
2. Enter the app password you generated above as the **Password**.
3. Enter `imap.mail.yahoo.com` as the **Host**.
4. Keep the default **Port** number of `993`. Check with your email administrator if this port doesn't work.
5. Turn on the **SSL/TLS** toggle.
6. Check with your email administrator about whether to **Allow Self-**

Signed Certificates.

Refer to Set up IMAP for Yahoo mail account for more information.

Imperva WAF credentials

–8<– “_snippets/integrations/builtin/credentials/cred-only-statement.md”

Prerequisites

Create an Imperva WAF account.

Supported authentication methods

- API key

Related resources

Refer to Imperva WAF’s documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n’s website.

Using API key

To configure this credential, you’ll need:

- An **API ID**
- An **API Key**

Refer to Imperva WAF’s API Key Management documentation for instructions on generating and viewing API Keys and IDs.

Intercom credentials

You can use these credentials to authenticate the following nodes:

- Intercom

Prerequisites

- Create an Intercom developer account.
- Create an app in your developer hub.

Supported authentication methods

- API key

Related resources

Refer to Intercom's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Intercom automatically generates an **Access Token** when you create an app. Use this **Access Token** as your n8n **API Key**. Refer to How to get your Access Token for more detailed instructions.

Invoice Ninja credentials

You can use these credentials to authenticate the following nodes:

- Invoice Ninja
- Invoice Ninja Trigger

Prerequisites

Create an Invoice Ninja account. Only the Pro and Enterprise plans support API integrations.

Supported authentication methods

- API key

Related resources

Refer to Invoice Ninja's v4 API documentation and v5 API documentation for more information about the APIs.

Using API key

To configure this credential, you'll need:

- A **URL**: If Invoice Ninja hosts your installation, use either of the default URLs mentioned. If you're self-hosting your installation, use the URL of your Invoice Ninja instance.
- An **API Token**: Generate an API token in **Settings > Account Management > API Tokens**.
- An optional **Secret**, available only for v5 API users

Iterable credentials

You can use these credentials to authenticate the following nodes:

- Iterable

Prerequisites

Create an Iterable account.

Supported authentication methods

- API key

Related resources

Refer to Iterable's API documentation for more information about the service:

- US-based Iterable projects
- Europe-based Iterable projects

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to Iterable's Creating API keys documentation for instructions on creating API keys.

Jenkins credentials

You can use these credentials to authenticate the following nodes:

- Jenkins

Prerequisites

Create an account on a Jenkins instance.

Supported authentication methods

- API token

Related resources

Jenkins doesn't provide public API documentation; API documentation for each page is available from the user interface in the bottom right. Refer to those detailed pages for more information about the service.

Refer to Jenkins Remote Access API for information on the API and API wrappers.

Using API token

To configure this credential, you'll need:

- The **Jenkins Username**: For the user whom the token belongs to
- A **Personal API Token**: Generate this from the user's **profile details > Configure > Add new token**. Refer to these Stack Overflow instructions for more detail.
- The **Jenkins Instance URL**

Jenkins rebuilt their API token setup in 2018. If you're working with an older Jenkins instance, be sure you're using a non-legacy API token. Refer to Security Hardening: New API token system in Jenkins 2.129+ for more information.

Jina AI credentials

You can use these credentials to authenticate the following nodes:

- Jina AI

Supported authentication methods

- API key

Related resources

Refer to Jina AI's reader API documentation and Jina AI's search API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- **API key**: A Jina AI API key. You can get your free API key without creating an account by doing the following:
  1. Visit the Jina AI website.
  2. Select **API** on the page.
  3. Select **API KEY & BILLING** in the API app widget.
  4. Copy the key labeled "This is your unique key. Store it securely!".

Jina AI API keys start with 10 million free tokens that you can use non-commercially. To top up your key or use commercially, scroll on the **API KEY & BILLING** tab of the **API** widget and select the top up option that best fits your needs.

Jira credentials

You can use these credentials to authenticate the following nodes:

- Jira
- Jira Trigger

Prerequisites

Create a Jira Software Cloud or Server account.

Supported authentication methods

- SW Cloud API token: Use this method with Jira Software Cloud.
- SW Server account: Use this method with Jira Software Server.

Related resources

Refer to Jira's API documentation for more information about the service.

Using SW Cloud API token

To configure this credential, you'll need an account on Jira Software Cloud.

Then:

1. Log in to your Atlassian profile > **Security > API tokens** page, or jump straight there using this link.
2. Select **Create API Token**.
3. Enter a good **Label** for your token, like `n8n integration`.
4. Select **Create**.
5. Copy the API token.
6. In n8n, enter the **Email** address associated with your Jira account.
7. Paste the API token you copied as your **API Token**.
8. Enter the **Domain** you access Jira on, for example `https://example.atlassian.net`.

Refer to Manage API tokens for your Atlassian account for more information.

Using SW Server account

To configure this credential, you'll need an account on Jira Software Server.

Then:

1. Enter the **Email** address associated with your Jira account.
2. Enter your Jira account **Password**.
3. Enter the **Domain** you access Jira on.

JotForm credentials

You can use these credentials to authenticate the following nodes:

- JotForm Trigger

Supported authentication methods

- API key

Related resources

Refer to JotForm's API documentation for more information about the service.

Using API key

To configure this credential, you'll need a JotForm account and:

- An **API Key**
- The **API Domain**

To set it up:

1. Go to **Settings > API**.
2. Select **Create New Key**.
3. Select the **Name** in JotForm to update the API key name to something meaningful, like `n8n integration`.
4. Copy the **API Key** and enter it in your n8n credential.
5. In n8n, select the **API Domain** that applies to you based on the forms you're using:
   - **api.jotform.com**: Use this unless the other form types apply to you.
   - **eu-api.jotform.com**: Select this if you're using JotForm EU Safe Forms.
   - **hipaa-api.jotform.com**: Select this if you're using JotForm HIPAA forms.

Refer to the JotForm API documentation for more information on creating keys and API domains.

JWT credentials

You can use these credentials to authenticate the following nodes:

- JWT
- Webhook

Supported authentication methods

- Passphrase: Signed with a secret with HMAC algorithm
- Private key (PEM key): For use with Private Key JWT with RSA or

ECDSA algorithm

Related resources

Refer to the JSON Web Token spec for more details.

For a more verbose introduction, refer to the JWT website Introduction to JSON Web Tokens. Refer to JSON Web Token (JWT) Signing Algorithms Overview for more information on selecting between the two types and the algorithms involved.

Using Passphrase

To configure this credential:

1. Select the **Key Type** of **Passphrase**.
2. Enter the Passphrase **Secret**
3. Select the **Algorithm** used to sign the assertion. Refer to Available algorithms below for a list of supported algorithms.

Using private key (PEM key)

To configure this credential: 1. Select the **Key Type** of **PEM Key**. 2. A **Private Key**: Obtained from generating a Key Pair. Refer to Generate RSA Key Pair for an example. 3. A **Public Key**: Obtained from generating a Key Pair. Refer to Generate RSA Key Pair for an example. 4. Select the **Algorithm** used to sign the assertion. Refer to Available algorithms below for a list of supported algorithms.

Available algorithms

This n8n credential supports the following algorithms:

- `HS256`
- `HS384`
- `HS512`
- `RS256`
- `RS384`
- `RS512`
- `ES256`
- `ES384`
- `ES512`
- `PS256`
- `PS384`
- `PS512`
- `none`

Kafka credentials

You can use these credentials to authenticate the following nodes:

- Kafka

- Kafka Trigger

Supported authentication methods

- Client ID

Related resources

Refer to Kafka's documentation for more information about using the service.

If you're new to Kafka, refer to the Apache Kafka Quickstart for initial setup.

Refer to Encryption and Authentication using SSL for working with SSL in Kafka.

Using client ID

To configure this credential, you'll need a running Kafka environment and:

- A **Client ID**
- A list of relevant **Brokers**
- Username/password authentication details if your Kafka environment uses authentication

To set it up:

1. Enter the `CLIENT-ID` of the client or consumer group in the **Client ID** field in your credential.
2. Enter a comma-separated list of relevant **Brokers** for the credential to use in the format `<broker-service-name>:<port>`. Use the name you gave the broker when you defined it in the `services` list. For example, `kafka-1:9092,kafka-2:9092` would add the brokers `kafka-1` and `kafka-2` on port `9092`.
3. If your Kafka environment doesn't use SSL, turn off the **SSL** toggle.
4. If you've enabled authentication using SASL in your Kafka environment, turn on the **Authentication** toggle. Then add:
    1. The **Username**
    2. The **Password**
    3. Select the broker's configured **SASL Mechanism**. Refer to SASL configuration for more information. Options include:
        - `Plain`
        - `scram-sha-256`
        - `scram-sha-512`

Keap credentials

You can use these credentials to authenticate the following nodes:

- Keap
- Keap Trigger

Prerequisites

Create a Keap developer account.

Supported authentication methods

- OAuth2

Related resources

Refer to Keap's REST API documentation for more information about the service.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you need to configure OAuth2 from scratch or need more detail on what's happening in the OAuth web flow, refer to the instructions in the Getting Started with OAuth2 documentation.

Kibana credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

- Create an Elasticsearch account.
- If you're creating a new account to test with, load some sample data into Kibana. Refer to the Kibana quick start for more information.

Supported authentication methods

- Basic auth

Related resources

Refer to Kibana's API documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using basic auth

To configure this credential, you'll need:

- The **URL** you use to access Kibana, for example `http://localhost:5601`
- A **Username**: Use the same username that you use to log in to Elastic.
- A **Password**: Use the same password that you use to log in to Elastic.

Kitemaker credentials

You can use these credentials to authenticate the following nodes:

- Kitemaker

Prerequisites

Create a Kitemaker account.

Supported authentication methods

- API access token

Related resources

Refer to Kitemaker's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- A **Personal Access Token**: Generate a personal access token from **Manage > Developer settings**. Refer to API Authentication for more detailed instructions.

KoboToolbox credentials

You can use these credentials to authenticate the following nodes:

- KoboToolbox trigger
- KoboToolbox

Prerequisites

Create a KoboToolbox account.

Supported authentication methods

- API token

Related resources

Refer to KoboToolbox's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- An **API Root URL**: Enter the URL of the KoboToolbox server where you created your account. For the Global KoboToolbox Server, use `https://kf.kobotoolbox.org`. For the European Union KoboToolbox Server, use `https://eu.kobotoolbox.org`.
- An **API Token**: Displayed in your **Account Settings**. Refer to Getting your API token for more information.

LDAP credentials

You can use these credentials to authenticate the following nodes:

- LDAP

Prerequisites

Create a server directory using Lightweight Directory Access Protocol (LDAP).

Some common LDAP providers include:

- Jumpcloud
- Azure ADDS
- Okta

Supported authentication methods

- LDAP server details

Related resources

Refer to your LDAP provider's own documentation for detailed information.

For general LDAP information, refer to Basic LDAP concepts for a basic overview and The LDAP Bind Operation for information on how the bind operation and authentication work.

Using LDAP server details

To configure this credential, you'll need:

- The **LDAP Server Address**: Use the IP address or domain of your LDAP server.
- The **LDAP Server Port**: Use the number of the port used to connect to the LDAP server.
- The **Binding DN**: Use the Binding Distinguished Name (Bind DN) for your LDAP server. This is the user account the credential should log in as. If you're using Active Directory, this may look something like `cn=administrator, cn=Users, dc=n8n, dc=io`. Refer to your LDAP provider's documentation for more information on identifying this DN and the related password.
- The **Binding Password**: Use the password for the **Binding DN** user.
- Select the **Connection Security**: Options include:
  - `None`
  - `TLS`
  - `STARTTLS`
- *Optional:* Enter a numeric value in seconds to set a **Connection Timeout**.

Lemlist credentials

You can use these credentials to authenticate the following nodes:

- Lemlist
- Lemlist Trigger

Prerequisites

Create an account on a Lemlist instance.

Supported authentication methods

- API key

Related resources

Refer to Lemlist's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Access your API key in **Settings > Integrations**. Refer to the API Authentication documentation for more information.

Line credentials

You can use these credentials to authenticate the following nodes:

- Line

Supported authentication methods

- Notify OAuth2

Related resources

Refer to Line Notify's API documentation for more information about the service.

Using Notify OAuth2

To configure this credential, you'll need a Line account and:

- A **Client ID**
- A **Client Secret**

To generate both, connect Line with Line Notify. Then:

1. Open the Line Notify page to add a new service.
2. Enter a **Service name**. This name displays when someone tries to connect to the service.
3. Enter a **Service description**.
4. Enter a **Service URL**
5. Enter your **Company/Enterprise**.
6. Select your **Country/region**.
7. Enter your name or team name as the **Representative**.
8. Enter a valid **Email address**. Line will verify this email address before the service is fully registered. Use an email address you have ready access to.
9. Copy the **OAuth Redirect URL** from your n8n credential and enter it as the **Callback URL** in Line Notify.
10. Select **Agree and continue** to agree to the terms of service.
11. Verify the information you entered is correct and select **Add**.
12. Check your email and open the Line Notify Registration URL to verify your email address.
13. Once verification is complete, open **My services**.
14. Select the service you just added.
15. Copy the **Client ID** and enter it in your n8n credential.
16. Select the option to **Display** the **Client Secret**. Copy the **Client Secret** and enter it in your n8n credential.
17. In n8n, select **Connect my account** and follow the on-screen prompts to finish the credential.

Refer to the Authentication section of Line Notify's API documentation for more information.

Linear credentials

You can use these credentials to authenticate the following nodes:

- Linear Trigger
- Linear

Prerequisites

Create a Linear account.

Supported authentication methods

- API key
- OAuth2

Related resources

Refer to Linear's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- A personal **API Key**: Create a dedicated personal API key in your **Settings > Security & access**. Refer to the Linear Personal API keys documentation for more information.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated when you create a new OAuth2 application.
- A **Client Secret**: Generated when you create a new OAuth2 application.
- Select the **Actor**: The actor defines how the OAuth2 application should create issues, comments and other changes. Options include:
  - **User** (Linear's default): The application creates resources as the authorizing user. Use this option if you want each user to do their own authentication.
  - **Application**: The application creates resources as itself. Use this option if you have only one user (like an admin) authorizing the application.
- To use this credential with the Linear Trigger node, you must enable the **Include Admin Scope** toggle.

Refer to the Linear OAuth2 Authentication documentation for more detailed instructions and explanations. Use the n8n **OAuth Redirect URL** as the **Redirect callback URL** in your Linear OAuth2 application.

LingvaNex credentials

You can use these credentials to authenticate the following nodes:

- LingvaNex

Prerequisites

Create a LingvaNex account.

Supported authentication methods

- API key

Related resources

Refer to Lingvanex's Cloud API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Generate an API key from your **Account** page. Refer to Where can I get the authorization key? for more detailed instructions.

LinkedIn credentials

You can use these credentials to authenticate the following nodes:

- LinkedIn

Prerequisites

- Create a LinkedIn account.
- Create a LinkedIn Company Page.

Supported authentication methods

- **Community Management OAuth2**: Use this method if you're a new LinkedIn user or creating a new LinkedIn app.
- **OAuth2**: Use this method for older LinkedIn apps and user accounts.

Related Resources

Refer to LinkedIn's Community Management API documentation for more information about the service.

This credential works with API version `202404`.

Using Community Management OAuth2

Use this method if you're a new LinkedIn user or creating a new LinkedIn app.

To configure this credential, you'll need a LinkedIn account, a LinkedIn Company Page, and:

- A **Client ID**: Generated after you create a new developer app.
- A **Client Secret**: Generated after you create a new developer app.

To create a new developer app and set up the credential:

1. Log into LinkedIn and select this link to create a new developer app.
2. Enter an **App name** for your app, like `n8n integration`.
3. For the **LinkedIn Page**, enter a LinkedIn Company Page or use the **Create a new LinkedIn Page** link to create one on-the-fly. Refer to Associate an App with a LinkedIn Page for more information.
4. Add an **App logo**.
5. Check the box to agree to the **Legal agreement**.
6. Select **Create app**.
7. This should open the **Products** tab. Select the products/APIs you want to enable for your app. For the LinkedIn node to work properly, you must include and configure:
    - **Share on LinkedIn**
    - **Sign In with LinkedIn using OpenID Connect**
    - **Advertising API** (if using it as an organization account rather than an individual)
8. Once you've requested access to the products you need, open the **Auth** tab.
9. Copy the **Client ID** and enter it in your n8n credential.
10. Select the icon to **Copy** the **Primary Client Secret**. Enter this in your n8n credential as the **Client Secret**.

Refer to Getting Access to LinkedIn APIs for more information on scopes and permissions.

Using OAuth2

Only use this method for older LinkedIn apps and user accounts.

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

All users must select:

- **Organization Support**: If turned on, the credential requests permission to post as an organization using the `w_organization_social` scope.

    - To use this option, you must put your app through LinkedIn's Community Management App Review process.
- **Legacy**: If turned on, the credential uses legacy scopes for `r_liteprofile` and `r_emailaddress` instead of the newer `profile` and `email` scopes.

If you're self-hosting n8n, you'll need to configure OAuth2 from scratch by creating a new developer app:

1. Log into LinkedIn and select this link to create a new developer app.
2. Enter an **App name** for your app, like `n8n integration`.
3. For the **LinkedIn Page**, enter a LinkedIn Company Page or use the **Create a new LinkedIn Page** link to create one on-the-fly. Refer to Associate an App with a LinkedIn Page for more information.
4. Add an **App logo**.
5. Check the box to agree to the **Legal agreement**.
6. Select **Create app**.
7. This should open the **Products** tab. Select the products/APIs you want to enable for your app. For the LinkedIn node to work properly, you must include:
    - **Share on LinkedIn**
    - **Sign In with LinkedIn using OpenID Connect**
8. Once you've requested access to the products you need, open the **Auth** tab.
9. Copy the **Client ID** and enter it in your n8n credential.
10. Select the icon to **Copy** the **Primary Client Secret**. Enter this in your n8n credential as the **Client Secret**.

Refer to Getting Access to LinkedIn APIs for more information on scopes and permissions.

LoneScale credentials

You can use these credentials to authenticate the following nodes:

- LoneScale
- LoneScale Trigger

Prerequisites

Create a LoneScale account.

Supported authentication methods

- API key

Related resources

Refer to LoneScale's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Refer to LoneScale's Generate an API key documentation to generate your key.

Magento 2 credentials

You can use these credentials to authenticate the following node:

- Magento 2

Prerequisites

- Create a Magento (Adobe Commerce) account.
- Set your store to **Allow OAuth Access Tokens to be used as standalone Bearer tokens**.
    - Go to **Admin > Stores > Configuration > Services > OAuth > Consumer Settings**.
    - Set the **Allow OAuth Access Tokens to be used as standalone Bearer tokens** option to **Yes**.
    - You can also enable this setting from the CLI by running the following command:

      ```
      bin/magento config:set
      oauth/consumer/enable_integration_as_bearer 1
      ```

This step is necessary until n8n updates the Magento 2 credentials to use OAuth. Refer to Integration Tokens for more information.

Supported authentication methods

- API access token

Related resources

Refer to Magento's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- A **Host**: Enter the address of your Magento store.
- An **Access Token**: Get an access token from the **Admin Panel**:
    1. Go to **System > Extensions > Integrations**.
    2. Add a new Integration.
    3. Go to the **API** tab and select the Magento resources you'd like

the n8n integration to access.
4. From the **Integrations** page, **Activate** the new integration.
5. Select **Allow** to display your access token so you can copy it and enter it in n8n.

Mailcheck credentials

You can use these credentials to authenticate the following nodes:

- Mailcheck

Prerequisites

Create a Mailcheck account.

Supported authentication methods

- API key

Related resources

Refer to Mailcheck's API documentation for more information about the service.

Using API Key

To configure this credential, you'll need:

- An **API Key**: Generate an API Key in the API section of your dashboard. Refer to Mailcheck's How to create an API key documentation for detailed instructions.

Mailchimp credentials

You can use these credentials to authenticate the following nodes:

- Mailchimp
- Mailchimp Trigger

Prerequisites

Create a Mailchimp account.

Supported authentication methods

- API key
- OAuth2

Refer to Selecting an authentication method for guidance on which method to use.

Related resources

Refer to Mailchimp's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Generate an API key in the API keys section of your Mailchimp account. Refer to Mailchimp's Generate your API key documentation for more detailed instructions.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you need to configure OAuth2 from scratch, register an application. Refer to the Mailchimp OAuth2 documentation for more information.

Selecting an authentication method

Mailchimp suggests using an API key if you're only accessing your own Mailchimp account's data:

> Use an API key if you're writing code that tightly couples *your* application's data to *your* Mailchimp account's data. If you ever need to access *someone else's* Mailchimp account's data, you should be using OAuth 2 (source)

MailerLite credentials

You can use these credentials to authenticate the following nodes:

- MailerLite
- MailerLite Trigger

Prerequisites

Create a MailerLite account.

Supported authentication methods

- API key

Related resources

Refer to MailerLite's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Generate an API key from the **Integrations** menu. Refer to the API Authentication documentation for more detailed instructions.

Enable the **Classic API** toggle if the API key is for a MailerLite Classic account instead of the newer MailerLite experience.

Mailgun credentials

You can use these credentials to authenticate the following nodes:

- Mailgun

Prerequisites

- Create a Mailgun account.
- Add and verify a domain in Mailgun or use the provided sandbox domain for testing.

Supported authentication methods

- API key

Related resources

Refer to Mailgun's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Domain**: If your Mailgun account is based in Europe, select **api.eu.mailgun.net**; otherwise, select **api.mailgun.net**. Refer to Mailgun Base URLs for more information.
- An **Email Domain**: Enter the email sending domain you're working with. If you have multiple sending domains, refer to Working with multiple email domains for more information.
- An **API Key**: View your API key in **Settings > API Keys**. Refer to Mailgun's API Authentication documentation for more detailed instructions.

Working with multiple email domains

If your Mailgun account includes multiple sending domains, create a separate credential for each email domain you're working with.

Mailjet credentials

You can use these credentials to authenticate the following nodes:

- Mailjet
- Mailjet Trigger

Prerequisites

Create a Mailjet account.

Supported authentication methods

- Email API key: For use with Mailjet's Email API
- SMS token: For use with Mailjet's SMS API

Related resources

Refer to Mailjet's Email API documentation and Mailjet's SMS API documentation for more information about each service.

Using Email API key

To configure this credential, you'll need:

- An **API Key**: View and generate API keys in your Mailjet API Key Management page.
- A **Secret Key**: View your API Secret Keys in your Mailjet API Key Management page.
- *Optional:* Select whether to use **Sandbox Mode** for calls made using this credential. When turned on, all API calls use Sandbox mode: the API will still validate the payloads but won't deliver the actual messages. This can be useful to troubleshoot any payload error messages without actually sending messages. Refer to Mailjet's Sandbox Mode documentation for more information.

For this credential, you can use either:

- Mailjet's primary API key and secret key
- A subaccount API key and secret key

Refer to Mailjet's How to create a subaccount (or additional API key) documentation for detailed instructions on creating more API keys. Refer to What are subaccounts and how does it help me? page for more information on Mailjet subaccounts and when you might want to use one.

Using SMS Token

To configure this credential, you'll need:

- An access **Token**: Generate a new token from Mailjet's SMS Dashboard.

Malcore credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Malcore account.

Related resources

Refer to Malcore's API documentation for more information about authenticating with the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API key

To configure this credential, you'll need:

- An **API Key**: Get an API Key from your **Account > API**.

Refer to Using the Malcore API for more information.

Mandrill credentials

You can use these credentials to authenticate the following nodes:

- Mandrill

Prerequisites

- Create a Mailchimp Transactional email account
- Log in to Mandrill with your Mailchimp account.

If you already have a Mailchimp account with a Standard plan or higher, enable Transactional Emails within that account to use Mandrill.

Supported authentication methods

- API key

Related resources

Refer to Mailchimp's Transactional API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Generate an API key from the Mandrill Settings. Refer to Mailchimp's Generate your API key documentation for more detailed instructions.

Marketstack credentials

You can use these credentials to authenticate the following nodes:

- Marketstack

Prerequisites

Create a Marketstack account.

Supported authentication methods

- API key

Related resources

Refer to Marketstack's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: View and generate API keys in your Marketstack account dashboard.
- Select whether to **Use HTTPS**: Make this selection based on your Marketstack account plan level:
  - Free plan: Turn off **Use HTTPS**
  - All other plans: Turn on **Use HTTPS**

Matrix credentials

You can use these credentials to authenticate the following nodes:

- Matrix

Prerequisites

Create an account on a Matrix server. Refer to Creating an account for more information.

Supported authentication methods

- API access token

Related resources

Refer to the Matrix Specification for more information about the service.

Refer to the documentation for the specific client you're using to access the Matrix server.

Using API access token

To configure this credential, you'll need:

- An **Access Token**: This token is tied to the account you use to log into Matrix with.
- A **Homeserver URL**: This is the URL of the homeserver you entered when you created your account. n8n prepopulates this with matrix.org's own server; adjust this if you're using a server hosted elsewhere.

Instructions for getting these details vary depending on the client you're using to access the server. Both the **Access Token** and the **Homeserver URL** can most commonly be found in **Settings > Help & About > Advanced**, but refer to your client's documentation for more details.

Mattermost credentials

You can use these credentials to authenticate the following nodes:

- Mattermost

Supported authentication methods

- API access token

Related resources

Refer to Mattermost's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need a Mattermost account and:

- A personal **Access Token**
- Your Mattermost **Base URL**.

To set it up:

1. In Mattermost, go to **Profile > Security > Personal Access Tokens**.
2. Select **Create Token**.
3. Enter a **Token description**, like `n8n integration`.
4. Select **Save**.
5. Copy the **Token ID** and enter it as the **Access Token** in your n8n credential.
6. Enter your Mattermost URL as the **Base URL**.
7. By default, n8n connects only if SSL certificate validation succeeds. To connect even if SSL certificate validation fails, turn on **Ignore SSL Issues**.

Refer to the Mattermost Personal access tokens documentation for more information.

Enable personal access tokens

Not seeing the **Personal Access Tokens** option has two possible causes:

- Mattermost doesn't have the personal access tokens integration enabled.
- You're trying to generate a personal access token as a non-admin user who doesn't have permission to generate personal access tokens.

To identify the root cause and resolve it:

1. Log in to Mattermost as an admin.
2. Go to **System Console > Integrations > Integration Management**.
3. Confirm that **Enable personal access tokens** is set to **true**. If it's not, change.
4. Go to **System Console > User Management > Users**.
5. Search for the user account you want to allow to generate personal access tokens.
6. Select the **Actions** dropdown for the user and select **Manage roles**.
7. Check the box for **Allow this account to generate personal access tokens** and **Save**.

Refer to the Mattermost Personal access tokens documentation for more information.

Mautic credentials

You can use these credentials to authenticate the following nodes:

- Mautic
- Mautic Trigger

Supported authentication methods

- Basic auth
- OAuth2

Related resources

Refer to Mautic's API documentation for more information about the service.

Using basic auth

To configure this credential, you'll need an account on a Mautic instance and:

- Your **URL**
- A **Username**
- A **Password**

To set it up:

1. In Mautic, go to **Configuration > API Settings**.
2. If **Enable HTTP basic auth?** is set to **No**, change it to **Yes** and save. Refer to the API Settings documentation for more information.
3. In n8n, enter the Base **URL** of your Mautic instance.
4. Enter your Mautic **Username**.
5. Enter your Mautic **Password**.

Using OAuth2

To configure this credential, you'll need an account on a Mautic instance and:

- A **Client ID**: Generated when you create new API credentials.
- A **Client Secret**: Generated when you create new API credentials.
- Your **URL**

To set it up:

1. In Mautic, go to **Configuration > Settings**.
2. Select **API Credentials**.

3. Select the option to **Create new client**.
4. Select **OAuth 2** as the **Authorization Protocol**.
5. Enter a **Name** for your credential, like `n8n integration`.
6. In n8n, copy the **OAuth Callback URL** and enter it as the **Redirect URI** in Mautic.
7. Select **Apply**.
8. Copy the **Client ID** from Mautic and enter it in your n8n credential.
9. Copy the **Client Secret** from Mautic and enter it in your n8n credential.
10. Enter the Base **URL** of your Mautic instance.

Refer to What is Mautic's API? for more information.

Enable the API

To enable the API in your Mautic instance:

1. Go to **Settings > Configuration**.
2. Select **API Settings**.
3. Set **API enabled?** to **Yes**.
4. **Save** your changes.

Refer to How to use the Mautic API for more information.

Medium credentials

You can use these credentials to authenticate the following nodes:

- Medium

Prerequisites

- Create an account on Medium.
- For OAuth2, request access to credentials by emailing yourfriends@medium.com.

Supported authentication methods

- API access token
- OAuth2

Related resources

Refer to Medium's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- An API **Access Token**: Generate a token in **Settings > Security and apps > Integration tokens**. Use the integration token this generates as your n8n **Access Token**.

Refer to the Medium API Self-issued access tokens documentation for more information.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**
- A **Client Secret**

To generate a **Client ID** and **Client Secret**, you'll need access to the **Developers** menu. From there, create a new application to generate the Client ID and Secret.

Use these settings for your new application:

- Select **OAuth 2** as the **Authorization Protocol**
- Copy the **OAuth Callback URL** from n8n and use this as the **Callback URL** in Medium.

MessageBird credentials

You can use these credentials to authenticate the following nodes:

- MessageBird

Prerequisites

Create a Bird account.

Supported authentication methods

- API key

Related resources

Refer to MessageBird's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: To generate an appropriate key, visit the Access keys page in MessageBird. Refer to the API authorization documentation for detailed instructions.

Metabase credentials

You can use these credentials to authenticate the following nodes:

- Metabase node

Prerequisites

Create a Metabase account with access to a Metabase instance.

Supported authentication methods

- Basic auth

Related resources

Refer to Metabase's API documentation for more information about the service.

Using basic auth

To configure this credential, you'll need:

- A **URL**: Enter the base URL of your Metabase instance. If you're using a custom domain, use that URL.
- A **Username**: Enter your Metabase username.
- A **Password**: Enter your Metabase password.

Microsoft credentials

You can use these credentials to authenticate the following nodes:

- Microsoft Dynamics CRM
- Microsoft Excel
- Microsoft Graph Security
- Microsoft OneDrive
- Microsoft Outlook
- Microsoft SharePoint
- Microsoft Teams
- Microsoft Teams Trigger
- Microsoft To Do

Prerequisites

- Create a Microsoft Azure account.
- Create at least one user account with access to the appropriate service.
- If a corporate Microsoft Entra account manages the user account, the administrator account has enabled the option “User can consent to apps accessing company data on their behalf” for this user (see the Microsoft Entra documentation).

Supported authentication methods

- OAuth2

Related resources

Refer to the linked Microsoft API documentation below for more information about each service’s API:

- Dynamics CRM: Web API
- Excel: Graph API
- Graph Security: Graph API
- OneDrive: Graph API
- Outlook: Graph API and Outlook API
- Teams: Graph API
- To Do: Graph API

Using OAuth2

–8<– “_snippets/integrations/builtin/credentials/cloud-oauth-button.md”

Some Microsoft services require extra information for OAuth2. Refer to Service-specific settings for more guidance on those services.

For self-hosted users, there are two main steps to configure OAuth2 from scratch:

1. Register an application with the Microsoft Identity Platform.
2. Generate a client secret for that application.

Follow the detailed instructions for each step below. For more detail on the Microsoft OAuth2 web flow, refer to Microsoft authentication and authorization basics.

Register an application

Register an application with the Microsoft Identity Platform:

1. Open the Microsoft Application Registration Portal.
2. Select **Register an application**.
3. Enter a **Name** for your app.
4. In **Supported account types**, select **Accounts in any organizational directory (Any Azure AD directory - Multi-tenant) and personal Microsoft accounts (for example, Skype, Xbox)**.
5. In **Register an application**:
    1. Copy the **OAuth Callback URL** from your n8n credential.

    2. Paste it into the **Redirect URI (optional)** field.
    3. Select **Select a platform > Web**.
6. Select **Register** to finish creating your application.
7. Copy the **Application (client) ID** and paste it into n8n as the **Client ID**.

Refer to Register an application with the Microsoft Identity Platform for more information.

Generate a client secret

With your application created, generate a client secret for it:

1. On your Microsoft application page, select **Certificates & secrets** in the left navigation.
2. In **Client secrets**, select **+ New client secret**.
3. Enter a **Description** for your client secret, such as `n8n credential`.
4. Select **Add**.
5. Copy the **Secret** in the **Value** column.
6. Paste it into n8n as the **Client Secret**.
7. If you see other fields in the n8n credential, refer to Service-specific settings below for guidance on completing those fields.
8. Select **Connect my account** in n8n to finish setting up the connection.
9. Log in to your Microsoft account and allow the app to access your info.

Refer to Microsoft's Add credentials for more information on adding a client secret.

Service-specific settings

The following services require extra information for OAuth2:

Dynamics

Dynamics OAuth2 requires information about your Dynamics domain and region. Follow these extra steps to complete the credential:

1. Enter your Dynamics **Domain**.
2. Select the Dynamics data center **Region** you're within.

Refer to the Microsoft Datacenter regions documentation for more information on the region options and corresponding URLs.

Microsoft (general)

The general Microsoft OAuth2 also requires you to provide a space-separated list of **Scope**s for this credential.

Refer to Scopes and permissions in the Microsoft identity platform for a list of possible scopes.

Outlook

Outlook OAuth2 supports the credential accessing a user's primary email inbox or a shared inbox. By default, the credential will access a user's primary email inbox. To change this behavior:

1. Turn on **Use Shared Inbox**.
2. Enter the target user's UPN or ID as the **User Principal Name**.

SharePoint

SharePoint OAuth2 requires information about your SharePoint **Subdomain**.

To complete the credential, enter the **Subdomain** part of your SharePoint URL. For example, if your SharePoint URL is `https://tenant123.sharepoint.com`, the subdomain is `tenant123`.

SharePoint requires the following permissions:

Application permissions:

- `Sites.Read.All`
- `Sites.ReadWrite.All`

Delegated permissions:

- `SearchConfiguration.Read.All`
- `SearchConfiguration.ReadWrite.All`

Common issues

Here are the known common errors and issues with Microsoft OAuth2 credentials.

–8<– "_snippets/integrations/builtin/credentials/microsoft-need-admin-approval.md"

Microsoft Azure Monitor credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

- Create a Microsoft Azure account or subscription
- An app registered in Microsoft Entra ID

Supported authentication methods

- OAuth2

Related resources

Refer to Microsoft Azure Monitor's API documentation for more information about the service.

Using OAuth2

To configure this credential, you'll need a Microsoft Azure account and:

- A **Client ID**
- A **Client Secret**
- A **Tenant ID**
- The **Resource** you plan to access

Refer to Microsoft Azure Monitor's API documentation for more information about authenticating to the service.

Microsoft Entra ID credentials

You can use these credentials to authenticate the following nodes:

- Microsoft Entra ID

Prerequisites

- Create a Microsoft Entra ID account or subscription.
- If the user account is managed by a corporate Microsoft Entra account, the administrator account has enabled the option "User can consent to apps accessing company data on their behalf" for this user (see the Microsoft Entra documentation).

Microsoft includes an Entra ID free plan when you create a Microsoft Azure account.

Supported authentication methods

- OAuth2

Related resources

Refer to Microsoft Entra ID's documentation for more information about the service.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

For self-hosted users, there are two main steps to configure OAuth2 from scratch:

1. Register an application with the Microsoft Identity Platform.
2. Generate a client secret for that application.

Follow the detailed instructions for each step below. For more detail on the Microsoft OAuth2 web flow, refer to Microsoft authentication and authorization basics.

Register an application

Register an application with the Microsoft Identity Platform:

1. Open the Microsoft Application Registration Portal.
2. Select **Register an application**.
3. Enter a **Name** for your app.
4. In **Supported account types**, select **Accounts in any organizational directory (Any Azure AD directory - Multi-tenant) and personal Microsoft accounts (for example, Skype, Xbox)**.
5. In **Register an application**:
    1. Copy the **OAuth Callback URL** from your n8n credential.
    2. Paste it into the **Redirect URI (optional)** field.
    3. Select **Select a platform > Web**.
6. Select **Register** to finish creating your application.
7. Copy the **Application (client) ID** and paste it into n8n as the **Client ID**.

Refer to Register an application with the Microsoft Identity Platform for more information.

Generate a client secret

With your application created, generate a client secret for it:

1. On your Microsoft application page, select **Certificates & secrets** in the left navigation.
2. In **Client secrets**, select **+ New client secret**.
3. Enter a **Description** for your client secret, such as `n8n credential`.
4. Select **Add**.
5. Copy the **Secret** in the **Value** column.
6. Paste it into n8n as the **Client Secret**.
7. Select **Connect my account** in n8n to finish setting up the connection.
8. Log in to your Microsoft account and allow the app to access your info.

Refer to Microsoft's Add credentials for more information on adding a client secret.

Setting custom scopes

Microsoft Entra ID credentials use the following scopes by default:

- `openid`
- `offline_access`
- `AccessReview.ReadWrite.All`
- `Directory.ReadWrite.All`
- `NetworkAccessPolicy.ReadWrite.All`
- `DelegatedAdminRelationship.ReadWrite.All`
- `EntitlementManagement.ReadWrite.All`
- `User.ReadWrite.All`
- `Directory.AccessAsUser.All`

- `Sites.FullControl.All`
- `GroupMember.ReadWrite.All`

To select different scopes for your credentials, enable the **Custom Scopes** slider and edit the **Enabled Scopes** list. Keep in mind that some features may not work as expected with more restrictive scopes.

Common issues

Here are the known common errors and issues with Microsoft Entra credentials.

–8<– “_snippets/integrations/builtin/credentials/microsoft-need-admin-approval.md”

Microsoft SQL credentials

You can use these credentials to authenticate the following nodes:

- Microsoft SQL

Prerequisites

Create a user account on a Microsoft SQL server database.

Supported authentication methods

- SQL database connection

Related resources

Refer to Microsoft’s Connect to SQL Server documentation for more information about connecting to the service.

Using SQL database connection

To configure this credential, you’ll need:

- The **Server** name
- The **Database** name
- Your **User** account/ID
- Your **Password**
- The **Port** to use for the connection
- The **Domain** name
- Whether to use **TLS**
- Whether to **Ignore SSL Issues**
- The **Connect Timeout**
- The **Request Timeout**
- The **TDS Version** the connection should use

To set up the database connection:

1. Enter the SQL Server Host Name as the **Server**. In an existing SQL Server connection, the host name comes before the instance name in the format `HOSTNAME\INSTANCENAME`. Find the host name:
    - In the **Object Explorer** pane as the top-level object for your database.
    - In the footer of a query window.
    - Viewing the current connection **Properties** and looking for **Name** or **Display Name**.
    - Refer to Find SQL Server Instance Name | When you're connected to SQL Server for more information. You can also find the information in the Error logs.
2. Enter the SQL Server Instance Name as the **Database** name. Find this name using the same steps listed above for finding the host name.
    - If you don't see an instance name in any of these places, then your database uses the default `MSSQLSERVER` instance name.
3. Enter your **User** account name or ID.
4. Enter your **Password**.
5. For the **Port**:
    - SQL Server defaults to `1433`.
    - If you can't connect over port 1433, check the Error logs for the phrase `Server is listening on` to identify the port number you should enter.
6. You only need to enter the **Domain** name if users in multiple domains access your database. Run this SQL query to get the domain name:

    ```
    SELECT DEFAULT_DOMAIN()[DomainName];
    ```
7. Select whether to use **TLS**.
8. Select whether to **Ignore SSL Issues**: If turned on, the credential will connect even if SSL certificate validation fails.
9. Enter the number of milliseconds n8n should attempt the initial connection to complete before disconnecting as the **Connect Timeout**. Refer to the SqlConnection.ConnectionTimeout property documentation for more information.
    - SQL Server stores this timeout as seconds, while n8n stores it as milliseconds. If you're copying your SQL Server defaults, multiple by 100 before entering the number here.
10. Enter the number of milliseconds n8n should wait on a given request before timing out as the **Request Timeout**. This is basically a query timeout parameter. Refer to Troubleshoot query time-out errors for more information.
11. Select the Tabular Data Stream (TDS) protocol to use from the **TDS Version** dropdown. If the server doesn't support the version you select here, the connection uses a negotiated alternate version. Refer to Appendix A: Product Behavior for a more detailed breakdown of the TDS versions' compatibility with different SQL Server versions and .NET frameworks. Options include:

- **7_4 (SQL Server 2012 ~ 2019)**: TDS version 7.4.
- **7_3_B (SQL Server 2008R2)**: TDS version 7.3.B.
- **7_3_A (SQL Server 2008)**: TDS version 7.3.A.
- **7_2 (SQL Server 2005)**: TDS version 7.2.
- **7_1 (SQL Server 2000)**: TDS version 7.1.

Milvus credentials

You can use these credentials to authenticate the following nodes:

- Milvus Vector Store

Prerequisites

Create and run an Milvus instance. Refer to the Install Milvus for more information.

Supported authentication methods

- Basic auth

Related resources

Refer to Milvus's Authentication documentation for more information about setting up authentication.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using basic auth

To configure this credential, you'll need:

- **Base URL**: The base URL of your Milvus instance. The default is `http://localhost:19530`.
- **Username**: The username to authenticate to your Milvus instance. The default value is `root`.
- **Password**: The password to authenticate to your Milvus instance. The default value is `Milvus`.

Mindee credentials

You can use these credentials to authenticate the following nodes:

- Mindee

Prerequisites

Create a Mindee account.

Supported authentication methods

- Invoice API key: For use with the Invoice OCR API
- Receipt API key: For use with the Receipt OCR API

Related resources

Refer to Mindee's Invoice OCR API documentation and Mindee's Receipt OCR API documentation for more information about each service.

Using invoice API key

To configure this credential, you'll need:

- An **API Key**: Refer to the Mindee Create & Manage API Keys documentation for instructions on creating API keys.

Using receipt API key

To configure this credential, you'll need:

- An **API Key**: Refer to the Mindee Create & Manage API Keys documentation for instructions on creating API keys.

Miro credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Miro account.

Supported authentication methods

- OAuth2

Related resources

Refer to Miro's API documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using OAuth2

To configure this credential, you'll need a Miro account and app, as well as:

- A **Client ID**: Generated when you create a new OAuth2 application.
- A **Client Secret**: Generated when you create a new OAuth2 application.

Refer to Miro's API documentation for more information about authenticating to the service.

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you're self-hosting n8n, you'll need to create an app to configure OAuth2. Refer to Miro's OAuth documentation for more information about setting up OAuth2.

MISP credentials

You can use these credentials to authenticate the following nodes:

- MISP

Prerequisites

Install and run a MISP instance.

Supported authentication methods

- API key

Related resources

Refer to MISP's Automation API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: In MISP, these are called Automation keys. Get an automation key from **Event Actions > Automation**. Refer to MISP's automation keys documentation for instructions on generating more keys.
- A **Base URL**: Your MISP URL.
- Select whether to **Allow Unauthorized Certificates**: If turned on, the credential will connect even if SSL certificate validation fails.

Mist credentials

–8<– “_snippets/integrations/builtin/credentials/cred-only-statement.md”

Prerequisites

Create a Mist account and organization. Refer to Create a Mist account and Organization for detailed instructions.

Supported authentication methods

- API token

Related resources

Refer to Mist’s documentation for more information about the service. If you’re logged in to your Mist account, go to https://api.mist.com/api/v1/docs/Home to view the full API documentation.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n’s website.

Using API token

To configure this credential, you’ll need:

- An **API Token**: You can use either a User API token or an Org API token. Refer to How to generate a user API token for instructions on generating a User API token. Refer to Org API token for instructions on generating an Org API token.
- Select the **Region** you’re in. Options include:
  - **Europe**: Select this option if your cloud environment is in any of the EMEA regions.
  - **Global**: Select this option if your cloud environment is in any of the global regions.

Mistral Cloud credentials

You can use these credentials to authenticate the following nodes:

- Mistral AI
- Mistral Cloud
- Embeddings Mistral Cloud

Prerequisites

- Create a Mistral La Plateforme account.
- You must add payment information in **Workspace > Billing** and

activate payments to enable API keys. Refer to Account setup for more information.

Supported authentication methods

- API key

Related resources

Refer to Mistral's API documentation for more information about the APIs.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

To configure this credential, you'll need:

- An **API Key**

Once you've added payment information to your Mistral Cloud account:

1. Sign in to your Mistral account.
2. Go to the **API Keys** page.
3. Select **Create new key**.
4. Copy the API key and enter it in your n8n credential.

Refer to Account setup for more information.

/// note | Paid account required Mistral requires you to add payment information and activate payments to use API keys. Refer to the Prerequisites section above for more information.

Mocean credentials

You can use these credentials to authenticate the following nodes:

- Mocean

Prerequisites

Create a Mocean account.

Supported authentication methods

- API key

Related resources

Refer to Mocean's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**
- An **API Secret**

Both the key and secret are accessible in your Mocean Dashboard. Refer to API Authentication for more information.

---

monday.com credentials

You can use these credentials to authenticate the following nodes:

- monday.com

Supported authentication methods

- API token
- OAuth2

Related resources

Refer to monday.com's API documentation for more information about authenticating with the service.

Using API token

To configure this credential, you'll need a monday.com account and:

- An API **Token V2**

To get your token:

1. In your monday.com account, select your profile picture in the top right corner.
2. Select **Developers**. The Developer Center opens in a new tab.
3. In the Developer Center, select **My Access Tokens > Show**.
4. Copy your personal token and enter it in your n8n credential as the **Token V2**.

Refer to monday.com API Authentication for more information.

Using OAuth2

To configure this credential, you'll need a monday.com account and:

- A **Client ID**
- A **Client Secret**

To generate both these fields, register a new monday.com application:

1. In your monday.com account, select your profile picture in the top right corner.
2. Select **Developers**. The Developer Center opens in a new tab.
3. In the Developer Center, select **Build app**. The app details open.
4. Enter a **Name** for your app, like `n8n integration`.
5. Copy the **Client ID** and enter it in your n8n credential.
6. **Show** the **Client Secret**, copy it, and enter it in your n8n credential.
7. In the left menu, select **OAuth**.
8. For **Scopes**, select `boards:write` and `boards:read`.
9. Select **Save Scopes**.
10. Select the **Redirect URLs** tab.
11. Copy the **OAuth Redirect URL** from n8n and enter it as the **Redirect URL**.
12. **Save** your changes in monday.com.
13. In n8n, select **Connect my account** to finish the setup.

Refer to Create an app for more information on creating apps.

Refer to OAuth and permissions for more information on the available scopes and setting up the Redirect URL.

MongoDB credentials

You can use these credentials to authenticate the following nodes:

- MongoDB
- MongoDB Atlas Vector Store
- MongoDB Chat Memory

Prerequisites

- Create a user account with the appropriate permissions on a MongoDB server.
- As a Project Owner, add all the n8n IP addresses to the IP Access List Entries in the project's **Network Access**. Refer to Add IP Access List entries for detailed instructions.

If you are setting up MongoDB from scratch, create a cluster and a database. Refer to the MongoDB Atlas documentation for more detailed instructions on these steps.

Supported authentication methods

- Database connection - Connection string
- Database connection - Values

Related resources

Refer to the MongoDBs Atlas documentation for more information about the service.

Using database connection - Connection string

To configure this credential, you'll need the Prerequisites listed above. Then:

1. Select **Connection String** as the **Configuration Type**.
2. Enter your MongoDB **Connection String**. To get your connection string in MongoDB, go to **Database > Connect**.
    1. Select **Drivers**.
    2. Copy the code you see in **Add your connection string into your application code**. It will be something like: `mongodb+srv://yourName:yourPassword@clusterName.mongodb.net/?retryWrites=true&w=majority`.
    3. Replace the `<password>` and `<username>` in the connection string with the database user's credentials you'll be using.
    4. Enter that connection string into n8n.
    5. Refer to Connection String for information on finding and formatting your connection string.
3. Enter your **Database** name. This is the name of the database that the user whose details you added to the connection string is logging into.
4. Select whether to **Use TLS**: Turn on to use TLS. You must have your MongoDB database configured to use TLS and have an x.509 certificate generated. Add information for these certificate fields in n8n:
    - **CA Certificate**
    - **Public Client Certificate**
    - **Private Client Key**
    - **Passphrase**

Refer to MongoDB's x.509 documentation for more information on working with x.509 certificates.

Using database connection - Values

To configure this credential, you'll need the Prerequisites listed above. Then:

1. Select **Values** as the **Configuration Type**.
2. Enter the database **Host** name or address.
3. Enter the **Database** name.
4. Enter the **User** you'd like to log in as.
5. Enter the user's **Password**.
6. Enter the **Port** to connect over. This is the port number your server uses to listen for incoming connections.
7. Select whether to **Use TLS**: Turn on to use TLS. You must have your MongoDB database configured to use TLS and have an x.509 certificate generated. Add information for these certificate fields in n8n:
    - **CA Certificate**
    - **Public Client Certificate**
    - **Private Client Key**
    - **Passphrase**

Refer to MongoDB's x.509 documentation for more information on working with x.509 certificates.

Monica CRM credentials

You can use these credentials to authenticate the following nodes:

- Monica CRM

Prerequisites

Sign up for a Monica CRM account or self-host an instance.

Supported authentication methods

- API token

Related resources

Refer to Monica's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- Your **Environment**:
  - Select **Cloud-Hosted** if you access your Monica instance through Monica.
  - Select **Self-Hosted** if you have self-hosted Monica on your own server. Provide your **Self-Hosted Domain**.
- An **API Token**: Generate a token in **Settings > API**.

Motorhead credentials

You can use these credentials to authenticate the following nodes:

- Motorhead

Supported authentication methods

- API key

Related resources

Refer to Motorhead's API documentation for more information about the service.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

To configure this credential, you'll need a Motorhead account and:

- Your **Host** URL
- An **API Key**
- A **Client ID**

To set it up, you'll generate an API key:

1. If you're self-hosting Motorhead, update the **Host** URL to match your Motorhead URL.
2. In Motorhead, go to **Settings > Organization**.
3. In the **API Keys** section, select **Create**.
4. Enter a **Name** for your API Key, like `n8n integration`.
5. Select **Generate**.
6. Copy the **apiKey** and enter it in your n8n credential.
7. Return to the API key list.
8. Copy the **clientID** for the key and enter it as the **Client ID** in your n8n credential.

Refer to Generate an API key for more information.

MQTT credentials

You can use these credentials to authenticate the following nodes:

- MQTT
- MQTT Trigger

Prerequisites

Install an MQTT broker.

MQTT provides a list of Servers/Brokers at MQTT Software.

Supported authentication methods

- Broker connection

Related resources

Refer to MQTT's documentation for more information about the MQTT protocol.

Refer to your broker provider's documentation for more detailed configuration and details.

Using broker connection

To configure this credential, you'll need:

- Your MQTT broker's **Protocol**
- The **Host**
- The **Port**
- A **Username** and **Password** to authenticate with
- If you're using **SSL**, the relevant certificates and keys

To set things up:

1. Select the broker's **Protocol**, which determines the URL n8n uses. Options include:
   - **Mqtt**: Begin the URL with the standard `mqtt:` protocol.
   - **Mqtts**: Begin the URL with the secure `mqtts:` protocol.
   - **Ws**: Begin the URL with the WebSocket `ws:` protocol.
2. Enter your broker **Host**.
3. Enter the **Port** number n8n should use to connect to the broker host.
4. Enter the **Username** to log into the broker as.
5. Enter that user's **Password**.
6. If you want to receive QoS 1 and 2 messages while offline, turn off the **Clean Session** toggle.
7. Enter a **Client ID** you'd like the credential to use. If you leave this blank, n8n will generate one for you. You can use a fixed or expression-based Client ID.
   - Client IDs can be useful to identify and track connection access. n8n recommends using something with `n8n` in it for easier auditing.
8. If your MQTT broker uses SSL, turn the **SSL** toggle on. Once you turn it on:
   1. Select whether to use **Passwordless** connection with certificates, which is like the SASL mechanism EXTERNAL. If turned on:
      1. Select whether to **Reject Unauthorized Certificate**: If turned off, n8n will connect even if the certificate validation fails.
      2. Add an SSL **Client Certificate**.
      3. Add an SSL **Client Key** for the Client Certificate.
   2. One or more SSL **CA Certificates**.

Refer to your MQTT broker provider's documentation for more detailed configuration instructions.

MSG91 credentials

You can use these credentials to authenticate the following nodes:

- MSG91

Prerequisites

Create a MSG91 account.

Supported authentication methods

- API key

Related resources

Refer to MSG91's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **Authentication Key**: To get your Authentication Key, go to the user menu and select **Authkey**. Refer to MSG91's Where can I find my authentication key? documentation for more information.

IP Security

MSG91 enables IP Security by default for authkeys.

For the n8n credentials to function with this setting enabled, add all the n8n IP addresses as whitelisted IPs in MSG91. You can add them in one of two places, depending on your desired security level:

- To allow any/all authkeys in the account to work with n8n, add the n8n IP addresses in the **Company's whitelisted IPs** section of the **Authkey** page.
- To allow only specific authkeys to work with n8n, add the n8n IP addresses in the **Whitelisted IPs** section of an authkey's details.

MySQL credentials

You can use these credentials to authenticate the following nodes:

- MySQL
- Agent

Prerequisites

Create a user account on a MySQL server database.

Supported authentication methods

- Database connection

Related resources

Refer to MySQL's documentation for more information about the service.

Using database connection

To configure this credential, you'll need:

- The server **Host**: The database's host name or IP address.
- The **Database** name.
- A **User** name.
- A **Password** for that user.
- The **Port** number used by the MySQL server.
- **Connect Timeout**: The number of milliseconds during the initial database connection before a timeout occurs.
- **SSL**: If your database is using SSL, turn this on and add details for the SSL certificate.
- **SSH Tunnel**: Choose whether to connect over an SSH tunnel. An SSH tunnel lets un-encrypted traffic pass over an encrypted connection and enables authorized remote access to servers protected from outside connections by a firewall.

To set up your database connection credential:

1. Enter your database's hostname as the **Host** in your n8n credential. Run this query to confirm the hostname:

   ```
   SHOW VARIABLES WHERE Variable_name = 'hostname';
   ```

2. Enter your database's name as the **Database** in your n8n credential. Run this query to confirm the database name:

   ```
   SHOW DATABASES;
   ```

3. Enter the username of a **User** in the database. This user should have appropriate permissions for whatever actions you want n8n to perform.

4. Enter the **Password** for that user.

5. Enter the **Port** number used by the MySQL server (default is `3306`). Run this query to confirm the port number:

   ```
   SHOW VARIABLES WHERE Variable_name = 'port';
   ```

6. Enter the **Connect Timeout** you'd like the node to use. The Connect Timeout is the number of milliseconds during the initial database connection the node should wait before timing out. n8n defaults to `10000` which is the default used by MySQL of 10 seconds. If you want to match your database's `connect_timeout`, run this query to get it, then multiply by 1000 before entering it in n8n:

   ```
   SHOW VARIABLES WHERE Variable_name = 'connect_timeout';
   ```

7. If your database uses SSL and you'd like to use **SSL** for the connection, turn this option on in the credential. If you turn it on, enter the information from your MySQL SSL certificate in these fields:

   1. Enter the `ca.pem` file contents in the **CA Certificate** field.
   2. Enter the `client-key.pem` file contents in the **Client Private Key** field.
   3. Enter the `client-cert.pem` file contents in the **Client Certificate** field.

8. If you want to use **SSH Tunnel** for the connection, turn this option on in the credential. Otherwise, skip it. If you turn it on:

1. Select the **SSH Authenticate with** to set the SSH Tunnel type to build:
   - Select **Password** if you want to connect to SSH using a password.
   - Select **Private Key** if you want to connect to SSH using an identity file (private key) and a passphrase.
2. Enter the **SSH Host**. n8n uses this host to create the SSH URI formatted as: `[user@]host:port`.
3. Enter the **SSH Port**. n8n uses this port to create the SSH URI formatted as: `[user@]host:port`.
4. Enter the **SSH User** to connect with. n8n uses this user to create the SSH URI formatted as: `[user@]host:port`.
5. If you selected **Password** for **SSH Authenticate with**, add the **SSH Password**.
6. If you selected **Private Key** for **SSH Authenticate with**:
   1. Add the contents of the **Private Key** or identity file used for SSH. This is the same as using the `ssh-identity-file` option with the `shell-connect()` command in MySQL.
   2. If the **Private Key** was created with a passphrase, enter that **Passphrase**. This is the same as using the `ssh-identity-pass` option with the `shell-connect()` command in MySQL. If the **Private Key** has no passphrase, leave this field blank.

Refer to MySQL | Creating SSL and RSA Certificates and Keys for more information on working with SSL certificates in MySQL. Refer to MySQL | Using an SSH Tunnel for more information on working with SSH tunnels in MySQL.

NASA credentials

You can use these credentials to authenticate the following nodes:

- NASA

Supported authentication methods

- API key

Related resources

Refer to the **Browse APIs** section of the NASA Open APIs for more information about the service.

Using an API key

To configure this credential, you'll need:

- An **API Key**

To generate an API key:

1. Go to the NASA Open APIs page.

2. Complete the fields in the **Generate API Key** section.
3. Copy the **API Key** and enter it in your n8n credential.

Netlify credentials

You can use these credentials to authenticate the following nodes:

- Netlify
- Netlify Trigger

Prerequisites

Create a Netlify account.

Supported authentication methods

- API access token

Related resources

Refer to Netlify's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- An **Access Token**: Generate an Access Token in **Applications > Personal Access Tokens**. Refer to Netlify API Authentication for more detailed instructions.

Netscaler ADC credentials

You can use these credentials to authenticate the following nodes:

- Netscaler ADC node

Prerequisites

Install a NetScaler/Citrix ADC appliance.

Supported authentication methods

- Basic auth

Related resources

Refer to Netscaler ADC's 14.1 NITRO API documentation for more information about the service.

Using basic auth

To configure this credential, you'll need:

- A **URL**: Enter the URL of your NetScaler/Citrix ADC instance.
- A **Username**: Enter your NetScaler/Citrix ADC username.
- A **Password**: Enter your NetScaler/Citrix ADC password.

Refer to Performing Basic Netscaler ADC Operations for more information.

Nextcloud credentials

You can use these credentials to authenticate the following nodes:

- Nextcloud

Supported authentication methods

- Basic auth
- OAuth2

Related resources

Refer to Nextcloud's API documentation for more information about the service.

Refer to Nextcloud's user manual for more information on installing and configuring Nextcloud.

Using basic auth

To configure this credential, you'll need a Nextcloud account and:

- Your **WebDAV URL**
- Your **User** name
- Your **Password** or an app password

To set it up:

1. To create your **WebDAV URL**: If Nextcloud is in the root of your domain: Enter the URL you use to access Nextcloud and add `/remote.php/webdav/`. For example, if you access Nextcloud at `https://cloud.n8n.com`, your WebDAV URL is `https://cloud.n8n.com/remote.php/webdav`.
   - If you have Nextcloud installed in a subdirectory, enter the URL you use to access Nextcloud and add `/<subdirectory>/remote.php/webdav/`. Replace `<subdirectory>` with the subdirectory Nextcloud's installed in.
   - Refer to Nextcloud's Third-party WebDAV clients

documentation for more information on constructing your WebDAV URL.
2. Enter your **User** name.
3. For the **Password**, Nextcloud recommends using an app password rather than your user password. To create an app password:
    1. In the Nextcloud Web interface, select your avatar in the top right and select **Personal settings**.
    2. In the left menu, choose **Security**.
    3. Scroll to the bottom to the **App Password** section and create a new app password.
    4. Copy that app password and enter it in n8n as your **Password**.

Using OAuth2

To configure this credential, you'll need a Nextcloud account and:

- An **Authorization URL** and **Access Token URL**: These depend on the URL you use to access Nextcloud.
- A **Client ID**: Generated once you add an OAuth2 client application in **Administrator Security Settings**.
- A **Client Secret**: Generated once you add an OAuth2 client application in **Administrator Security Settings**.
- A **WebDAV URL**: This depends on the URL you use to access Nextcloud.

To set it up:

1. In Nextcloud, open your **Administrator Security Settings**.
2. Find the **Add client** section under **OAuth 2.0 clients**.
3. Enter a **Name** for your client, like `n8n integration`.
4. Copy the **OAuth Callback URL** from n8n and enter it as the **Redirection URI**.
5. Then select **Add** in Nextcloud.
6. In n8n, update the **Authorization URL** to replace `https://nextcloud.example.com` with the URL you use to access Nextcloud. For example, if you access Nextcloud at `https://cloud.n8n.com`, the Authorization URL is `https://cloud.n8n.com/apps/oauth2/authorize`.
7. In n8n, update the **Access Token URL** to replace `https://nextcloud.example.com` with the URL you use to access Nextcloud. For example, if you access Nextcloud at `https://cloud.n8n.com`, the Access Token URL is `https://cloud.n8n.com/apps/oauth2/api/v1/token`.
8. Copy the Nextcloud **Client Identifier** for your OAuth2 client and enter it as the **Client ID** in n8n.
9. Copy the Nextcloud **Secret** and enter it as the **Client Secret** in n8n.
10. In n8n, to create your **WebDAV URL**: If Nextcloud is in the root of your domain, enter the URL you use to access Nextcloud and add `/remote.php/webdav/`. For example, if you access Nextcloud at

`https://cloud.n8n.com`, your WebDAV URL is `https://cloud.n8n.com/remote.php/webdav`.

- If you have Nextcloud installed in a subdirectory, enter the URL you use to access Nextcloud and add `/<subdirectory>/remote.php/webdav/`. Replace `<subdirectory>` with the subdirectory Nextcloud's installed in.
- Refer to Nextcloud's Third-party WebDAV clients documentation for more information on constructing your WebDAV URL.

Refer to the Nextcloud OAuth2 Configuration documentation for more detailed instructions.

NocoDB credentials

You can use these credentials to authenticate the following nodes:

- NocoDB

Supported authentication methods

- API token (recommended)
- User auth token

Related resources

Refer to NocoDB's API documentation for more information about the service.

Using API token

To configure this credential, you'll need a NocoDB instance and:

- An **API Token**
- Your database **Host**

To generate an API token:

1. Log into NocoDB and select the **User menu** in the bottom left sidebar.
2. Select **Account Settings**.
3. Open the **Tokens** tab.
4. Select **Add new API token**.
5. Enter a **Name** for your token, like `n8n integration`.
6. Select **Save**.
7. Copy the **API Token** and enter it in your n8n credential.
8. Enter the **Host** of your NocoDB instance in your n8n credential, for example `http://localhost:8080`.

Refer to the NocoDB API Tokens documentation for more detailed instructions.

Using user auth token

Before NocoDB deprecated it, user auth token was a temporary token designed for quick experiments with the API, valid for a session until the user logs out or for 10 hours.

To configure this credential, you'll need a NocoDB instance and:

- A **User Token**
- Your database **Host**

To generate a user auth token:

1. Log into NocoDB and select the **User menu** in the bottom left sidebar.
2. Select **Copy Auth token**.
3. Enter that auth token as the **User Token** in n8n.
4. Enter the **Host** of your NocoDB instance, for example `http://localhost:8080`.

Refer to the NocoDB Auth Tokens documentation for more information.

Notion credentials

You can use these credentials to authenticate the following nodes:

- Notion
- Notion Trigger

Prerequisites

Create a Notion account with admin level access.

Supported authentication methods

- API integration token: Used for internal integrations.
- OAuth2: Used for public integrations.

Related resources

Refer to Notion's API documentation for more information about the service.

Using API integration token

To configure this credential, you'll need:

- An **Internal Integration Secret**: Generated once you create a Notion integration.

To generate an integration secret, create a Notion integration and grab the integration secret from the **Secrets** tab:

1. Go to your Notion integration dashboard.
2. Select the **+ New integration** button.
3. Enter a **Name** for your integration, for example `n8n integration`. If desired, add a **Logo**.
4. Select **Submit** to create your integration.
5. Open the **Capabilities** tab. Select these capabilities:
   - `Read content`
   - `Update content`
   - `Insert content`
   - `User information without email addresses`
6. Be sure to **Save changes**.
7. Select the **Secrets** tab.
8. Copy the **Internal Integration Token** and add it as your n8n **Internal Integration Secret**.

Refer to the Internal integration auth flow setup documentation for more information about authenticating to the service.

Share Notion page(s) with the integration

For your integration to interact with Notion, you must give your integration page permission to interact with page(s) in your Notion workspace:

1. Visit the page in your Notion workspace.
2. Select the triple dot menu at the top right of a page.
3. In **Connections**, select **Connect to**.
4. Use the search bar to find and select your integration from the dropdown list.

Once you share at least one page with the integration, you can start making API requests. If the page isn't shared, any API requests made will respond with an error.

Refer to Integration permissions for more information.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated once you configure a public integration.
- A **Client Secret**: Generated once you configure a public integration.

You must create a Notion integration and set it to public distribution:

1. Go to your Notion integration dashboard.
2. Select the **+ New integration** button.
3. Enter a **Name** for your integration, for example `n8n integration`. If desired, add a **Logo**.
4. Select **Submit** to create your integration.
5. Open the **Capabilities** tab. Select these capabilities:
   - `Read content`
   - `Update content`
   - `Insert content`
   - `User information without email addresses`

6. Select **Save changes**.
7. Go to the **Distribution** tab.
8. Turn on the **Do you want to make this integration public?** control.
9. Enter your company name and website in the **Organization Information** section.
10. Copy the n8n **OAuth Redirect URL** and add it to as a **Redirect URI** in the Notion integration's **OAuth Domain & URLs** section.
11. Go to the **Secrets** tab.
12. Copy the **Client ID** and **Client Secret** and add them to your n8n credential.

Refer to Notion's public integration auth flow setup for more information about authenticating to the service.

Internal vs. public integrations

Internal integrations are:

- Specific to a single workspace.
- Accessible only to members of that workspace.
- Ideal for custom workspace enhancements.

Internal integrations use a simpler authentication process (the integration secret) and don't require any security review before publishing.

Public integrations are:

- Usable across multiple, unrelated Notion workspaces.
- Accessible by any Notion user, regardless of their workspace.
- Ideal for catering to broad use cases.

Public integrations use the OAuth 2.0 protocol for authentication. They require a Notion security review before publishing.

For a more detailed breakdown of the two integration types, refer to Notion's Internal vs. Public Integrations documentation.

---

npm credentials

You can use these credentials to authenticate the following nodes:

- npm

Prerequisites

Create an npm account.

Supported authentication methods

- API access token

Related resources

Refer to npm's external integrations documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- An **Access Token**: Create an access token by selecting **Access Tokens** from your profile menu. Refer to npm's Creating and viewing access tokens documentation for more detailed instructions.
- A **Registry URL**: If you're using a custom npm registry, update the **Registry URL** to that custom registry. Otherwise, keep the public registry value.

Odoo credentials

You can use these credentials to authenticate the following nodes:

- Odoo

Supported authentication methods

- API key (Recommended)
- Password

Related resources

Refer to Odoo's External API documentation for more information about the service.

Refer to the Odoo Getting Started tutorial if you're new to Odoo.

Using API key

To configure this credential, you'll need a user account on an Odoo database and:

- Your **Site URL**
- Your **Username**
- An **API key**
- Your **Database name**

To set up the credential with an API key:

1. Enter your Odoo server or site URL as the **Site URL**.
2. Enter your **Username** as it's displayed on your **Change password** screen in Odoo.
3. To use an API key, go to **Your Profile > Preferences > Account Security > Developer API Keys**.

    - If you don’t have this option, you may need to upgrade your Odoo plan. Refer to Required plan type for more information.
4. Select **New API Key**.
5. Enter a **Description** for the key, like `n8n integration`.
6. Select **Generate Key**.
7. Copy the key and enter it as the **Password or API key** in your n8n credential.
8. Enter your Odoo **Database name**, also known as the instance name.

Refer to Odoo API Keys for more information.

Using password

To configure this credential, you’ll need a user account on an Odoo database and:

- Your **Site URL**
- Your **Username**
- Your **Password**
- Your **Database name**

To set up the credential with a password:

1. Enter your Odoo server or site URL as the **Site URL**.
2. Enter your **Username** as it’s displayed on your **Change password** screen in Odoo.
3. To use a password, enter your user password in the **Password or API key** field.
4. Enter your Odoo **Database name**, also known as the instance name.

Required plan type

Okta credentials

You can use these credentials to authenticate the following nodes:

- Okta

Prerequisites

Create an Okta free trial or create an admin account on an existing Okta org.

Supported authentication methods

- SSWS API Access token

Related resources

Refer to Okta's documentation for more information about the service.

Using SSWS API access token

To configure this credential, you'll need:

- The **URL**: The base URL of your Okta org, also referred to as your unique subdomain. There are two quick ways to access it:
  1. In the Admin Console, select your **Profile**, hover over the domain listed below your username, and select the **Copy** icon. Paste this into n8n, but be sure to add `https://` before it.
  2. Copy the base URL of your Admin Console URL, for example `https://dev-123456-admin.okta.com`. Paste it into n8n and remove `-admin`, for example: `https://dev-123456.okta.com`.
- An **SSWS Access Token**: Create a token by going to **Security > API > Tokens > Create token**. Refer to Create Okta API tokens for more information.

Ollama credentials

You can use these credentials to authenticate the following nodes:

- Ollama
- Chat Ollama
- Embeddings Ollama

Prerequisites

Create and run an Ollama instance with one user. Refer to the Ollama Quick Start for more information.

Supported authentication methods

- Instance URL

Related resources

Refer to Ollama's API documentation for more information about the service.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using instance URL

To configure this credential, you'll need:

- The **Base URL** of your Ollama instance or remote authenticated Ollama instances.
- (Optional) The **API Key** for Bearer token authentication if connecting to a remote, authenticated proxy.

The default **Base URL** is `http://localhost:11434`, but if you've set the `OLLAMA_HOST` environment variable, enter that value. If you have issues connecting to a local n8n server, try `127.0.0.1` instead of `localhost`.

If you're connecting to Ollama through authenticated proxy services (such as Open WebUI) you must include an API key. If you don't need authentication, leave this field empty. When provided, the API key is sent as a Bearer token in the `Authorization` header of the request to the Ollama API.

Refer to How do I configure Ollama server? for more information.

Ollama and self-hosted n8n

If you're self-hosting n8n on the same machine as Ollama, you may run into issues if they're running in different containers.

For this setup, open a specific port for n8n to communicate with Ollama by setting the `OLLAMA_ORIGINS` variable or adjusting `OLLAMA_HOST` to an address the other container can access.

Refer to Ollama's How can I allow additional web origins to access Ollama? for more information.

One Simple API credentials

You can use these credentials to authenticate the following nodes:

- One Simple API

Prerequisites

Create a One Simple API account.

Supported authentication methods

- API token

Related resources

Refer to One Simple API's documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- An **API token**: Create a new API token on the API Tokens page. Be sure you select appropriate permissions for the token.

You can also access the API Tokens page by selecting your **Profile > API Tokens**.

Onfleet credentials

You can use these credentials to authenticate the following nodes:

- Onfleet
- Onfleet Trigger

Prerequisites

Create an Onfleet administrator account.

Supported authentication methods

- API key

Related resources

Refer to Onfleet's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API key**: To create an API key, log into your organization's administrator account. Select **Settings > API & Webhooks**, then select **+** to create a new key. Refer to Onfleet's Creating an API key documentation for more information.

OpenAI credentials

You can use these credentials to authenticate the following nodes:

- OpenAI
- Chat OpenAI
- Embeddings OpenAI
- LM OpenAI

Prerequisites

Create an OpenAI account.

Supported authentication methods

- API key

Related resources

Refer to OpenAI's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**
- An **Organization ID**: Required if you belong to multiple organizations; otherwise, leave this blank.

To generate your API Key:

1. Login to your OpenAI account or create an account.
2. Open your API keys page.
3. Select **Create new secret key** to create an API key, optionally naming the key.
4. Copy your key and add it as the **API Key** in n8n.

Refer to the API Quickstart Account Setup documentation for more information.

To find your Organization ID:

1. Go to your Organization Settings page.
2. Copy your Organization ID and add it as the **Organization ID** in n8n.

Refer to Setting up your organization for more information. Note that API requests made using an Organization ID will count toward the organization's subscription quota.

OpenCTI credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create an OpenCTI developer account.

Authentication methods

- API key

Related resources

Refer to OpenCTI's documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API key

To configure this credential, you'll need:

- An **API Key**: To get your API key, go to your **Profile > API access**. Refer to the OpenCTI Integrations Authentication documentation for more information.

OpenRouter credentials

You can use these credentials to authenticate the following nodes:

- Chat OpenRouter

Prerequisites

Create a OpenRouter account.

Supported authentication methods

- API key

Related resources

Refer to OpenRouter's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**

To generate your API Key:

1. Login to your OpenRouter account or create an account.
2. Open your API keys page.
3. Select **Create new secret key** to create an API key, optionally naming the key.
4. Copy your key and add it as the **API Key** in n8n.

Refer to the OpenRouter Quick Start page for more information.

OpenWeatherMap credentials

You can use these credentials to authenticate the following nodes:

- OpenWeatherMap

Supported authentication methods

- API access token

Related resources

Refer to OpenWeatherMap's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need an OpenWeatherMap account and:

- An **Access Token**

To get your **Access Token**:

1. After you verify your email address, OpenWeatherMap includes an **API Key** in your welcome email.
2. Copy that key and enter it in your n8n credential.

If you'd prefer to create a new key:

1. To create a new key, go to **Account > API Keys**.
2. In the **Create Key** section, enter an **API Key Name**, like `n8n integration`.
3. Select **Generate** to generate your key.
4. Copy the generated key and enter it in your n8n credential.

Oracle Database credentials

You can use these credentials to authenticate the following nodes:

- OracleDB
- Agent

Prerequisites

Create a user account on an Oracle Database server.

Supported authentication methods

- Database connection

Related resources

Refer to Oracle Database documentation for more information about the service.

Using database connection

To configure this credential, you'll need:

- A **User** name.
- A **Password** for that user.
- **Connection String**: The Oracle Database instance to connect to. The string can be an Easy Connect string, or a TNS Alias from a tnsnames.ora file, or the Oracle Database instance.
- **Use Optional Oracle Client Libraries**: If you want to work with Oracle Database advanced features, turn this on. This option internally uses node-oracledb Thick mode. Additional settings to enable node-oracledb Thick mode are required. Refer to Enabling Thick mode documentation for more information. This option is not available in the official n8n docker images.
- **Use SSL**: If your Connection String is using SSL, turn this on and configure additional details for the SSL Authentication.
- **Wallet Password**: The password to decrypt the Privacy Enhanced Mail (PEM)-encoded private certificate, if it is encrypted.
- **Wallet Content**: The security credentials required to establish a mutual TLS (mTLS) connection to Oracle Database.
- **Distinguished Name**: The distinguished name (DN) that should be matched with the certificate DN.
- **Match Distinguished Name**: Whether the server certificate DN should be matched in addition to the regular certificate verification that is performed.
- **Allow Weak Distinguished Name Match**: Whether the secure DN matching behavior which checks both the listener and server certificates has to be performed.
- **Pool Min**: The number of connections established to the database when a pool is created.
- **Pool Max**: The maximum number of connections to which a connection pool can grow.
- **Pool Increment**: The number of connections that are opened whenever a connection request exceeds the number of currently open connections.
- **Pool Maximum Session Life Time**: The number of connections that are opened whenever a connection request exceeds the number of currently open connections.
- **Pool Connection Idle Timeout**: The number of connections that are opened whenever a connection request exceeds the number of currently open connections.
- **Connection Class Name**: DRCP/PRCP Connection Class. Refer to Enabling DRCP for more information.
- **Connection Timeout**: The timeout duration in seconds for an application to establish an Oracle Net connection.
- **Transport Connection Timeout**: The maximum number of seconds to wait to establish a connection to the database host.
- **Keepalive Probe Interval**: The number of minutes between the sending of keepalive probes.

To set up your database connection credential:

1. Enter your database's username as the **User** in your n8n credential.

2. Enter the user's **Password**.

3. Enter your database's connection string as the **Connection String** in your n8n credential.
4. If your database uses SSL and you'd like to configure **SSL** for the connection, turn this option on in the credential. If you turn it on, enter the information of your Oracle Database SSL certificate in these fields:
    1. Enter the wallet password, if any, in the **Wallet Password** field.
    2. Enter PEM-encoded wallet file, **ewallet.pem** contents in the 'Expanded' layout of the **Wallet Content** field. This will ensure that all the whitespaces from the PEM-encoded wallet file are retained. Direct copy-paste into the **Wallet Content** field will strip out the whitespaces and lead to connection errors.

Refer to node-oracledb for more information on working with TLS connections.

Oura credentials

You can use these credentials to authenticate the following nodes:

- Oura

Prerequisites

Create an Oura account.

Supported authentication methods

- API access token

Related resources

Refer to Oura's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- A **Personal Access Token**: To generate a personal access token, go to the Personal Access Tokens page and select **Create A New Personal Access Token**.

Refer to How to Generate Personal Access Tokens for more information.

Paddle credentials

You can use these credentials to authenticate the following nodes:

- Paddle

Prerequisites

Create a Paddle account.

Supported authentication methods

- API access token (Classic)

Related resources

Refer to Paddle Classic's API documentation for more information about the service.

Using API access token (Classic)

To configure this credential, you'll need:

- A **Vendor Auth Code**: Created when you generate an API key.
- A **Vendor ID**: Displayed when you generate an API key.
- **Use Sandbox Environment API**: When turned on, nodes using this credential will hit the Sandbox API endpoint instead of the live API endpoint.

To generate an auth code and view your Vendor ID, go to **Paddle > Developer Tools > Authentication > Generate Auth Code**. Select **Reveal Auth Code** to display the Auth Code. Refer to API Authentication for more information.

PagerDuty credentials

You can use these credentials to authenticate the following nodes:

- PagerDuty

Prerequisites

Create a PagerDuty account.

Supported authentication methods

- API token
- OAuth2

Related resources

Refer to PagerDuty's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- A general access **API Token**: To generate an API token, go to **Integrations > Developer Tools > API Access Keys > Create New API Key**. Refer to Generate a General Access REST API key for more information.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you need to configure OAuth2 from scratch, register a new Pagerduty app.

Use these settings for registering your app:

- In the **Category** dropdown list, select **Infrastructure Automation**.
- In the **Functionality** section, select **OAuth 2.0**.

Once you **Save** your app, open the app details and edit your app configuration to use these settings:

- Within the **OAuth 2.0** section, select **Add**.
- Copy the **OAuth Callback URL** from n8n and paste it into the **Redirect URL** field.
- Copy the **Client ID** and **Client Secret** from PagerDuty and add these to your n8n credentials.
- Select **Read/Write** from the **Set Permission Scopes** dropdown list.

Refer to the instructions in App functionality for more information on available functionality. Refer to the PagerDuty OAuth Functionality documentation for more information on the OAuth flow.

PayPal credentials

You can use these credentials to authenticate the following nodes:

- PayPal
- PayPal Trigger

Prerequisites

Create a PayPal developer account.

Supported authentication methods

- API client and secret

Related resources

Refer to Paypal's API documentation for more information about the service.

Using API client and secret

To configure this credential, you'll need:

- A **Client ID**: Generated when you create an app.
- A **Secret**: Generated when you create an app.
- An **Environment**: Select **Live** or **Sandbox**.

To generate the **Client ID** and **Secret**, log in to your Paypal developer dashboard. Select **Apps & Credentials > Rest API apps > Create app**. Refer to Get client ID and client secret for more information.

Peekalink credentials

You can use these credentials to authenticate the following nodes:

- Peekalink

Prerequisites

Create a Peekalink account.

Supported authentication methods

- API key

Related resources

Refer to Peekalink's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: To get your API key, access your Peekalink dashboard and copy the key in the **Your API Key** section. Refer to Get your API key for more information.

Perplexity credentials

You can use these credentials to authenticate the following nodes:

- Perplexity

Supported authentication methods

- API key

Related resources

Refer to Perplexity's API documentation for more information about the service.

Using API key

To configure this credential, you'll need a Perplexity account and:

- **a Perplexity API key**: You can find out how to create a Perplexity API key in the Perplexity API getting started guide.

Refer to Perplexity's API documentation for more information about authenticating to the service.

PhantomBuster credentials

You can use these credentials to authenticate the following nodes:

- PhantomBuster

Prerequisites

Create a PhantomBuster account.

Supported authentication methods

- API key

Related resources

Refer to PhantomBuster's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: To get an API key, go to **Workspace settings** > **Third party API keys** and select **+ Add API Key**. Refer to How to find my API key for more information.

Philips Hue credentials

You can use these credentials to authenticate the following nodes:

- Philips Hue

Prerequisites

Create a Philips Hue account.

Supported authentication methods

- OAuth2

Related resources

Refer to Philips Hue's CLIP API documentation for more information about the service.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you're using the built-in OAuth connection, you don't need to enter an **APP ID**.

If you need to configure OAuth2 from scratch, you'll need a Philips Hue developer account

Create a new remote app on the Add new Hue Remote API app page.

Use these settings for your app:

- Copy the **OAuth Callback URL** from n8n and add it as a **Callback URL**.
- Copy the **AppId**, **ClientId**, and **ClientSecret** and enter these in the corresponding fields in n8n.

Pinecone credentials

You can use these credentials to authenticate the following nodes:

- Pinecone Vector Store

Supported authentication methods

- API key

Related resources

Refer to Pinecone's documentation for more information about the service.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

To configure this credential, you'll need a Pinecone account and:

- An **API Key**

To get an API key:

1. Open your Pinecone console.
2. Select the project you want to create an API key for. If you don't have any existing projects, create one. Refer to Pinecone's Quickstart for more information.
3. Go to **API Keys**.
4. Copy the API Key displayed there and enter it in your n8n credential.

Refer to Pinecone's API Authentication documentation for more information.

Pipedrive credentials

You can use these credentials to authenticate the following nodes:

- Pipedrive
- Pipedrive Trigger

Supported authentication methods

- API token
- OAuth2

Related resources

Refer to Pipedrive's developer documentation for more information about the service.

Using API token

To configure this credential, you'll need a Pipedrive account and:

- An **API Token**

To get your API token:

1. Open your **API Personal Preferences**.

2. Copy **Your personal API token** and enter it in your n8n credential.

If you have multiple companies, you'll need to select the correct company first:

1. Select your account name and be sure you're viewing the correct company.
2. Then select **Company Settings**.
3. Select **Personal Preferences**.
4. Select the **API** tab.
5. Copy **Your personal API token** and enter it in your n8n credential.

Refer to How to find the API token for more information.

Using OAuth2

To configure this credential, you'll need a Pipedrive developer sandbox account and:

- A **Client ID**
- A **Client Secret**

To get both, you'll need to register a new app:

1. Select your profile name in the upper right corner.
2. Find the company name of your sandbox account and select **Developer Hub**.
3. Select **Create an app**.
4. Select **Create public app**. The app's **Basic info** tab opens.
5. Enter an **App name** for your app, like `n8n integration`.
6. Copy the **OAuth Redirect URL** from n8n and add it as the app's **Callback URL**.
7. Select **Save**. The app's **OAuth & access scopes** tab opens.
8. Turn on appropriate **Scopes** for your app. Refer to Pipedrive node scopes and Pipedrive Trigger node scopes below for more guidance.
9. Copy the **Client ID** and enter it in your n8n credential.
10. Copy the **Client Secret** and enter it in your n8n credential.

Refer to Registering a public app for more information.

Pipedrive node scopes

The scopes you add to your app depend on which node(s) you want to use it for in n8n and what actions you want to complete with those.

Scopes you may need for the Pipedrive node:

Object	Node action	UI scope	Actual scope

Activity	Get data of an activity Get data of all activities	**Activities: Read only** or **Activities: Full Access**	`activities:read` or `activities:full`
Activity	Create Delete Update	**Activities: Full Access**	`activities:full`
Deal	Get data of a deal Get data of all deals Search a deal	**Deals: Read only** or **Deals: Full Access**	`deals:read` or `deals:full`
Deal	Create Delete Duplicate Update	**Deals: Full Access**	`deals:full`
Deal Activity	Get all activities of a deal	**Activities: Read only** or **Activities: Full Access**	`activities:read` or `activities:full`
Deal Product	Get all products in a deal	**Products: Read Only** or **Products: Full Access**	`products:read` or `products:full`
File	Download Get data of a file	Refer to note below	Refer to note below
File	Create Delete	Refer to note below	Refer to note below
Lead	Get data of a lead Get data of all leads	**Leads: Read only** or **Leads: Full access**	`leads:read` or `leads:full`
Lead	Create Delete Update	**Leads: Full access**	`leads:full`
Note	Get data of a note Get data of all notes	Refer to note below	Refer to note below
Note	Create Delete Update	Refer to note below	Refer to note below
Organization	Get data of an organization Get data of all organizations Search	**Contacts: Read Only** or **Contacts: Full Access**	`contacts:read` or `contacts:full`
Organization	Create Delete Update	**Contacts: Full Access**	`contacts:full`
Person	Get data of a person Get data of all persons	**Contacts: Read Only** or **Contacts: Full**	`contacts:read` or `contacts:full`

	Search	**Access**	
Person	Create Delete Update	**Contacts: Full Access**	`contacts:full`
Product	Get data of all products	**Products: Read Only**	`products:read`

The Pipedrive node also supports Custom API calls. Add relevant scopes for whatever custom API calls you intend to make.

Refer to Scopes and permissions explanations for more information.

Pipedrive Trigger node scopes

The Pipedrive Trigger node requires the **Webhooks: Full access** (`webhooks:full`) scope.

Plivo credentials

You can use these credentials to authenticate the following nodes:

- Plivo

Prerequisites

Create a Plivo account.

Supported authentication methods

- Basic auth

Related resources

Refer to Plivo's API documentation for more information about the service.

Using basic auth

To configure this credential, you'll need:

- An **Auth ID**: Acts like your username. Copy yours from the **Overview** page of the Plivo console.
- An **Auth Token**: Acts like a password. Copy yours from the **Overview** page of the Plivo console.

Refer to How can I change my Auth ID or Auth Token? for more detailed instructions.

Postgres credentials

You can use these credentials to authenticate the following nodes:

- Postgres
- Agent
- Postgres Chat Memory
- PGVector Vector Store

Prerequisites

Create a user account on a Postgres server.

Supported authentication methods

- Database connection

Related resources

Refer to Postgres's documentation for more information about the service.

Using database connection

To configure this credential, you'll need:

- The **Host** or domain name for the server.
- The **Database** name.
- A **User** name.
- A user **Password**.
- **Ignore SSL Issues**: Set whether the credential connects if SSL validation fails.
- **SSL**: Choose whether to use SSL in your connection.
- The **Port** number to use for the connection.
- **SSH Tunnel**: Choose if you want to use SSH to encrypt the network connection with the Postgres server.

To set up the database connection:

1. Enter the **Host** or domain name for the Postgres server. You can either run the `/conninfo` command to confirm the host name or run this query:

   ```
   SELECT inet_server_addr();
   ```

2. Enter the **Database** name. Run the `/conninfo` command to confirm the database name.

3. Enter the **User** name of the user you wish to connect as.

4. Enter the user's **Password**.

5. **Ignore SSL Issues**: If you turn this on, the credential will connect even if SSL validation fails.

6. **SSL**: Choose whether to use SSL in your connection. Refer to Postgres SSL Support for more information. Options include:

    - **Allow**: Sets the `ssl-mode` parameter to `allow`. First try a non-SSL connection; if that fails, try an SSL connection.
    - **Disable**: Sets the `ssl-mode` parameter to `disable`. Only try a non-SSL connection.
    - **Require**: Sets the `ssl-mode` parameter to `require`. Only try an SSL connection. If a root CA file is present, verify that a trusted certificate authority (CA) issued the server certificate.

7. Enter the **Port** number to use for the connection. You can either run the `/conninfo` command to confirm the host name or run this query:

    ```
    SELECT inet_server_port();
    ```

8. **SSH Tunnel**: Turn this setting on to connect to the database over SSH. Refer to SSH tunnel limitations for some guidance around using SSH. Once turned on, you'll need:

    1. Select **SSH Authenticate with** to set the SSH Tunnel type to build:
        - Select **Password** if you want to connect to SSH using a password.
        - Select **Private Key** if you want to connect to SSH using an identity file (private key) and a passphrase.
    2. Enter the remote bind address you're connecting to as the **SSH Host**.
    3. **SSH Port**: Enter the local port number for the SSH tunnel.
    4. **SSH Postgres Port**: Enter the remote end of the tunnel, the port number the database server is using.
    5. **SSH User**: Enter the username to log in as.
    6. If you selected **Password** for SSH Authenticate with, add the user's **SSH Password**.
    7. If you selected **Private Key** for **SSH Authenticate with**:
        1. Add the contents of the **Private Key** or identity file used for SSH.
        2. If the **Private Key** was created with a passphrase, enter that **Passphrase**. If the **Private Key** has no passphrase, leave this field blank.

Refer to Secure TCP/IP Connections with SSH Tunnels for more information.

SSH tunnel limitations

Only use the **SSH Tunnel** setting if:

- You're using the credential with the Postgres node (Agent node doesn't support SSH tunnels).
- You have an SSH server running on the same machine as the Postgres server.
- You have a user account that can log in using `ssh`.

PostHog credentials

You can use these credentials to authenticate the following nodes:

- PostHog

Prerequisites

Create a PostHog account or host PostHog on your server.

Supported authentication methods

- API key

Related resources

Refer to PostHog's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- The API **URL**: Enter the correct domain for your API requests:
  - On US Cloud, use `https://us.i.posthog.com` for public POST-only endpoints or `https://us.posthog.com` for private endpoints.
  - On EU Cloud, use `https://eu.i.posthog.com` for public POST-only endpoints or `https://eu.posthog.com` for private endpoints.
  - For self-hosted instances, use your self-hosted domain.
  - Confirm yours by checking your PostHog instance URL.
- An **API Key**: The API key you use depends on whether you're accessing public or private endpoints:
  - For public POST-only endpoints, use a Project API key from your project's **General** Settings.
  - For private endpoints, use a Personal API key from your User account's **Personal API Keys** Settings. Refer to How to obtain a personal API key for more information.

Postmark credentials

You can use these credentials to authenticate the following nodes:

- Postmark Trigger

Prerequisites

Create a Postmark account on a Postmark server.

Supported authentication methods

- API token

Related resources

Refer to Postmark's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- A **Server API Token**: The Server API token is accessible by Account Owners, Account Admins, and users who have Server Admin privileges on a server. Get yours from the **API Tokens** tab under your Postmark server. Refer to API Authentication for more information.

ProfitWell credentials

You can use these credentials to authenticate the following nodes:

- ProfitWell

Prerequisites

Create a ProfitWell account.

Supported authentication methods

- API token

Related resources

Refer to Profitwell's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- An **API Token**: To get an API key or token, go to **Account Settings > Integrations** and select **ProfitWell API**.

Pushbullet credentials

You can use these credentials to authenticate the following nodes:

- Pushbullet

Prerequisites

Create a Pushbullet account.

Supported authentication methods

- OAuth2

Related resources

Refer to Pushbullet's API documentation for more information about the service.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated when you create a Pushbullet app, also known as an OAuth client.
- A **Client Secret**: Generated when you create a Pushbullet app, also known as an OAuth client.

To generate the **Client ID** and **Client Secret**, go to the create client page. Copy the **OAuth Redirect URL** from n8n and add this as your **redirect_uri** for the app/client. Use the **client_id** and **client_secret** from the OAuth Client in your n8n credential.

Refer to Pushbullet's OAuth2 Guide for more information.

Pushcut credentials

You can use these credentials to authenticate the following nodes:

- Pushcut
- Pushcut Trigger

Prerequisites

Download the Pushcut app.

Supported authentication methods

- API key

Related resources

Refer to Pushcut's Guides documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: To generate an API key, go to **Account > Integrations > Add API Key**. Refer to Create an API key for more information.

Pushover credentials

You can use these credentials to authenticate the following nodes:

- Pushover

Prerequisites

Create a Pushover account.

Supported authentication methods

- API key

Related resources

Refer to Pushover's API documentation for more information about authenticating with the service.

Using API Key

To configure this credential, you'll need:

- An **API Key**: Generated when you register an application. Refer to Application Registration for more information.

QRadar credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Qradar account.

Supported authentication methods

- API key

Related resources

Refer to QRadar's documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API key

To configure this credential, you'll need:

- An **API Key**: Also known as an authorized service token. Use the **Manage Authorized Services** window on the **Admin** tab to create an authentication token. Refer to Creating an authentication token for more information.

Qdrant credentials

You can use these credentials to authenticate the following nodes:

- Qdrant Vector Store

Supported authentication methods

- API key

Related resources

Refer to Qdrant's documentation for more information.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

To configure this credential, you'll need a Qdrant cluster and:

- An **API Key**
- Your **Qdrant URL**

To set it up:

1. Go to the Cloud Dashboard.
2. Select **Access Management** to display available API keys (or go to the **API Keys** section of the **Cluster detail** page).
3. Select **Create**.
4. Select the cluster you want the key to have access to in the dropdown.
5. Select **OK**.
6. Copy the API Key and enter it in your n8n credential.
7. Enter the URL for your Qdrant cluster in the **Qdrant URL**. Refer

to Qdrant Web UI for more information.

Refer to Qdrant's authentication documentation for more information on creating and using API keys.

Qualys credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Qualys user account with any user role except Contact.

Supported authentication methods

- Basic auth

Related resources

Refer to Qualys's documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using basic auth

To configure this credential, you'll need:

- A **Username**
- A **Password**
- A **Requested With** string: Enter a user description, like a user agent, or keep the default `n8n application`. This sets the required `X-Requested-With` header.

QuestDB credentials

You can use these credentials to authenticate the following nodes:

- QuestDB

Prerequisites

Create a user account on an instance of QuestDB.

Supported authentication methods

- Database connection

Related resources

Refer to QuestDB's documentation for more information about the service.

Using database connection

To configure this credential, you'll need:

- The **Host**: Enter the host name or IP address for the server.
- The **Database**: Enter the database name, for example `qdb`.
- A **User**: Enter the username for the user account as configured in `pg.user` or `pg.readonly.user` property in `server.conf`. Default value is `admin`.
- A **Password**: Enter the password for the user account as configured in `pg.password` or `pg.readonly.password` property in `server.conf`. Default value is `quest`.
- **SSL**: Select whether the connection should use SSL, which sets the `sslmode` parameter. Options include:
  - **Allow**
  - **Disable**
  - **Require**
- The **Port**: Enter the port number to use for the connection. Default is `8812`.

Refer to List of supported connection properties for more information.

Quick Base credentials

You can use these credentials to authenticate the following nodes:

- Quick Base

Prerequisites

Create a Quick Base account.

Supported authentication methods

- API key

Related resources

Refer to Quick Base's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- A **Hostname**: The string of characters located between `https://` and `/db` in your Quick Base URL.
- A **User Token**: To generate a token, select your **Profile > My preferences > My User Information > Manage my user tokens**. Refer to Creating and using user tokens for detailed instructions.

QuickBooks credentials

You can use these credentials to authenticate the following nodes:

- QuickBooks

Prerequisites

Create an Intuit developer account.

Supported authentication methods

- OAuth2

Related resources

Refer to Intuit's API documentation for more information about the service.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated when you create an app.
- A **Client Secret**: Generated when you create an app.
- An **Environment**: Select whether this credential should access your **Production** or **Sandbox** environment.

To generate your **Client ID** and **Client Secret**, create an app.

Use these settings when creating your app:

- Select appropriate scopes for your app. Refer to Learn about scopes for more information.
- Enter the **OAuth Redirect URL** from n8n as a **Redirect URI** in the app's **Development > Keys & OAuth** section.
- Copy the **Client ID** and **Client Secret** from the app's **Development > Keys & OAuth** section to enter in n8n. Refer to Get the Client ID and Client Secret for your app for more information.

Refer to Intuit's Set up OAuth 2.0 documentation for more information on the entire process.

RabbitMQ credentials

You can use these credentials to authenticate the following nodes:

- RabbitMQ
- RabbitMQ Trigger

Supported authentication methods

- User connection

Related resources

Refer to RabbitMQ's Connections documentation for more information about the service.

Using user connection

To configure this credential, you'll need to have a RabbitMQ broker installed and:

1. Enter the **Hostname** for the RabbitMQ broker.
2. Enter the **Port** the connection should use.
3. Enter a **User** the connection should use to log in as.
    - The default is `guest`. RabbitMQ recommends using a different user in production environments. Refer to Access Control | The Basics for more information. If you're using the `guest` account with a non-localhost connection, refer to `guest` user issues below for troubleshooting tips.
4. Enter the user's **Password**.
    - The default password for the `guest` user is `guest`.
5. Enter the virtual host the connection should use as the **Vhost**. The default virtual host is `/`.
6. Select whether the connection should use **SSL**. If turned on, also set:
    - **Passwordless**: Select whether the SSL certificate connection users SASL mechanism EXTERNAL (turned off) or doesn't use a password (turned on). If turned on, you'll also need to enter:
        - The **Client Certificate**: Paste the text of the SSL client certificate to use.
        - The **Client Key**: Paste the SSL client key to use.
        - The **Passphrase**: Paste the SSL passphrase to use.
    - **CA Certificates**: Paste the text of the SSL CA certificates to use.

`guest` user issues

If you use the `guest` user for the credential and you try to access a remote host, you may see a connection error. The RabbitMQ logs show an error like this:

```
[error] <0.918.0> PLAIN login refused: user 'guest' can only connect
via localhost
```

This happens because RabbitMQ prohibits the default `guest` user from connecting from remote hosts. It can only connect over the `localhost`.

To resolve this error, you can:

- Update the `guest` user to allow it remote host access.
- Create or use a different user to connect to the remote host. The `guest` user is the only user limited by default.

Refer to "guest" user can only connect from localhost for more information.

Raindrop credentials

You can use these credentials to authenticate the following nodes:

- Raindrop

Prerequisites

Create a Raindrop account.

Supported authentication methods

- OAuth2

Related resources

Refer to Raindrop's API documentation for more information about the service.

Using OAuth

To configure this credential, you'll need:

- A **Client ID**
- A **Client Secret**

Generate both by creating a Raindrop app.

To create an app, go to **Settings > Integrations** and select **+ Create new app** in the **For Developers** section.

Use these settings for your app:

- Copy the **OAuth Redirect URL** from n8n and add it as a **Redirect URI** in your app.
- Copy the **Client ID** and **Client Secret** from the Raindrop app and enter them in your n8n credential.

Rapid7 InsightVM credentials

–8<– “_snippets/integrations/builtin/credentials/cred-only-statement.md”

Prerequisites

Create a Rapid7 InsightVM account.

Supported authentication methods

- API key

Related resources

Refer to Rapid7 InsightVM’s API documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n’s website.

Using API key

To configure this credential, you’ll need a Rapid7 InsightVM account and:

- A **URL**: The API endpoint URL where the resource or data you are requesting lives. You can find more information about the expected format in the endpoint section of the Rapid7’s API overview.
- An **API Key**: Refer to Rapid7’s Managing Platform API Keys documentation to create an API key.

Refer to Rapid7 InsightVM’s API documentation for more information about authenticating to the service.

Recorded Future credentials

–8<– “_snippets/integrations/builtin/credentials/cred-only-statement.md”

Prerequisites

Create a Recorded Future account.

Supported authentication methods

- API access token

Related resources

Refer to Recorded Future's documentation for more information about the service. The rest of Recorded Future's help center requires a paid account.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API access token

To configure this credential, you'll need:

- An API **Access Token**

Refer to the Recorded Future APIs documentation for more information on getting your API access token.

Reddit credentials

You can use these credentials to authenticate the following nodes:

- Reddit

Prerequisites

Create a Reddit account.

Supported authentication methods

- OAuth2

Related resources

Refer to Reddit's developer documentation for more information about the service.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**
- A **Client Secret**

Generate both by creating a third-party app. Visit the previous link or go to your **profile > Settings > Safety & Privacy > Manage third-party app authorization > are you a developer? create an app**.

Use these settings for your app:

- Copy the **OAuth Callback URL** from n8n and use it as your app's

redirect uri.
- The app's client ID displays underneath your app name. Copy that and add it as your n8n **Client ID**.
- Copy the app's **secret** and add it as your n8n **Client Secret**.

Redis credentials

You can use these credentials to authenticate the following nodes:

- Redis
- Redis Chat Memory
- Redis Vector Store

Supported authentication methods

- Database connection

Related resources

Refer to Redis's developer documentation for more information about the service.

Using database connection

You'll need a user account on a Redis server and:

- A **Password**
- The **Host** name
- The **Port** number
- A **Database Number**
- **SSL**

To configure this credential:

1. Enter your user account **Password**.
2. Enter the **Host** name of the Redis server. The default is `localhost`.
3. Enter the **Port** number the connection should use. The default is `6379`.
    - This number should match the `tcp_port` listed when you run the `INFO` command.
4. Enter the **Database Number**. The default is `0`.
5. If the connection should use SSL, turn on the **SSL** toggle. If this toggle is off, the connection uses TCP only.
6. If you enable **SSL**, you have the option to **disable TLS verification**. Toggle to use self-signed certificates. WARNING: This makes the connection less secure.

Refer to Connecting to Redis | Generic client for more information.

Rocket.Chat credentials

You can use these credentials to authenticate the following nodes:

- Rocket.Chat

Prerequisites

- Create a Rocket.Chat account.
- Your account must have the `create-personal-access-tokens` permission to generate personal access tokens.

Supported authentication methods

- API access token

Related resources

Refer to Rocket.Chat's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- Your **User ID**: Displayed when you generate an access token.
- An **Auth Key**: Your personal access token. To generate an access token, go to your **avatar > Account > Personal Access Tokens**. Copy the token and add it as the n8n **Auth Key**.
- Your Rocket.Chat **Domain**: Also known as your default URL or workspace URL.

Refer to Personal Access Tokens for more information.

Rundeck credentials

You can use these credentials to authenticate the following nodes:

- Rundeck

Prerequisites

Create a user account on a Rundeck server.

Supported authentication methods

- API token

Related resources

Refer to Rundeck's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- Your **URL**: Enter the base URL of your Rundeck server, for example `http://myserver:4440`. Refer to URLs for more information.
- A user API **Token**: To generate a user API token, go to your **Profile > User API Tokens**. Refer to User API tokens for more information.

S3 credentials

You can use these credentials to authenticate the following nodes:

- S3

Prerequisites

Create an account on an S3-compatible server. Use the S3 node for generic or non-AWS S3 like:

- DigitalOcean Spaces
- MinIO
- Wasabi

Supported authentication methods

- S3 endpoint

Related resources

Refer to your S3-compatible provider's documentation for more information on the services. For example, refer to Wasabi's REST API documentation or DigitalOcean's Spaces API Reference Documentation.

Using S3 endpoint

To configure this credential, you'll need:

- An **S3 Endpoint**: Enter the URL endpoint for the S3 storage backend.
- A **Region**: Enter the region for your S3 storage. Some providers call this the "region slug."
- An **Access Key ID**: Enter the S3 access key your S3 provider uses to access the bucket or space. Some providers call this API keys.
- A **Secret Access Key**: Enter the secret access key for the **Access Key ID**.

- **Force Path Style**: When turned on, the connection uses path-style addressing for buckets.
- **Ignore SSL Issues**: When turned on, n8n will connect even if SSL certificate validation fails.

More detailed instructions for DigitalOcean Spaces and Wasabi follow. If you're using a different provider, refer to their documentation for more information.

Using DigitalOcean Spaces

To configure the credential for use with DigitalOcean spaces:

1. In DigitalOceans, go to the control panel and open **Settings**. Your endpoint should be listed there. Prepend `https://` to that endpoint and enter it as the **S3 Endpoint** in n8n.
    - Your DigitalOceans endpoint depends on the data center region your bucket's in.
2. For the **Region**, enter the region your bucket's located in, for example, `nyc3`.
    - If you plan to use this credential to create new Spaces, enter `us-east-1` instead.
3. From your DigitalOceans control panel, go to **API**.
4. Open the **Spaces Keys** tab.
5. Select **Generate New Key**.
6. Enter a **Name** for your key, like `n8n integration` and select the checkmark.
7. Copy the **Key** displayed next to the name and enter this as the **Access Key ID** in n8n.
8. Copy the **Secret** value and enter this as the **Secret Access Key** in n8n.
    - Refer to Sharing Access to Buckets with Access Keys for more information on generating the key and secret.
9. Keep the **Force Path Style** toggle turned off unless you want to use subdomain/virtual calling format.
10. Decide how you want the n8n credential to handle SSL:
    - To respect SSL certificate validation, keep the default of **Ignore SSL Issues** turned off.
    - To connect even if SSL certificate validation fails, turn on **Ignore SSL Issues**.

Refer to DigitalOcean's Spaces API Reference Documentation for more information.

Using Wasabi

To configure the credential for use with Wasabi:

1. For the **S3 Endpoint**, enter the service URL for your bucket's region. Start it with `https://`.
    - Refer to Service URLs for Wasabi's Storage Regions to identify the correct URL.
2. For the **Region**, enter the region slug portion of the service URL. For example, if you entered `https://s3.us-east-2.wasabisys.com` as the **S3 Endpoint**, `us-east-2` is the region.
3. Log into you Wasabi Console as the root user.
4. Open the **Menu** and select **Access Keys**.
5. Select **CREATE NEW ACCESS KEY**.
6. Select whether the key is for the **Root User** or a **Sub-User** and

select **CREATE**.
7. Copy the **Access Key** and enter it in n8n as the **Access Key ID**.
8. Copy the **Secret Key** and enter it in n8n as the **Secret Access Key**.
   - Refer to Creating a New Access Key for more information on generating the key and secret.
9. Wasabi recommends turning on the **Force Path Style** toggle "because the path-style offers the greatest flexibility in bucket names, avoiding domain name issues." Refer to the Wasabi REST API Introduction for more information.
10. Decide how you want the n8n credential to handle SSL:
    - To respect SSL certificate validation, keep the default of **Ignore SSL Issues** turned off.
    - To connect even if SSL certificate validation fails, turn on **Ignore SSL Issues**.

Salesforce credentials

You can use these credentials to authenticate the following nodes:

- Salesforce
- Salesforce trigger

Supported authentication methods

- JWT
- OAuth2

Related resources

Refer to Salesforce's developer documentation for more information about the service.

Using JWT

To configure this credential, you'll need a Salesforce account and:

- Your **Environment Type** (Production or Sandbox)
- A **Client ID**: Generated when you create a connected app.
- Your Salesforce **Username**
- A **Private Key** for a self-signed digital certificate

To set things up, first you'll create a private key and certificate, then a connected app:

1. In n8n, select the **Environment Type** for your connection. Choose the option that best describes your environment from **Production** or **Sandbox**.
2. Enter your Salesforce **Username**.
3. Log in to your org in Salesforce.
4. You'll need a private key and certificate issued by a certification authority. Use your own key/cert or use OpenSSL to create a key and a self-signed digital certificate. Refer to the Salesforce Create

a Private Key and Self-Signed Digital Certificate documentation for instructions on creating your own key and certificate.
5. From **Setup** in Salesforce, enter `App Manager` in the Quick Find box, then select **App Manager**.
6. On the App Manager page, select **New Connected App**.
7. Enter the required **Basic Info** for your connected app, including a **Name** and **Contact Email address**. Refer to Salesforce's Configure Basic Connected App Settings documentation for more information.
8. Check the box to **Enable OAuth Settings**.
9. For the **Callback URL**, enter `http://localhost:1717/OauthRedirect`.
10. Check the box to **Use digital signatures**.
11. Select **Choose File** and upload the file that contains your digital certificate, such as `server.crt`.
12. Add these **OAuth scopes**:
    - **Full access (full)**
    - **Perform requests at any time (refresh_token, offline_access)**
13. Select **Save**, then **Continue**. The **Manage Connected Apps** page should open to the app you just created.
14. In the **API (Enable OAuth Settings)** section, select **Manage Consumer Details**.
15. Copy the **Consumer Key** and add it to your n8n credential as the **Client ID**.
16. Enter the contents of the private key file in n8n as **Private Key**.
    - Use the multi-line editor in n8n.
    - Enter the private key in standard PEM key format: `-----BEGIN PRIVATE KEY-----  KEY DATA GOES HERE  -----END PRIVATE KEY-----`

These steps are what's required on the n8n side. Salesforce recommends setting refresh token policies, session policies, and OAuth policies too:

14. In Salesforce, select **Back to Manage Connected Apps**.
15. Select **Manage**.
16. Select **Edit Policies**.
17. Review the **Refresh Token Policy** field. Salesforce recommends using expire refresh token after 90 days.
18. In the **Session Policies** section, Salesforce recommends setting **Timeout Value** to 15 minutes.
19. In the **OAuth Policies** section, select **Admin approved users are pre-authorized for permitted users** for **Permitted Users**, and select **OK**.
20. Select **Save**.
21. Select **Manage Profiles**, select the profiles that are pre-authorized to use this connected app, and select **Save**.
22. Select **Manage Permission Sets** to select the permission sets. Create permission sets if necessary.

Refer to Salesforce's Create a Connected App in Your Org documentation for more information.

Using OAuth2

To configure this credential, you'll need a Salesforce account.

–8<– “_snippets/integrations/builtin/credentials/cloud-oauth-button.md”

Cloud and hosted users will need to select your **Environment Type**. Choose between **Production** and **Sandbox**.

If you’re self-hosting n8n, you’ll need to configure OAuth2 from scratch by creating a connected app:

1. In n8n, select the **Environment Type** for your connection. Choose the option that best describes your environment from **Production** or **Sandbox**.
2. Enter your Salesforce **Username**.
3. Log in to your org in Salesforce.
4. From **Setup** in Salesforce, enter `App Manager` in the Quick Find box, then select **App Manager**.
5. On the App Manager page, select **New Connected App**.
6. Enter the required **Basic Info** for your connected app, including a **Name** and **Contact Email address**. Refer to Salesforce’s Configure Basic Connected App Settings documentation for more information.
7. Check the box to **Enable OAuth Settings**.
8. For the **Callback URL**, enter `http://localhost:1717/OauthRedirect`.
9. Add these **OAuth scopes**:
    - **Full access (full)**
    - **Perform requests at any time (refresh_token, offline_access)**
10. Make sure the following settings are unchecked:
    - **Require Proof Key for Code Exchange (PKCE) Extension for Supported Authorization Flows**
    - **Require Secret for Web Server Flow**
    - **Require Secret for Refresh Token Flow**
11. Select **Save**, then **Continue**. The **Manage Connected Apps** page should open to the app you just created.
12. In the **API (Enable OAuth Settings)** section, select **Manage Consumer Details**.
13. Copy the **Consumer Key** and add it to your n8n credential as the **Client ID**.
14. Copy the **Consumer Secret** and add it to your n8n credential as the **Client Secret**.

These steps are what’s required on the n8n side. Salesforce recommends setting refresh token policies and session policies, too:

14. In Salesforce, select **Back to Manage Connected Apps**.
15. Select **Manage**.
16. Select **Edit Policies**.
17. Review the **Refresh Token Policy** field. Salesforce recommends using expire refresh token after 90 days.
18. In the **Session Policies** section, Salesforce recommends setting **Timeout Value** to 15 minutes.

Refer to Salesforce’s Create a Connected App in Your Org documentation for more information.

Common issues

Connection issues when authenticating with Salesforce from n8n Cloud

If you encounter connection issues when authenticating with Salesforce from n8n Cloud, you might need to enable a specific system permission in your Salesforce user profiles:

1. In Salesforce, go to **Setup**.
2. In the **Quick Find** box, search for `Profiles`.
3. Select the profile used by the user connecting to n8n (for example, System Administrator or the relevant profile).
4. Click **Edit** or use the new **Profile** interface if it's available.
5. Locate the **Administrative Permissions** section.
6. Enable the checkbox for **Approve Connected Apps for Non-Admins**. This checkbox might also appear as **Approve apps connected not installed** depending on your Salesforce language or translation.
7. Click **Save**.

This permission is not enabled by default, even for administrator profiles, and must be manually activated. Without this permission, you might experience authentication failures when trying to connect n8n to Salesforce.

Salesmate credentials

You can use these credentials to authenticate the following nodes:

- Salesmate

Prerequisites

Create a Salesmate account.

Supported authentication methods

- API token

Related resources

Refer to Salesmate's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- A **Session Token**: An **Access Key**. Generate an access key in **My Account > Access Key**. Refer to Access Rights and Keys for more information.
- A **URL**: Your Salesmate domain name/base URL, for example `n8n.salesmate.io`.

SearXNG credentials

You can use these credentials to authenticate the following nodes:

- SearXNG Tool

Supported authentication methods

- API URL

Related resources

Refer to SearXNG's documentation for more information about the service.

Using API URL

To configure this credential, you'll need an instance of SearXNG running at an URL that's accessible from n8n:

- **API URL**: The URL of the SearXNG instance you want to connect to.

Refer to SearXNG's Administrator documentation for more information about running the service.

SeaTable credentials

You can use these credentials to authenticate the following nodes:

- SeaTable
- SeaTable Trigger

Prerequisites

Create a SeaTable account on either a cloud or self-hosted SeaTable server.

Supported authentication methods

- API key

Related resources

Refer to SeaTable's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **Environment**: Select the environment that matches your SeaTable instance:
  - **Cloud-Hosted**
  - **Self-Hosted**
- An **API Token (of a Base)**: Generate a **Base-Token** in SeaTable from the base options **> Advanced > API Token**.
  - Use **Read-Write** permission for your token.
  - Refer to Creating an API token for more information.
- A **Timezone**: Select the timezone of your SeaTable server.

SecurityScorecard credentials

You can use these credentials to authenticate the following nodes:

- SecurityScorecard

Prerequisites

Create a SecurityScorecard account.

Supported authentication methods

- API key

Related resources

Refer to SecurityScorecard's Developer documentation and API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Generate an API key in one of two ways:
  - As a user in **My Settings > API**. Refer to Get an API key for more information.
  - As a bot user: View the bot user and select **create token**. Refer to Authenticate with a bot user for more information.

Segment credentials

You can use these credentials to authenticate the following nodes:

- Segment

Prerequisites

Create a Segment account.

Supported authentication methods

- API key

Related resources

Refer to Segment's Sources documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- A **Write Key**: To get a Write Key, go to **Sources > Add Source**. Add a **Node.js** source and copy that write key to add to your n8n credential.

Refer to Locate your Write Key for more information.

Sekoia credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Sekoia SOC platform account.

Supported authentication methods

- API key

Related resources

Refer to Sekoia's documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API key

To configure this credential, you'll need:

- An **API Key**: To generate an API key, select **+ API Key**. Refer to Create an API key for more information.

Send Email credentials

You can use these credentials to authenticate the following nodes:

- Send Email

Prerequisites

- Create an email account on a service that supports SMTP.
- Some email providers require that you enable or set up outgoing SMTP or generate an app password. Refer to your provider's documentation to see if there are other required steps.

Supported authentication methods

- SMTP account

Related resources

Simple Message Transfer Protocol (SMTP) is a standard protocol for sending and receiving email. Most email providers offer instructions on setting up their service with SMTP. Refer to your provider's SMTP instructions.

Using SMTP account

To configure this credential, you'll need:

- A **User** email address
- A **Password**: This may be the user's password or an app password. Refer to the documentation for your email provider.
- The **Host**: The SMTP host address for your email provider, often formatted as `smtp.<provider>.com`. Check with your provider.
- A **Port** number: The port depends on the encryption method:
  - Port `465` for SSL/TLS (implicit encryption)
  - Port `587` for STARTTLS (explicit encryption)
  - Port `25` for no encryption (not recommended) Check with your email provider for their specific requirements.
- **SSL/TLS**: This toggle controls the encryption method:
  - Turn **ON** for port `465` (uses implicit SSL/TLS encryption)
  - Turn **OFF** for port `587` (uses STARTTLS explicit encryption)
  - Turn **OFF** for port `25` (no encryption)
- **Disable STARTTLS**: When SSL/TLS is disabled, the SMTP server can still try to upgrade the TCP connection using STARTTLS. Turning this on prevents that behaviour.
- **Client Host Name**: If needed by your provider, add a client host name. This name identifies the client to the server.

Provider instructions

Refer to the quickstart guides for these common email providers.

Gmail

Refer to Gmail.

Outlook.com

Refer to Outlook.com.

Yahoo

Refer to Yahoo.

My provider isn't listed

If your email provider isn't listed here, search for `SMTP settings` to find their instructions. (These instructions may also be included with `IMAP settings` or `POP settings`.)

Gmail Send Email credentials

Follow these steps to configure the Send Email credentials with a Gmail account.

Prerequisites

To follow these instructions, you must first:

1. Enable 2-step Verification on your Gmail account.
2. Generate an app password.

Enable 2-step Verification

–8<– "_snippets/integrations/builtin/credentials/email/gmail-two-step-verification.md"

Generate an app password

–8<– "_snippets/integrations/builtin/credentials/email/gmail-app-password.md"

Set up the credential

To set up the Send Email credential to use Gmail:

1. Enter your Gmail email address as the **User**.
2. Enter the app password you generated above as the **Password**.
3. Enter `smtp.gmail.com` as the **Host**.
4. For the **Port**:

    - Keep the default `465` for SSL or if you're unsure what to use.
    - Enter `587` for TLS.
5. Turn on the **SSL/TLS** toggle.

Refer to the Outgoing Mail (SMTP) Server settings in Read Gmail messages on other email clients using POP for more information. If the settings above don't work for you, check with your email administrator.

Outlook.com Send Email credentials

Follow these steps to configure the Send Email credentials with an Outlook.com account.

Set up the credential

To configure the Send Email credential to use an Outlook.com account:

1. Enter your Outlook.com email address as the **User**.
2. Enter your Outlook.com password as the **Password**.
3. Enter `smtp-mail.outlook.com` as the **Host**.
4. Enter `587` for the **Port**.
5. Turn on the **SSL/TLS** toggle.

Refer to Microsoft's POP, IMAP, and SMTP settings for Outlook.com documentation for more information. If the settings above don't work for you, check with your email administrator.

Use an app password

–8<– "_snippets/integrations/builtin/credentials/email/outlook-app-password.md"

Yahoo Send Email credentials

Follow these steps to configure the Send Email credentials with a Yahoo account.

Prerequisites

To follow these instructions, you must first generate an app password:

–8<– "_snippets/integrations/builtin/credentials/email/yahoo-app-password.md"

Set up the credential

To configure the Send Email credential to use Yahoo Mail:

1. Enter your Yahoo email address as the **User**.
2. Enter the app password you generated above as the **Password**.
3. Enter `smtp.mail.yahoo.com` as the **Host**.
4. For the **Port**:
   - Keep the default `465` for SSL or if you're unsure what to use.
   - Enter `587` for TLS.
5. Turn on the **SSL/TLS** toggle.

Refer to IMAP server settings for Yahoo Mail for more information. If the settings above don't work for you, check with your email administrator.

SendGrid credentials

You can use these credentials to authenticate the following nodes:

- SendGrid

Supported authentication methods

- API key

Related resources

Refer to SendGrid's API documentation for more information about the service.

Using API key

To configure this credential, you'll need a SendGrid account and:

- An **API Key**

To create an API key:

1. In the Twilio SendGrid app, go to **Settings > API Keys**.
2. Select **Create API Key**.
3. Enter a **Name** for your API key, like `n8n integration`.
4. Select **Full Access**.
5. Select **Create & View**.
6. Copy the key and enter it in your n8n credential.

Refer to Create API Keys for more information.

Sendy credentials

You can use these credentials to authenticate the following nodes:

- Sendy

Prerequisites

Host a Sendy application.

Supported authentication methods

- API key

Related resources

Refer to Sendy's API documentation for more information about the service.

Using API Key

To configure this credential, you'll need:

- A **URL**: The URL of your Sendy application.
- An **API Key**: Get your API key from your user profile > **Settings > Your API Key**.

Sentry.io credentials

You can use these credentials to authenticate the following nodes:

- Sentry.io

Prerequisites

Create a Sentry.io account.

Supported authentication methods

- API token
- OAuth2
- Server API token: Use for self-hosted Sentry.

Related resources

Refer to Sentry.io's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- An API **Token**: Generate a **User Auth Token** in **Account > Settings > User Auth Tokens**. Refer to User Auth Tokens for more information.

Using OAuth

–8<– “_snippets/integrations/builtin/credentials/cloud-oauth-button.md”

If you need to configure OAuth2 from scratch, create an integration with these settings:

- Copy the n8n **OAuth Callback URL** and add it as an **Authorized Redirect URI**.
- Copy the **Client ID** and **Client Secret** and add them to your n8n credential.

Refer to Public integrations for more information on creating the integration.

Using Server API token

To configure this credential, you’ll need:

- An API **Token**: Generate a **User Auth Token** in **Account > Settings > User Auth Tokens**. Refer to User Auth Tokens for more information.
- The **URL** of your self-hosted Sentry instance.

Serp credentials

You can use these credentials to authenticate the following nodes:

- Serp

Prerequisites

Create a SerpApi account.

Supported authentication methods

- API key

Related resources

Refer to Serp’s API documentation for more information about the service.

–8<– “_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md”

Using API key

To configure this credential, you'll need:

- An **API Key**

To get your API key:

1. Go to **Your Account > API Key**.
2. Copy **Your Private API Key** and enter it as the **API Key** in your n8n credential.

ServiceNow credentials

You can use these credentials to authenticate the following nodes:

- ServiceNow

Prerequisites

Create a ServiceNow developer account.

Supported authentication methods

- Basic auth
- OAuth2

Related resources

Refer to ServiceNow's API documentation for more information about the service.

Using basic auth

To configure this credential, you'll need:

- A **User** name: Enter your ServiceNow username.
- A **Password**: Enter your ServiceNow password.
- A **Subdomain**: The subdomain for your servicenow instance is in your instance URL: `https://<subdomain>.service-now.com/`. For example, if the full URL is `https://dev99890.service-now.com`, then the subdomain is `dev99890`.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated once you register a new app.
- A **Client Secret**: Generated once you register a new app.
- A **Subdomain**: The subdomain for your servicenow instance is in your instance URL: `https://<subdomain>.service-now.com/`. For

example, if the full URL is `https://dev99890.service-now.com`, then the subdomain is `dev99890`.

To generate your **Client ID** and **Client Secret**, register a new app in **System OAuth > Application Registry > New > Create an OAuth API endpoint for external clients**. Use these settings for your app:

- Copy the **Client ID** and add it to your n8n credential.
- Enter a **Client Secret** or leave it blank to automatically generate a random secret. Add this secret to your n8n credential.
- Copy the n8n **OAuth Redirect URL** and add it as a **Redirect URL**.

Refer to How to setup OAuth2 authentication for RESTMessageV2 integrations for more information.

---

seven credentials

You can use these credentials to authenticate the following nodes:

- seven

Prerequisites

Create a seven developer account.

Supported authentication methods

- API key

Related resources

Refer to seven's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API key**: Go to **Account > Developer > API Keys** to create an API key. Refer to API First Steps for more information.

Shopify credentials

You can use these credentials to authenticate the following nodes with Shopify.

- Shopify
- Shopify Trigger

Supported authentication methods

- Access token (recommended): For private apps/single store use. Can be created by regular admins.
- OAuth2: For public apps. Must be created by partner accounts.
- API key: Deprecated.

Related resources

Refer to Shopify's authentication documentation for more information about the service.

Using access token

To configure this credential, you'll need a Shopify admin account and:

- Your **Shop Subdomain**
- An **Access Token**: Generated when you create a custom app.
- An **APP Secret Key**: Generated when you create a custom app.

To set up the credential, you'll need to create and install a custom app:

1. Enter your **Shop Subdomain**.
    - Your subdomain is within the URL: `https://<subdomain>.myshopify.com`. For example, if the full URL is `https://n8n.myshopify.com`, the Shop Subdomain is `n8n`.
2. In Shopify, go to **Admin > Settings > Apps and sales channels**.
3. Select **Develop apps**.
4. Select **Create a custom app**.
5. In the modal window, enter the **App name**.
6. Select an **App developer**. The app developer can be the store owner or any account with the **Develop apps** permission.
7. Select **Create app**.
8. Select **Select scopes**. In the **Admin API access scopes** section, select the API scopes you want for your app.
    - To use all functionality in the Shopify node, add the `read_orders`, `write_orders`, `read_products`, and `write_products` scopes.
    - Refer to Shopify API Access Scopes for more information on the available scopes.
9. Select **Save**.
10. Select **Install app**.
11. In the modal window, select **Install app**.
12. Open the app's **API Credentials** section.

13. Copy the **Admin API Access Token**. Enter this in your n8n credential as the **Access Token**.
14. Copy the **API Secret Key**. Enter this in your n8n credential as the **APP Secret Key**.

Refer to Creating a custom app and Generate access tokens for custom apps in the Shopify admin for more information on these steps.

Using OAuth2

To configure this credential, you'll need a Shopify partner account and:

- A **Client ID**: Generated when you create a custom app.
- A **Client Secret**: Generated when you create a custom app.
- Your **Shop Subdomain**

To set up the credential, you'll need to create and install a custom app:

1. Open your Shopify Partner dashboard.
2. Select **Apps** from the left navigation.
3. Select **Create app**.
4. In the **Use Shopify Partners** section, enter an **App name**.
5. Select **Create app**.
6. When the app details open, copy the **Client ID**. Enter this in your n8n credential.
7. Copy the **Client Secret**. Enter this in your n8n credential.
8. In the left menu, select **Configuration**.
9. In n8n, copy the **OAuth Redirect URL** and paste it into the **Allowed redirection URL(s)** in the **URLs** section.
10. In the **URLs** section, enter an **App URL** for your app. The host entered here needs to match the host for the **Allowed redirection URL(s)**, like the base URL for your n8n instance.
11. Select **Save and release**.
12. Select **Overview** from the left menu. At this point, you can choose to **Test your app** by installing it to one of your stores, or **Choose distribution** to distribute it publicly.
13. In n8n, enter the **Shop Subdomain** of the store you installed the app to, either as a test or as a distribution.
    - Your subdomain is within the URL: `https://<subdomain>.myshopify.com`. For example, if the full URL is `https://n8n.myshopify.com`, the Shop Subdomain is `n8n`.

Using API key

To configure this credential, you'll need:

- An **API Key**
- A **Password**
- Your **Shop Subdomain**: Your subdomain is within the URL: `https://<subdomain>.myshopify.com`. For example, if the full URL is `https://n8n.myshopify.com`, the Shop Subdomain is `n8n`.
- *Optional:* A **Shared Secret**

Common issues

Here are some common issues setting up the Shopify credential and steps to resolve or troubleshoot them.

Enable custom app development

If you don't see the option to **Create a custom app**, no one's enabled custom app development for your store.

To enable custom app development, you must log in either as a store owner or as a user with the **Enable app development** permission:

1. In Shopify, go to **Admin > Settings > Apps and sales channels**.
2. Select **Develop apps**.
3. Select **Allow custom app development**.
4. Read the warning and information provided and select **Allow custom app development**.

Forbidden credentials error

If you get a **Couldn't connect with these settings / Forbidden - perhaps check your credentials** warning when you test the credentials, this may be due to your app's access scope dependencies. For example, the `read_orders` scope also requires `read_products` scope. Review the scopes you have assigned and the action you're trying to complete.

Shuffler credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Shuffler account on either a cloud or self-hosted instance.

Supported authentication methods

- API key

Related resources

Refer to Shuffler's documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API key

To configure this credential, you'll need:

- An **API Key**: Get your API key from the **Settings** page.

SIGNL4 credentials

You can use these credentials to authenticate the following nodes:

- SIGNL4

Prerequisites

Create a SIGNL4 account.

Supported authentication methods

- Webhook secret

Related resources

Refer to SIGNL4's Inbound Webhook documentation for more information about the service.

Using webhook secret

To configure this credential, you'll need:

- A **Team Secret**: SIGNL4 includes this secret in the "✅ Sign up complete" email as the last part of the webhook URL. If your webhook URL is `https://connect.signl4.com/webhook/helloworld`, your team secret would be `helloworld`.

Slack credentials

You can use these credentials to authenticate the following nodes:

- Slack
- Slack Trigger

Supported authentication methods

- API access token:
  - Required for the Slack Trigger node.
  - Works with the Slack node, but not recommended.
- OAuth2:
  - Recommended method for the Slack node.
  - Doesn't work with the Slack Trigger node.

Related resources

Refer to Slack's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need a Slack account and:

- An **Access Token**

To generate an access token, create a Slack app:

1. Open your Slack API Apps page.
2. Select **Create New App > From scratch**.
3. Enter an **App Name**.
4. Select the **Workspace** where you'll be developing your app.
5. Select **Create App**. The app details open.
6. In the left menu under **Features**, select **OAuth & Permissions**.
7. In the **Scopes** section, select appropriate scopes for your app. Refer to Scopes for a list of recommended scopes.
8. After you've added scopes, go up to the **OAuth Tokens** section and select **Install to Workspace**. You must be a Slack workspace admin to complete this action.
9. Select **Allow**.
10. Copy the **Bot User OAuth Token** and enter it as the **Access Token** in your n8n credential.
11. If you're using this credential for the Slack Trigger, follow the steps in Slack Trigger configuration to finish setting up your app.

Refer to the Slack API Quickstart for more information.

Slack Trigger configuration

To use your Slack app with the Slack Trigger node:

1. Go to Your Apps in Slack and select the app you want to use.
2. Go to **Features > Event Subscriptions**.
3. Turn on the **Enable Events** control.
4. In n8n, copy the **Webhook URL** and enter it as the **Request URL** in your Slack app.
5. Once verified, select the bot events to subscribe to. Use the **Trigger on** field in n8n to filter these requests.
    - To use an event not in the list, add it as a bot event and select **Any Event** in the n8n node.

Refer to Quickstart | Configuring the app for event listening for more information.

n8n recommends enabling request signature verification for your Slack Trigger for additional security:

1. Go to Your Apps in Slack and select the app you want to use.
2. Go to **Settings > Basic Information**.

3. Copy the value of **Signing**.
4. In n8n, Paste this value into the **Signature Secret** field for the credential.

Using OAuth2

–8<– “_snippets/integrations/builtin/credentials/cloud-oauth-button.md”

If you’re self-hosting n8n and need to configure OAuth2 from scratch, you’ll need a Slack account and:

- A **Client ID**
- A **Client Secret**

To get both, create a Slack app:

1. Open your Slack API Apps page.
2. Select **Create New App > From scratch**.
3. Enter an **App Name**.
4. Select the **Workspace** where you’ll be developing your app.
5. Select **Create App**. The app details open.
6. In **Settings > Basic Information**, open the **App Credentials** section.
7. Copy the **Client ID** and **Client Secret**. Paste these into the corresponding fields in n8n.
8. In the left menu under **Features**, select **OAuth & Permissions**.
9. In the **Redirect URLs** section, select **Add New Redirect URL**.
10. Copy the **OAuth Callback URL** from n8n and enter it as the new Redirect URL in Slack.
11. Select **Add**.
12. Select **Save URLs**.
13. In the **Scopes** section, select appropriate scopes for your app. Refer to Scopes for a list of scopes.
14. After you’ve added scopes, go up to the **OAuth Tokens** section and select **Install to Workspace**. You must be a Slack workspace admin to complete this action.
15. Select **Allow**.
16. At this point, you should be able to select the OAuth button in your n8n credential to connect.

Refer to the Slack API Quickstart for more information. Refer to the Slack Installing with OAuth documentation for more details on the OAuth flow itself.

Scopes

Scopes determine what permissions an app has.

- If you want your app to act on behalf of users who authorize the app, add the required scopes under the **User Token Scopes** section.
- If you’re building a bot, add the required scopes under the **Bot Token Scopes** section.

Here’s the list of scopes the OAuth credential requires, which are a good starting point:

Scope name	Notes
channels:read	
channels:write	Not available as a bot token scope
channels:history	
chat:write	
files:read	
files:write	
groups:read	
groups:history	
im:read	
im:history	
mpim:read	
mpim:history	
reactions:read	
reactions:write	
stars:read	Not available as a bot token scope
stars:write	Not available as a bot token scope
usergroups:read	
usergroups:write	
users.profile:read	
users.profile:write	Not available as a bot token scope
users:read	
search:read	

Common issues

Token expired

–8<– “_snippets/integrations/builtin/credentials/slack/token-rotation.md”

Snowflake credentials

You can use these credentials to authenticate the following nodes:

- Snowflake

Prerequisites

Create a Snowflake account.

Supported authentication methods

- Database connection

Related resources

Refer to Snowflake's API documentation and SQL Command Reference for more information about the service.

Using database connection

To configure this credential, you'll need:

- An **Account** name: Your account name is the string of characters located between `https://` and `snowflakecomputing.com` in your Snowflake URL. For example, if the URL of your Snowflake account is `https://abc.eu-central-1.snowflakecomputing.com` then the name of your account is `abc.eu-central-1`.
- A **Database**: Enter the name of the database the credential should connect to.
- A **Warehouse**: Enter the name of the default virtual warehouse to use for the session after connecting. n8n uses this warehouse for performing queries, loading data, and so on.
- A **Username**
- A **Password**
- A **Schema**: Enter the schema you want to use after connecting.
- A **Role**: Enter the security role you want to use after connecting.
- **Client Session Keep Alive**: By default, client connections typically time out three or four hours after the most recent query execution. Turning this setting on sets the `clientSessionKeepAlive` parameter to true: the server will keep the client's connection alive indefinitely, even if the connection doesn't execute any queries.

Refer to Session Commands for more information on these settings.

SolarWinds IPAM credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Supported authentication methods

- Username & Password

Related resources

Refer to SolarWinds IPAM's API documentation for more information about the service.

Using Username & Password

To configure this credential, you'll need a SolarWinds IPAM account and:

- **URL**: The base URL of your SolarWinds IPAM server

- **Username**: The username you use to access SolarWinds IPAM
- **Password**: The password you use to access SolarWinds IPAM

Refer to SolarWinds IPAM's API documentation for more information about authenticating to the service.

SolarWinds Observability SaaS credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Supported authentication methods

- API Token

Related resources

Refer to SolarWinds Observability SaaS's API documentation for more information about the service.

Using API Token

To configure this credential, you'll need a SolarWinds Observability SaaS account and:

- **URL**: The URL you use to access the SolarWinds Observability SaaS platform
- **API Token**: An API token found in the SolarWinds Observability SaaS platform under **Settings > Api Tokens**

Refer to SolarWinds Observability SaaS's API documentation for more information about authenticating to the service.

Splunk credentials

You can use these credentials to authenticate the following nodes:

- Splunk

Prerequisites

- Download and install Splunk Enterprise.
- Enable token authentication in **Settings > Tokens**.

Supported authentication methods

- API auth token

Related resources

Refer to Splunk's Enterprise API documentation for more information about the service.

Using API auth token

To configure this credential, you'll need:

- An **Auth Token**: Once you've enabled token authentication, create an auth token in **Settings > Tokens**. Refer to Creating authentication tokens for more information.
- A **Base URL**: For your Splunk instance. This should include the protocol, domain, and port, for example: `https://localhost:8089`.
- **Allow Self-Signed Certificates**: If turned on, n8n will connect even if SSL validation fails.

Required capabilities

Your Splunk platform account and role must have certain capabilities to create authentication tokens:

- `edit_tokens_own`: Required if you want to create tokens for yourself.
- `edit_tokens_all`: Required if you want to create tokens for any user on the instance.

Refer to Define roles on the Splunk platform with capabilities for more information.

Spotify credentials

You can use these credentials to authenticate the following nodes:

- Spotify

Supported authentication methods

- OAuth2

Related resources

Refer to Spotify's Web API documentation for more information about the service.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you're self-hosting n8n, you'll need a Spotify Developer account so you can create a Spotify app:

1. Open the Spotify developer dashboard.
2. Select **Create an app**.
3. Enter an **App name**, like `n8n integration`.
4. Enter an **App description**.
5. Copy the **OAuth Redirect URL** from n8n and enter it as the **Redirect URI** in your Spotify app.
6. Check the box to agree to the Spotify Terms of Service and Branding Guidelines.
7. Select **Create**. The **App overview** page opens.
8. Copy the **Client ID** and enter it in your n8n credential.
9. Copy the **Client Secret** and enter it in your n8n credential.
10. Select **Connect my account** and follow the on-screen prompts to finish authorizing the credential.

Refer to Spotify Apps for more information.

SSH credentials

You can use these credentials to authenticate the following nodes:

- SSH

Prerequisites

- Create a remote server with SSH enabled.
- Create a user account that can `ssh` into the server using one of the following:
  - Their own password
  - An SSH private key

Supported authentication methods

- Password: Use this method if you have a user account that can `ssh` into the server using their own password.
- Private key: Use this method if you have a user account that uses an SSH key for the server or service.

Related resources

Secure Shell (SSH) protocol is a method for securely sending commands over a network. Refer to Connecting to GitHub with SSH for an example of SSH setup.

Using password

Use this method if you have a user account that can `ssh` into the server using their own password.

To configure this credential, you'll need to:

1. Enter the IP address of the server you're connecting to as the **Host**.
2. Enter the **Port** to use for the connection. SSH uses port `22` by default.
3. Enter the **Username** for a user account with `ssh` access on the server.
4. Enter the **Password** for that user account.

Using private key

Use this method if you have a user account that uses an SSH key for the server or service.

To configure this credential, you'll need to:

1. Enter the IP address of the server you're connecting to as the **Host**.
2. Enter the **Port** to use for the connection. SSH uses port `22` by default.
3. Enter the **Username** of the account that generated the private key.
4. Enter the entire contents of your SSH **Private Key**.
5. If you created a **Passphrase** for the **Private Key**, enter the passphrase.
    - If you didn't create a passphrase for the key, leave blank.

Stackby credentials

You can use these credentials to authenticate the following nodes:

- Stackby

Prerequisites

Create a Stackby account.

Supported authentication methods

- API key

Related resources

Refer to Stackby's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Go to your **Account Settings > API** to create an API Key. Refer to API Key for more information.

Storyblok credentials

You can use these credentials to authenticate the following nodes:

- Storyblok

Prerequisites

Create a Storyblok account.

Supported authentication methods

- Content API key: For read-only access
- Management API key: For full CRUD operations

Related resources

Refer to Storyblok's Content v1 API documentation and Management API documentation for more information about the services.

Using Content API key

To configure this credential, you'll need:

- A Content **API Key**: Go to your Storyblok workspace's **Settings > Access Tokens** to get an API key. Choose an **Access Level** of either **Public** (`version=published`) or **Preview** (`version-published` and `version=draft`). Enter this access token as your **API Key**. Refer to How to retrieve and generate access tokens for more detailed instructions.

Refer to Content v1 API Authentication for more information about supported operations with each Access Level.

Using Management API key

To configure this credential, you'll need:

- A **Personal Access Token**: Go to **My Account > Personal access tokens** to generate a new access token. Enter this access token as your **Personal Access Token**.

Strapi credentials

You can use these credentials to authenticate the following nodes:

- Strapi

Prerequisites

Create a Strapi admin account with:

- Access to an existing Strapi project.
- At least one collection type within that project.
- Published data within that collection type.

Refer to the Strapi developer Quick Start Guide for more information.

Supported authentication methods

- API user account: Requires a user account with appropriate content permissions.
- API token: Requires an admin account.

Related resources

Refer to Strapi's documentation for more information about the service.

Using API user account

To configure this credential, you'll need:

- A user **Email**: Must be for a user account, not an admin account. Refer to the more detailed instructions below.
- A user **Password**: Must be for a user account, not an admin account. Refer to the more detailed instructions below.
- The **URL**: Use the public URL of your Strapi server, defined in `./config/server.js` as the `url` parameter. Strapi recommends using an absolute URL.
  - For Strapi Cloud projects, use the URL of your Cloud project, for example: `https://my-strapi-project-name.strapiapp.com`
- The **API Version**: Select the version of the API you want your calls to use. Options include:
  - **Version 3**
  - **Version 4**

In Strapi, the configuration involves two steps:

1. Configure a role.
2. Create a user account.

Refer to the more detailed instructions below for each step.

Configure a role

For API access, use the Users & Permissions Plugin in **Settings > Users & Permissions Plugin**.

Refer to Configuring Users & Permissions Plugin for more information on the plugin. Refer to Configuring end-user roles for more information on roles.

For the n8n credential, the user must have a role that grants them API permissions on the collection type. For the role, you can either:

- Update the default **Authenticated** role to include the permissions and assign the user to that role. Refer to Configuring role's permissions for more information.
- Create a new role to include the permissions and assign the user to that role. Refer to Creating a new role for more information.

For either option, once you open the role:

1. Go to the **Permissions** section.
2. Open the section for the relevant collection type.
3. Select the permissions for the collection type that the role should have. Options include:
   - `create` (POST)
   - `find` and `findone` (GET)
   - `update` (PUT)
   - `delete` (DELETE)
4. Repeat for all relevant collection types.
5. Save the role.

Refer to Endpoints for more information on the permission options.

Create a user account

Now that you have an appropriate role, create an end-user account and assign the role to it:

1. Go to **Content Manager > Collection Types > User**.
2. Select **Add new entry**.
3. Fill in the user details. The n8n credential requires these fields, though your Strapi project may have more custom required fields:
   - **Username**: Required for all Strapi users.
   - **Email**: Enter in Strapi and use as the **Email** in the n8n credential.
   - **Password**: Enter in Strapi and use as the **Password** in the n8n credential.
   - **Role**: Select the role you set up in the previous step.

Refer to Managing end-user accounts for more information.

Using API token

To configure this credential, you'll need:

- An **API Token**: Create an API token from **Settings > Global Settings > API Tokens**. Refer to Strapi's Creating a new API token documentation for more details and information on regenerating API tokens.

- The **URL**: Use the public URL of your Strapi server, defined in `./config/server.js` as the `url` parameter. Strapi recommends using an absolute URL.

  - For Strapi Cloud projects, use the URL of your Cloud project, for example: `https://my-strapi-project-name.strapiapp.com`

- The **API Version**: Select the version of the API you want your calls to use. Options include:
  - **Version 3**
  - **Version 4**

Strava credentials

You can use these credentials to authenticate the following nodes:

- Strava
- Strava Trigger

Prerequisites

- Create a Strava account.
- Create a Strava application in **Settings > API**. Refer to Using OAuth2 for more information.

Supported authentication methods

- OAuth2

Related resources

Refer to Strava's API documentation for more information about the service.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated when you create a Strava app.
- A **Client Secret**: Generated when you create a Strava app.

Use these settings for your Strava app:

- In n8n, copy the **OAuth Callback URL**. Paste this URL into your Strava app's **Authorization Callback Domain**.
- Remove the protocol (`https://` or `http://`) and the relative URL (`/oauth2/callback` or `/rest/oauth2-credential/callback`) from the **Authorization Callback Domain**. For example, if the OAuth Redirect URL was originally `https://oauth.n8n.cloud/oauth2/callback`, the **Authorization Callback Domain** would be `oauth.n8n.cloud`.
- Copy the **Client ID** and **Client Secret** from your app and add them to your n8n credential.

Refer to Authentication for more information about Strava's OAuth flow.

Stripe credentials

You can use these credentials to authenticate the following nodes:

- Stripe Trigger
- Stripe

Supported authentication methods

- API key

Related resources

Refer to Stripe's API documentation for more information about the service.

Using API key

To configure this credential, you'll need a Stripe admin or developer account and:

- An API **Secret Key**

Before you generate an API key, decide whether to generate it in live mode or test mode. Refer to Test mode and live mode for more information about the two modes.

Live mode Secret key

To generate a Secret key in live mode:

1. Open the Stripe developer dashboard and select **API Keys**.
2. In the **Standard Keys** section, select **Create secret key**.
3. Enter a **Key name**, like `n8n integration`.
4. Select **Create**. The new API key displays.
5. Copy the key and enter it in your n8n credential as the **Secret Key**.

Refer to Stripe's Create a secret API key for more information.

Test mode Secret key

To use a Secret key in test mode, you must copy the existing one:

1. Go to your Stripe test mode developer dashboard and select **API Keys**.
2. In the **Standard Keys** section, select **Reveal test key** for the **Secret key**.
3. Copy the key and enter it in your n8n credential as the **Secret Key**.

Refer to Stripe's Create a secret API key for more information.

Test mode and live mode

All Stripe API requests happen within either test mode or live mode. Each mode has its own API key.

Use test mode to access simulated test data and live mode to access actual account data. Objects in one mode aren't accessible to the other.

Refer to API keys | Test mode versus live mode for more information about what's available in each mode and guidance on when to use each.

Key prefixes

Stripes' Secret keys always begin with `sk_`:

- Live keys begin with `sk_live_`.
- Test keys begin with `sk_test_`.

n8n hasn't tested these credentials with Restricted keys (prefixed `rk_`).

Supabase credentials

You can use these credentials to authenticate the following nodes:

- Supabase
- Supabase Vector Store

Prerequisites

Create a Supabase account.

Supported authentication methods

- API key

Related resources

Refer to Supabase's API documentation for more information about the service.

Using access token

To configure this credential, you'll need:

- A **Host**
- A **Service Role Secret**

To generate your API Key:

1. In your Supabase account, go to the **Dashboard** and create or select a project for which you want to create an API key.
2. Go to **Project Settings > API** to see the API Settings for your project.
3. Copy the **URL** from the **Project URL** section and enter it as your n8n **Host**. Refer to API URL and keys for more detailed instruction.
4. Reveal and copy the **Project API key** for the `service_role`. Copy that key and enter it as your n8n **Service Role Secret**. Refer to Understanding API Keys for more information on the `service_role` privileges.

SurveyMonkey credentials

You can use these credentials to authenticate the following nodes:

- SurveyMonkey Trigger

Prerequisites

- Create a SurveyMonkey account.
- Register an app from your **Developer dashboard > My apps**.
  - Refer to Required app scopes for information on the scopes you must use.

Supported authentication methods

- API access token
- OAuth2

Related resources

Refer to SurveyMonkey's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- An **Access Token**: Generated once you create an app.
- A **Client ID**: Generated once you create an app.
- A **Client Secret**: Generated once you create an app.

Once you've created your app and assigned appropriate scopes, go to **Settings > Credentials**. Copy the **Access Token**, **Client ID**, and **Secret** and add them to n8n.

Using OAuth

To configure this credential, you'll need:

- A **Client ID**: Generated once you create an app.
- A **Client Secret**: Generated once you create an app.

Once you've created your app and assigned appropriate scopes:

1. Go to the app's **Settings > Settings**.
2. From n8n, copy the **OAuth Redirect URL**.
3. Overwrite the app's existing **OAuth Redirect URL** with that URL.
4. Select **Submit Changes**.
5. Be sure the **Scopes** section contains the Required app scopes.

From the app's **Settings > Credentials**, copy the **Client ID** and **Client Secret** and add them to your n8n credential. You can now select **Connect my account** from n8n.

Required app scopes

Once you create your app, go to **Settings > Scopes**. Select these scopes for your n8n credential to work:

- **View Surveys**
- **View Collectors**
- **View Responses**
- **View Response Details**
- **Create/Modify Webhooks**
- **View Webhooks**

Select **Update Scopes** to save them.

SyncroMSP credentials

You can use these credentials to authenticate the following nodes:

- SyncroMSP

Prerequisites

Create a SyncroMSP account.

Supported authentication methods

- API key

Related resources

Refer to SyncroMSP's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Called an **API token** in SyncroMSP. To create an API token, go to your **user menu > Profile/Password > API Tokens** and select the option to **Create New Token**. Select **Custom Permissions** to enter a name for your token and adjust the permissions to match your requirements.
- Your **Subdomain**: Enter your SyncroMSP subdomain. This is visible in the URL of your SyncroMSP, located between `https://` and `.syncromsp.com`. If your full URL is `https://n8n-instance.syncromsp.com`, you'd enter `n8n-instance` as the subdomain.

Refer to API Tokens for more information on creating new tokens.

Sysdig Management credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Sysdig account or configure a local instance.

Supported authentication methods

- Access Key

Related resources

Refer to Sysdig's documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more.

Using API access key

To configure this credential, you'll need:

- An **Access Key**

Refer to the Sysdig Agent Access Keys documentation for instructions on obtaining the Access Key from the application.

Taiga credentials

You can use these credentials to authenticate the following nodes:

- Taiga
- Taiga Trigger

Prerequisites

Create a Taiga account.

Supported authentication methods

- Basic auth

Related resources

Refer to Taiga's API documentation for more information about the service.

Using basic auth

To configure this credential, you'll need:

- A **Username**: Enter your username or user email address. Refer to Normal login for more information.
- A **Password**: Enter your password.
- The **Environment**: Choose between **Cloud** or **Self-Hosted**. For **Self-Hosted** instances, you'll also need to add:
  - The **URL**: Enter your Taiga URL.

Tapfiliate credentials

You can use these credentials to authenticate the following nodes:

- Tapfiliate

Prerequisites

Create a Tapfiliate account.

Supported authentication methods

- API key

Related resources

Refer to Tapfiliate's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Get your API Key from your **Profile Settings > API**

Key.

Refer to Your API key for more information.

Telegram credentials

You can use these credentials to authenticate the following nodes:

- Telegram
- Telegram Trigger

Prerequisites

Create a Telegram account.

Supported authentication methods

- API bot access token

Related resources

Refer to Telegram's Bot API documentation for more information about the service.

Refer to the Telegram Bot Features documentation for more information on creating and working with bots.

Using API bot access token

To configure this credential, you'll need:

- A bot **Access Token**

To generate your access token:

1. Start a chat with the BotFather.
2. Enter the `/newbot` command to create a new bot.
3. The BotFather will ask you for a name and username for your new bot:
   - The **name** is the bot's name displayed in contact details and elsewhere. You can change the bot name later.
   - The **username** is a short name used in search, mentions, and t.me links. Use these guidelines when creating your username:
     - Must be between 5 and 32 characters long.
     - Not case sensitive.
     - May only include Latin characters, numbers, and underscores.
     - Must end in `bot`, like `tetris_bot` or `TetrisBot`.
     - You can't change the username later.
4. Copy the bot **token** the BotFather generates and add it as the **Access Token** in n8n.

Refer to the BotFather Create a new bot documentation for more information.

TheHive credentials

You can use these credentials to authenticate the following nodes:

- TheHive

Prerequisites

Install TheHive on your server.

Supported authentication methods

- API key

Related resources

Refer to TheHive 3's API documentation and TheHive 4's API documentation for more information about the services.

Using API key

To configure this credential, you'll need:

- An **API Key**: Create an API key from **Organization > Create API Key**. Refer to API Authentication for more information.
- Your **URL**: The URL of your TheHive server.
- An **API Version**: Choose between:
  - **TheHive 3 (api v0)**
  - **TheHive 4 (api v1)**
  - For TheHive 5, use TheHive 5 credentials instead.
- **Ignore SSL Issues**: When turned on, n8n will connect even if SSL certificate validation fails.

TheHive 5 credentials

You can use these credentials to authenticate the following nodes with TheHive 5.

- TheHive 5

Prerequisites

Install TheHive 5 on your server.

Supported authentication methods

- API key

Related resources

Refer to TheHive's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Users with `orgAdmin` and `superAdmin` accounts can generate API keys:
  - `orgAdmin` account: Go to **Organization > Create API Key** for the user you wish to generate a key for.
  - `superAdmin` account: Go to **Users > Create API Key** for the user you wish to generate a key for.
  - Refer to API Authentication for more information.
- A **URL**: The URL of your TheHive server.
- **Ignore SSL Issues**: When turned on, n8n will connect even if SSL certificate validation fails.

TimescaleDB credentials

You can use these credentials to authenticate the following nodes:

- TimescaleDB

Prerequisites

An available instance of TimescaleDB.

Supported authentication methods

- Database connection

Related resources

Refer to the Timescale documentation for more information about the service.

Using database connection

To configure this credential, you'll need:

- The **Host**: The fully qualified server name or IP address of your TimescaleDB server.

- The **Database**: The name of the database to connect to.
- A **User**: The user name you want to log in with.
- A **Password**: Enter the password for the database user you are connecting to.
- **Ignore SSL Issues**: If turned on, n8n will connect even if SSL certificate validation fails and you won't see the **SSL** selector.
- **SSL**: This setting controls the `ssl-mode` connection string for the connection. Options include:
  - **Allow**: Sets the `ssl-mode` parameter to `allow`. First try a non-SSL connection; if that fails, try an SSL connection.
  - **Disable**: Sets the `ssl-mode` parameter to `disable`. Only try a non-SSL connection.
  - **Require**: Sets the `ssl-mode` parameter to `require`, which is the default for TimescaleDB connection strings. Only try an SSL connection. If a root CA file is present, verify that a trusted certificate authority (CA) issued the server certificate.
- **Port**: The port number of the TimescaleDB server.

Refer to the Timescale connection settings documentation for more information about the non-SSL fields. Refer to Connect with a stricter SSL for more information about the SSL options.

Todoist credentials

You can use these credentials to authenticate the following nodes:

- Todoist

Supported authentication methods

- API key
- OAuth2

Related resources

Refer to Todoist's REST API documentation for more information about the service.

Using API key

To configure this credential, you'll need a Todoist account and:

- An **API Key**

To get your **API Key**:

1. In Todoist, open your **Integration settings**.
2. Select the **Developer** tab.
3. Copy your **API token** and enter it as the **API Key** in your n8n credential.

Refer to Find your API token for more information.

Using OAuth2

–8<– “_snippets/integrations/builtin/credentials/cloud-oauth-button.md”

If you’re self-hosting n8n, you’ll need a Todoist account and:

- A **Client ID**
- A **Client Secret**

Get both by creating an application:

1. Open the Todoist App Management Console.
2. Select **Create a new app**.
3. Enter an **App name** for your app, like `n8n integration`.
4. Select **Create app**.
5. Copy the n8n **OAuth Redirect URL** and enter it as the **OAuth redirect URL** in Todoist.
6. Copy the **Client ID** from Todoist and enter it in your n8n credential.
7. Copy the **Client Secret** from Todoist and enter it in your n8n credential.
8. Configure the rest of your Todoist app as it makes sense for your use case.

Refer to the Todoist Authorization Guide for more information.

Toggl credentials

You can use these credentials to authenticate the following nodes:

- Toggl Trigger

Prerequisites

Create a Toggl account.

Supported authentication methods

- Basic auth

Related resources

Refer to Toggl’s API documentation for more information about the service.

Using basic auth

To configure this credential, you’ll need:

- A **Username**: Enter your user email address.
- A **Password**: Enter your user password.

Refer to Authentication for more information.

TOTP credentials

You can use these credentials to authenticate the following nodes:

- TOTP

Prerequisites

Generate a TOTP **Secret** and **Label**.

Supported authentication methods

- Secret and label

Related resources

Time-based One-time Password (TOTP) is an algorithm that generates a one-time password (OTP) using the current time. Refer to Google Authenticator | Key URI format for more information.

Using secret and label

To configure this credential, you'll need:

- A **Secret**: The secret key encoded in the QR code during authenticator setup. It's an arbitrary key value encoded in Base32, for example: `BVDRSBXQB2ZEL5HE`. Refer to Google Authenticator Secret for more information.
- A **Label**: The identifier for the account. It contains an account name as a URI-encoded string. You can include prefixes to identify the provider or service managing the account. If you use prefixes, use a literal or url-encoded colon to separate the issuer prefix and the account name, for example: `GitHub:john-doe`. Refer to Google Authenticator Label for more information.

Travis CI credentials

You can use these credentials to authenticate the following nodes:

- Travis CI

Prerequisites

Create a Travis CI account.

Supported authentication methods

- API token

Related resources

Refer to Travis CI's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- An **API Token**: Get your API token from **Account Settings > API Token** or generate one through the Travis CI command line client .

Trellix ePO credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Trellix ePolicy Orchestrator account.

Supported authentication methods

- Basic auth

Related resources

Refer to Trellix ePO's documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using basic auth

To configure this credential, you'll need:

- A **Username** to connect as.
- A **Password** for that user account.

n8n uses These fields to build the `-u` parameter in the format of `-u username:pw`. Refer to Web API basics for more information.

Trello credentials

You can use these credentials to authenticate the following nodes:

- Trello
- Trello Trigger

Supported authentication methods

- API key

Related resources

Refer to Trello's API documentation for more information about the service.

Using API key

To configure this credential, you'll need a Trello account and:

- An **API Key**
- An **API Token**

To generate both the API Key and API Token, create a Trello Power-Up:

1. Open the Trello Power-Up Admin Portal.
2. Select **New**.
3. Enter a **Name** for your Power-Up, like `n8n integration`.
4. Select the **Workspace** the Power-Up should have access to.
5. Leave the **iframe connector URL** blank.
6. Enter appropriate contact information.
7. Select **Create**.
8. This should open the Power-Up to the **API Key** page. (If it doesn't, open that page.)
9. Select **Generate a new API Key**.
10. Copy the **API key** from Trello and enter it in your n8n credential.
11. In your Trello API key page, enter your n8n base URL as an **Allowed origin**.
12. In **Capabilities** make sure to select the necessary options.
13. Select the **Token** link next to your Trello **API Key**.
14. When prompted, select **Allow** to grant all the permissions asked for.
15. Copy the Trello **Token** and enter it as the n8n **API Token**.

Refer to Trello's API Introduction for more information on API keys and tokens. Refer to Trello's Power-Up Admin Portal for more information on creating Power-Ups.

Twake credentials

You can use these credentials to authenticate the following nodes:

- Twake

Prerequisites

Create a Twake account.

Supported authentication methods

- Cloud API key
- Server API key

Related resources

Refer to Twake's documentation for more information about the service.

Using Cloud API key

To configure this credential, you'll need:

- A **Workspace Key**: Generated when you install the **n8n** application to your Twake Cloud environment and select **Configure**. Refer to How to connect n8n to Twake for more detailed instructions.

Using Server API key

To configure this credential, you'll need:

- A **Host URL**: The URL of your Twake self-hosted instance.
- A **Public ID**: Generated when you create an app.
- A **Private API Key**: Generated when you create an app.

To generate your **Public ID** and **Private API Key**, create a Twake application:

1. Go to **Workspace Settings > Applications and connectors > Access your applications and connectors > Create an application**.
2. Enter appropriate details.
3. Once you've created your app, view its **API Details**.
4. Copy the **Public identifier** and add it as the n8n **Public ID**.
5. Copy the **Private key** and add it as the n8n **Private API Key**.

Refer to API settings for more information.

Twilio credentials

You can use these credentials to authenticate the following nodes:

- Twilio

- Twilio trigger

Supported authentication methods

- **Auth token**: Twilio recommends this method for local testing only.
- **API key**: Twilio recommends this method for production.

Related resources

Refer to Twilio's API documentation for more information about the service.

Using Auth Token

To configure this credential, you'll need a Twilio account and:

- Your Twilio **Account SID**
- Your Twilio **Auth Token**

To set up the credential:

1. In n8n, select **Auth Token** as the **Auth Type**.
2. In Twilio, go to **Console Dashboard > Account Info**.
3. Copy your **Account SID** and enter this in your n8n credential. This acts as a username.
4. Cop your **Auth Token** and enter this in your n8n credential. This acts as a password.

Refer to Auth Tokens and How to Change Them for more information.

Using API key

To configure this credential, you'll need a Twilio account and:

- Your Twilio **Account SID**
- An **API Key SID**: Generated when you create an API key.
- An **API Key Secret**: Generated when you create an API key.

To set up the credential:

1. In n8n, select **API Key** as the **Auth Type**.
2. In Twilio, go to **Console Dashboard > Account Info**.
3. Copy your **Account SID** and enter it in your n8n credential.
4. In Twilio, go to your account's **API keys & tokens** page.
5. Select **Create API Key**.
6. Enter a **Friendly name** for your API key, like `n8n integration`.
7. Select your **Key type**. n8n works with either **Main** or **Standard**. Refer to Selecting an API key type for more information.
8. Select **Create API Key** to finish creating the key.
9. On the **Copy secret key** page, copy the **SID** displayed with the key and enter it in your n8n credential **API Key SID**.
10. On the **Copy secret key** page, copy the **Secret** displayed with the key and enter it in your n8n credential **API Key Secret**.

Refer to Create an API key for more detailed instructions.

Selecting an API key type

When you create a Twilio API key, you must select a key type. The n8n credential works with **Main** and **Standard** key types.

Here are more details on the different API key types:

- **Main**: This key type gives you the same level of access as using your Account SID and Auth Token in API requests.
- **Standard**: This key type gives you access to all the functionality in Twilio's APIs except the API key resources and Account resources.
- **Restricted**: This key type is in beta. n8n hasn't tested the credential against this key type; if you try it, let us know if you run into any issues.

Refer to Types of API keys for more information on the key types.

Twist credentials

You can use these credentials to authenticate the following nodes:

- Twist

Prerequisites

- Create a Twist account.
- Create a general integration and configure a valid OAuth Redirect URL. Refer to Using OAuth2 for more information.

Supported authentication methods

- OAuth2

Related resources

Refer to Twist's API documentation for more information about authenticating with the service.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated once you create a general integration.
- A **Client Secret**: Generated once you create a general integration.

To generate your Client ID and Client Secret, create a general integration.

Use these settings for your integration's **OAuth Authentication**:

- Copy the **OAuth Redirect URL** from n8n and enter it as the **OAuth 2 redirect URL** in Twist.

- Select **Update OAuth settings** to save those changes.
- Copy the **Client ID** and **Client Secret** from Twist and enter them in the appropriate fields in n8n.

Local environment redirect URL

Twist doesn't accept a localhost callback URL. These steps should allow you to configure the OAuth credentials for the local environment:

1. Use ngrok to expose the local server running on port `5678` to the internet. In your terminal, run the following command:

   ```
   ngrok http 5678
   ```

2. Run the following command in a new terminal. Replace `<YOUR-NGROK-URL>` with the URL that you get from the previous step.

   ```
   export WEBHOOK_URL=<YOUR-NGROK-URL>
   ```

3. Use the generated URL as your **OAuth 2 redirect URL** in Twist.

Typeform credentials

You can use these credentials to authenticate the following nodes:

- Typeform Trigger

Supported authentication methods

- API token
- OAuth2

Related resources

Refer to Typeform's API documentation for more information about the service.

Using API token

To configure this credential, you'll need a Typeform account and:

- A personal **Access Token**

To get your personal access token:

1. Log into your Typeform account.
2. Select your profile avatar in the upper right and go to **Account > Your settings > Personal Tokens**.
3. Select **Generate a new token**.
4. Give your token a **Name**, like `n8n integration`.
5. For **Scopes**, select **Custom scopes**. Select these scopes:
   - Forms: Read

    - Webhooks: Read, Write
6. Select **Generate token**.
7. Copy the token and enter it in your n8n credential.

Refer to Typeform's Personal access token documentation for more information.

Using OAuth2

To configure this credential, you'll need a Typeform account and:

- A **Client ID**: Generated when you register an app.
- A **Client Secret**: Generated when you register an app.

To get your Client ID and Client Secret, register a new Typeform app:

1. Log into your Typeform account.
2. In the upper left, select the dropdown for your organization and select **Developer apps**.
3. Select **Register a new app**.
4. Enter an **App Name** that makes sense, like `n8n OAuth2 integration`.
5. Enter your n8n base URL as the **App website**, for example `https://n8n-sample.app.n8n.cloud/`.
6. From n8n, copy the **OAuth Redirect URL**. Enter this in Typeform as the **Redirect URI(s)**.
7. Select **Register app**.
8. Copy the **Client Secret** and enter it in your n8n credential.
9. In Typeform, select **Got it** to close the Client Secret modal.
10. The **Developer apps** panel displays your new app. Copy the **Client ID** and enter it in your n8n credential.
11. Once you enter both the **Client ID** and **Client Secret** in n8n, select **Connect my account** and follow the on-screen prompts to finish authorizing the app.

Refer to Create applications that integrate with Typeform's APIs for more information.

Unleashed Software credentials

You can use these credentials to authenticate the following nodes:

- Unleashed Software

Prerequisites

Create an Unleashed Software account.

Supported authentication methods

- API key

Related resources

Refer to Unleashed's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API ID**: Go to **Integrations > Unleashed API Access** to find your **API ID**.
- An **API Key**: Go to **Integrations > Unleashed API Access** to find your **API Key**.

Refer to Unleashed API Access for more information.

UpLead credentials

You can use these credentials to authenticate the following nodes:

- UpLead

Prerequisites

Create an UpLead account.

Supported authentication methods

- API key

Related resources

Refer to UpLead's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Go to your **Account > Profiles** to **Generate New API Key**.

---

uProc credentials

You can use these credentials to authenticate the following nodes:

- uProc

Prerequisites

Create a uProc account.

Supported authentication methods

- API key

Related resources

Refer to uProc's API documentation for more information about the service.

Using API Key

To configure this credential, you'll need:

- An **Email** address: Enter the email address you use to log in to uProc. This is also displayed in **Settings > Integrations > API Credentials**.
- An **API Key**: Go to **Settings > Integrations > API Credentials**. Copy the **API Key (real)** from the **API Credentials** section and enter it in your n8n credential.

UptimeRobot credentials

You can use these credentials to authenticate the following nodes:

- UptimeRobot

Prerequisites

Create an UptimeRobot account.

Supported authentication methods

- API key

Related resources

Refer to UptimeRobot's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Get your API Key from **My Settings > API Settings**. Create a **Main API Key** and enter this key in your n8n credential.

API key types

UptimeRobot supports three API key types:

- **Account-specific** (also known as **main**): Pulls data for multiple monitors.
- **Monitor-specific**: Pulls data for a single monitor.
- **Read-only**: Only runs `GET` API calls.

To complete all of the operations in the UptimeRobot node, use the **Main** or **Account-specific** API key type. Refer to API authentication for more information.

---

urlscan.io credentials

You can use these credentials to authenticate the following nodes:

- urlscan.io

Prerequisites

Create an urlscan.io account.

Supported authentication methods

- API key

Related resources

Refer to urlscan.io's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Get your API key from **Settings & API > API Keys**.

Venafi TLS Protect Cloud credentials

You can use these credentials to authenticate the following nodes:

- Venafi TLS Protect Cloud node
- Venafi TLS Protect Cloud Trigger node

Prerequisites

Create a Venafi TLS Protect Cloud account.

Supported authentication methods

- API key

Related resources

Refer to Venafi TLS Protect Cloud's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- A **Region**: Select the region that matches your business needs. Choose **EU** if you're located in the European Union. Otherwise, choose **US**.
- An **API Key**: Go to your **avatar > Preferences > API Keys** to get your API key. You can also use VCert to get your API key. Refer to Obtaining an API Key for more information.

Venafi TLS Protect Datacenter credentials

You can use these credentials to authenticate the following nodes:

- Venafi TLS Protect Datacenter node

Prerequisites

- Create a Venafi TLS Protect Datacenter account.
- Set the expiration and refresh time for tokens. Refer to Setting up token authentication for more information.
- Create an API integration in **API > Integrations**. Refer to Integrating other systems with Venafi products for detailed instructions.
  - Take note of the Client ID for your integration.
  - Choose the scopes needed for the operations you want to perform within n8n. Refer to the scopes table in Integrating other systems with Venafi products for more details on available scopes.

Supported authentication methods

- API integration

Related resources

Refer to Venafi's API integration documentation for more information about the service.

Using API integration

To configure this credential, you'll need:

- A **Domain**: Enter your Venafi TLS Protect Datacenter domain.
- A **Client ID**: Enter the **Client ID** from your API integration. Refer to the information and links in Prerequisites for more information on creating an API integration.
- A **Username**: Enter your username.
- A **Password**: Enter your password.
- **Allow Self-Signed Certificates**: If turned on, the credential will allow self-signed certificates.

Vercel AI Gateway credentials

You can use these credentials to authenticate the following nodes:

- Chat Vercel AI Gateway

Prerequisites

Create a Vercel account.

Supported authentication methods

- API key
- OIDC token

Related resources

Refer to the Vercel AI Gateway documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**

To generate your API Key:

1. Login to Vercel or create an account.
2. Go to the Vercel dashboard and select the **AI Gateway** tab.
3. Select **API keys** on the left side bar.
4. Select **Add key** and proceed with **Create key** from the Dialog.
5. Copy your key and add it as the **API Key** in n8n.

Using OIDC token

To configure this credential, you'll need:

- An **OIDC token**

To generate your OIDC token:

1. In local development, link your application to a Vercel project with the `vc link` command.
2. Run the `vercel env pull` command to pull the environment variables from Vercel.
3. Copy your token and add it as the **OIDC TOKEN** in n8n.

Vero credentials

You can use these credentials to authenticate the following nodes:

- Vero

Prerequisites

Create a Vero account.

Supported authentication methods

- API auth token

Related resources

Refer to Vero's API documentation for more information about the service.

Using API auth token

To configure this credential, you'll need:

- An **Auth Token**: Get your auth token from your Vero account settings. Refer to API authentication for more information.

VirusTotal credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a VirusTotal account.

Supported authentication methods

- API key

Related resources

Refer to VirusTotal's documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API key

To configure this credential, you'll need:

- An **API Token**: Go to your **user account menu > API key** to get your API key. Enter this as the **API Token** in your n8n credential. Refer to API authentication for more information.

Vonage credentials

You can use these credentials to authenticate the following nodes:

- Vonage

Prerequisites

Create a Vonage developer account.

Supported authentication methods

- API key

Related resources

Refer to Vonage's SMS API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**
- An **API Secret**

Get your **API Key** and **API Secret** from your developer dashboard **user account > Settings > API Settings**. Refer to Retrieve your account information for more information.

Weaviate credentials

You can use these credentials to authenticate the following nodes:

- Weaviate Vector Store

Supported authentication methods

- API key

Related resources

Refer to Weaviate's connection documentationfor more information on how to connect to Weaviate.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

Connection type: Weaviate Cloud

Create your Weaviate Cloud Database and follow these instructions get the following parameter values from your Weaviate Cloud Database:

- **Weaviate Cloud Endpoint**
- **Weaviate Api Key**

Note: Weaviate provides a free sandbox option for testing.

Connection type: Custom Connection

For this Connection Type, you need to deploy Weaviate on your own server, configured so n8n can access it. Refer to Weaviate's authentication documentation for information on creating and using API keys.

You can then provide the arguments for your custom connection:

- **Weaviate Api Key**: Your Weaviate API key.
- **Custom Connection HTTP Host**: The domain name or IP address of your Weaviate instance to use for HTTP API calls.
- **Custom Connection HTTP Port**: The port your Weaviate instance is running on for HTTP API calls. By default, this is 8080.
- **Custom Connection HTTP Secure**: Whether to connect to the Weaviate through HTTPS for HTTP API calls.
- **Custom Connection gRPC Host**: The hostname or IP address of your Weaviate instance to use for gRPC.
- **Custom Connection gRPC Port**: The gRPC API port for your Weaviate instance. By default, this is 50051.
- **Custom Connection gRPC Secure**: Whether to connect to the Weaviate through HTTPS for gRPC.

For community support, refer to Weaviate Forums.

Webex by Cisco credentials

You can use these credentials to authenticate the following nodes:

- Webex by Cisco
- Webex by Cisco Trigger

Prerequisites

Create a Webex by Cisco account (this should automatically get you developer account access).

Supported authentication methods

- OAuth2

Related resources

Refer to Webex's API documentation for more information about the service.

Using OAuth2

Should you need to configure OAuth2 from scratch, you'll need to create an integration to use this credential. Refer to the instructions in the Webex Registering your Integration documentation to begin.

n8n recommends using the following **Scopes** for your integration:

- `spark:rooms_read`
- `spark:messages_write`
- `spark:messages_read`
- `spark:memberships_read`
- `spark:memberships_write`
- `meeting:recordings_write`
- `meeting:recordings_read`
- `meeting:preferences_read`
- `meeting:schedules_write`
- `meeting:schedules_read`

Webflow credentials

You can use these credentials to authenticate the following nodes:

- Webflow
- Webflow Trigger

Prerequisites

- Create a Webflow account.
- Create a site: Required for API access token authentication only.

Supported authentication methods

- API access token
- OAuth2

Related resources

Refer to Webflow's API documentation for more information about the service.

Using API access token

To configure this credential, you'll need:

- A Site **Access Token**: Access tokens are site-specific. Go to your site's **Site Settings > Apps & integrations > API access** and select **Generate API token**. Refer to Get a Site Token for more information.

Using OAuth2

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you need to configure OAuth2 from scratch, register an application in your workspace.

Use these settings for your application:

- Copy the **OAuth callback URL** from n8n and add it as a **Redirect URI** in your application.
- Once you've created your application, copy the **Client ID** and **Client Secret** and enter them in your n8n credential.
- If you are using the Webflow Data API V1 (deprecated), enable the **Legacy** toggle. Otherwise, leave this inactive.

Refer to OAuth for more information on Webflow's OAuth web flow.

Webhook credentials

You can use these credentials to authenticate the following nodes:

- Webhook

Prerequisites

You must use the authentication method required by the app or service you want to query.

Supported authentication methods

- Basic auth
- Header auth
- JWT auth
- None

–8<– “_snippets/integrations/builtin/credentials/generic-auth/basic-auth.md”

–8<– “_snippets/integrations/builtin/credentials/generic-auth/header-auth.md”

Using JWT auth

JWT Auth is a method of authentication that uses JSON Web Tokens (JWT) to digitally sign data. This authentication method uses the **JWT credential** and can use either a **Passphrase** or **PEM Key** as key type. Refer to JWT credential for more information.

Wekan credentials

You can use these credentials to authenticate the following nodes:

- Wekan

Prerequisites

Install Wekan on your server.

Supported authentication methods

- Basic auth

Related resources

Refer to Wekan’s API documentation for more information about authenticating with the service.

Using basic auth

To configure this credential, you’ll need:

- A **Username**: Enter your Wekan username.
- A **Password**: Enter your Wekan password.
- A **URL**: Enter your Wekan domain.

WhatsApp Business Cloud credentials

You can use these credentials to authenticate the following nodes:

- WhatsApp Business Cloud
- WhatsApp Trigger

Requirements

To create credentials for WhatsApp, you need the following Meta assets:

- A Meta developer account: A developer account allows you to create and manage Meta apps, including WhatsApp integrations. ??? note “Set up a Meta developer account”
  1. Visit the Facebook Developers site.
  2. Click **Getting Started** in the upper-right corner (if the link says **My Apps**, you’ve already set up a developer account).
  3. Agree to terms and conditions.
  4. Provide a phone number for verification.
  5. Select your occupation or role.
- A Meta business portfolio: WhatsApp messaging services require a Meta business portfolio, formerly called a Business Manager account. The UI may still show either option. ??? note “Set up a Meta business portfolio”
  1. Visit the Facebook Business site.
  2. Select **Create an account**.
     - If you already have a Facebook Business account and portfolio, but want a new portfolio, open the business portfolio selector in the left-side menu and select **Create a business portfolio**.
  3. Enter a **Business portfolio name**.
  4. Enter your **name**.
  5. Enter a **business email**.
  6. Select **Submit** or **Create**.
- A Meta business app configured with WhatsApp: Once you have a developer account, you will create a Meta business app. ??? note “Set up a Meta business app with WhatsApp”
  1. Visit the Meta for Developers Apps dashboard
  2. Select **Create app**.
  3. In **Add products to your app**, select **Set up** in the WhatsApp tile. Refer to Add the WhatsApp Product for more detail.
  4. This opens the WhatsApp **Quickstart** page. Select your business portfolio.
  5. Select **Continue**.
  6. In the left-side menu, go to **App settings > Basic**.
  7. Set the **Privacy Policy URL** and **Terms of Service URL** for the app.
  8. Change the **App Mode** to **Live**.

Supported authentication methods

- API key: Use for the WhatsApp Business Cloud node.
- OAuth2: Use for the WhatsApp Trigger node.

Related resources

Refer to WhatsApp's API documentation for more information about the service.

Meta classifies users who create WhatsApp business apps as Tech Providers; refer to Meta's Get Started for Tech Providers for more information.

Using API key

You need WhatsApp API key credentials to use the WhatsApp Business Cloud node.

To configure this credential, you'll need:

- An API **Access Token**
- A **Business Account ID**

To generate an access token, follow these steps:

1. Visit the Meta for Developers Apps dashboard.
2. Select your Meta app.
3. In the left-side menu, select **WhatsApp > API Setup**.
4. Select **Generate access token** and confirm the access you want to grant.
5. Copy the **Access token** and add it to n8n as the **Access Token**.
6. Copy the **WhatsApp Business Account ID** and add it to n8n as the **Business Account ID**.

Refer to Test Business Messaging on WhatsApp for more information on the above steps.

Fully verifying and launching your app will take further configuration. Refer to Meta's Get Started for Tech Providers Steps 5 and beyond for more information. Refer to App Review for more information on the Meta App Review process.

Using OAuth2

You need WhatsApp OAuth2 credentials to use the WhatsApp Trigger node.

To configure this credential, you'll need:

- A **Client ID**
- A **Client Secret**

To retrieve these items, follow these steps:

1. Visit the Meta for Developers Apps dashboard.
2. Select your Meta app.
3. In the left-side menu, select **App settings > Basic**.
4. Copy the **App ID** and enter it as the **Client ID** within the n8n credential.
5. Copy the **App Secret** and enter it as the **Client Secret** within the n8n credential.

Fully verifying and launching your app will take further configuration. Refer to Meta's Get Started for Tech Providers Steps 5 and beyond for more information. Refer to App Review for more information on the Meta App Review process.

Wise credentials

You can use these credentials to authenticate the following nodes:

- Wise
- Wise Trigger

Prerequisites

Create a Wise account.

Supported authentication methods

- API token

Related resources

Refer to Wise's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- An **API Token**: Go to your **user menu > Settings > API tokens** to generate an API token. Enter the generated API key in your n8n credential. Refer to Getting started with the API for more information.
- Your **Environment**: Select the environment that best matches your Wise account environment.
  - If you're using a Wise test sandbox account, select **Test**.
  - Otherwise, select **Live**.
- **Private Key (Optional)**: For live endpoints requiring Strong Customer Authentication (SCA), generate a public and private key. Enter the private key here. Refer to Add a private key for more information.
  - If you're using a **Test** environment, you'll only need to enter a Private Key if you've enabled Strong Customer Authentication on the public keys management page.

Add a private key

Wise protects some live endpoints and operations with Strong Customer Authentication (SCA). Refer to Strong Customer Authentication & 2FA for details.

If you make a request to an endpoint that requires SCA, Wise returns a 403 Forbidden HTTP status code. The error returned will look like this:

> This request requires Strong Customer Authentication (SCA). Please add a key pair to your account and n8n credentials. See https://api-docs.transferwise.com/#strong-customer-authentication-personal-token

To use endpoints requiring SCA, generate an RSA key pair and add the relevant key information to both Wise and n8n:

1. Generate an RSA key pair:

```
$ openssl genrsa -out private.pem 2048
$ openssl rsa -pubout -in private.pem -out public.pem
```

2. Add the content of the public key `public.pem` to your Wise **user menu > Settings > API tokens > Manage public keys**.
3. Add the content of the private key `private.pem` in n8n to the **Private Key (Optional)**.

Refer to Personal Token SCA for more information.

Wolfram|Alpha credentials

You can use these credentials to authenticate the following nodes:

- Wolfram|Alpha

Supported authentication methods

- API key

Related resources

Refer to Wolfram|Alpha's Simple API documentation for more information about the service.

-8<- "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

To configure this credential, you'll need a registered Wolfram ID and:

- An **App ID**

To get an App ID:

1. Open the Wolfram|Alpha Developer Portal and go to **API Access**.
2. Select **Get an App ID**.
3. Enter a **Name** for your application, like `n8n integration`.
4. Enter a **Description** for your application.

5. Select **Simple API** as the **API**.
6. Select **Submit**.
7. Copy the generated **App ID** and enter it in your n8n credential.

Refer to **Getting Started** in the Wolfram|Alpha Simple API documentation for more information.

Resolve Forbidden connection error

If you enter your App ID and get an error that the credential is **Forbidden**, make sure that you have verified your email address for your Wolfram ID:

1. Go to your Wolfram ID Details.
2. If you don't see the **Verified** label underneath your **Email address**, select the link to **Send a verification email**.
3. You must open the link in that email to verify your email address.

It may take several minutes for the verification to populate to the API, but once it does, retrying the n8n credential should succeed.

WooCommerce credentials

You can use these credentials to authenticate the following nodes:

- WooCommerce
- WooCommerce Trigger

Prerequisites

- Install the WooCommerce plugin on your WordPress website.
- In WordPress, go to **Settings > Permalinks** and set your WordPress permalinks to use something other than **Plain**.

Supported authentication methods

- API key

Related resources

Refer to WooCommerce's REST API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- A **Consumer Key**: Created when you generate an API key.
- A **Consumer Secret**: Created when you generate an API key.
- A **WooCommerce URL**

To generate an API key and set up your credential:

1. Go to **WooCommerce > Settings > Advanced > Rest API > Add key**.
2. Select **Read/Write** from the **Permissions** dropdown.
3. Copy the generated **Consumer Key** and **Consumer Secret** and enter them into your n8n credentials.
4. Enter your WordPress site URL as the **WooCommerce URL**.
5. By default, n8n passes your credential details in the Authorization header. If you need to pass them as query string parameters instead, turn on **Include Credentials in Query**.

Refer to Generate Keys for more information.

Resolve “Consumer key is missing” error

When you try to connect your credentials, you may receive an error like this: `Consumer key is missing`.

This occurs when the server can’t parse the Authorization header details when authenticating over SSL.

To resolve it, turn on the **Include Credentials in Query** toggle to pass the consumer key/secret as query string parameters instead and retry the credential.

WordPress credentials

You can use these credentials to authenticate the following nodes:

- WordPress

Prerequisites

- Create a WordPress account or deploy WordPress on a server.

Supported authentication methods

- Basic auth

Related resources

Refer to WordPress’s API documentation for more information about the service.

Using basic auth

To configure this credential, you’ll need:

- Your WordPress **Username**
- A WordPress application **Password**

- Your **WordPress URL**
- Decide whether to **Ignore SSL Issues**

Using this credential involves three steps:

1. Enable two-step authentication.
2. Create an application password.
3. Set up the credential.

Refer to the detailed instructions below for each step.

Enable two-step authentication

To generate an application password, you must first enable Two-Step Authentication in WordPress. If you've already done this, skip to the next section.

1. Open your WordPress profile.
2. Select **Security** from the left menu.
3. Select **Two-Step Authentication**. The **Two-Step Authentication** page opens.
4. If Two-Step Authentication isn't enabled, you must enable it.
5. Choose whether to enable it using an authenticator app or SMS codes and follow the on-screen instructions.

Refer to WordPress's Enable Two-Step Authentication for detailed instructions.

Create an application password

With Two-Step Authentication enabled, you can now generate an application password:

1. From the WordPress **Security > Two-Step Authentication** page, select **+ Add new application password** in the **Application passwords** section.
2. Enter an **Application name**, like `n8n integration`.
3. Select **Generate Password**.
4. Copy the password it generates. You'll use this in your n8n credential.

Set up the credential

Congratulations! You're now ready to set up your n8n credential:

1. Enter your WordPress **Username** in your n8n credential.
2. Enter the application password you copied above as the **Password** in your n8n credential.
3. Enter the URL of your WordPress site as the **WordPress URL**.
4. Optional: Use the **Ignore SSL Issues** to choose whether you want the n8n credential to connect even if SSL certificate validation fails (turned on) or whether to respect SSL certificate validation (turned off).

Workable credentials

You can use these credentials to authenticate the following nodes:

- Workable Trigger

Prerequisites

Create a Workable account.

Supported authentication methods

- API key

Related resources

Refer to Workable's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- A **Subdomain**: Your Workable subdomain is the part of your Workable domain between `https://` and `.workable.com`. So if the full domain is `https://n8n.workable.com`, the subdomain is `n8n`. The subdomain is also displayed on your Workable **Company Profile** page.
- An **Access Token**: Go to your **profile > Integrations > Apps** and select **Generate API token**. Refer to Generate a new token for more information.

Wufoo credentials

You can use these credentials to authenticate the following nodes:

- Wufoo Trigger

Prerequisites

Create a Wufoo account.

Supported authentication methods

- API key

Related resources

Refer to Wufoo's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: Get your API key from the Wufoo Form Manager. To the right of a form, select **More > API Information**. Refer to Using API Information and Webhooks for more information.
- A **Subdomain**: Your subdomain is the part of your Wufoo URL that comes after `https://` and before `wufoo.com`. So if the full domain is `https://n8n.wufoo.com`, the subdomain is `n8n`. Admins can view the subdomain in the **Account Manager**. Refer to Your Subdomain for more information.

X (formerly Twitter) credentials

You can use these credentials to authenticate the following nodes:

- X (formerly Twitter)

Prerequisites

- Create an X developer account.
- Create a Twitter app or use the default project and app created when you sign up for the developer portal. Refer to each supported authentication method below for more details on the app's configuration.

Supported authentication methods

- OAuth2

Related resources

Refer to X's API documentation for more information about the service. Refer to X's API authentication documentation for more information about authenticating with the service.

Refer to Application-only Authentication for more information about app-only authentication.

Using OAuth2

Use this method if you're using n8n version 0.236.0 or later.

To configure this credential, you'll need:

- A **Client ID**
- A **Client Secret**

To generate your Client ID and Client Secret:

1. In the Twitter developer portal, open your project.

2. On the project's **Overview** tab, find the **Apps** section and select **Add App**.
3. Give your app a **Name** and select **Next**.
4. Go to the **App Settings**.
5. In the **User authentication settings**, select **Set Up**.
6. Set the **App permissions**. Choose **Read and write and Direct message** if you want to use all functions of the n8n X node.
7. In the **Type of app** section, select **Web App, Automated App or Bot**.
8. In n8n, copy the **OAuth Redirect URL**.
9. In your X app, find the **App Info** section and paste that URL in as the **Callback URI / Redirect URL**.
10. Add a **Website URL**.
11. Save your changes.
12. Copy the **Client ID** and **Client Secret** displayed in X and add them to the corresponding fields in your n8n credential.

Refer to X's OAuth 2.0 Authentication documentation for more information on working with this authentication method.

X rate limits

X has time-based rate limits per endpoint based on your developer access plan level. X calculates app rate limits and user rate limits independently. Refer to Rate limits for the access plan level rate limits and guidance on avoiding hitting them.

Use the guidance below for calculating rate limits:

- If you're using the deprecated OAuth method, user rate limits apply. You'll have one limit per time window for each set of users' access tokens.
- If you're Using OAuth2, app rate limits apply. You'll have a limit per time window for requests made by your app.

X calculates user rate limits and app rate limits independently.

Refer to X's Rate limits and authentication methods for more information about these rate limit types.

---

xAI credentials

You can use these credentials to authenticate the following nodes:

- Chat xAI Grok

Prerequisites

Create an xAI account.

Supported authentication methods

- API key

Related resources

Refer to xAI's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- An **API Key**: You can create a new API key on the xAI Console API Keys page.

Refer to the The Hitchhiker's Guide to Grok | xAI for more information.

Xata credentials

You can use these credentials to authenticate the following nodes:

- Xata

Prerequisites

Create a Xata database or an account on an existing database.

Supported authentication methods

- API key

Related resources

Refer to Xata's documentation for more information about the service.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

To configure this credential, you'll need:

- The **Database Endpoint**: The Workspace API requires that you identify the database you're requesting information from using this format: `https://{workspace-display-name}-{workspace-id}.{region}.xata.sh/db/{dbname}`. Refer to Workspace API for more information.
  - `{workspace-display-name}`: The workspace display name is an optional identifier you can include in your Database Endpoint. The API ignores it, but including it can make it easier to figure out which workspace this database is in if you're saving multiple credentials.
  - `{workspace-id}`: The unique ID of the workspace, 6

alphanumeric characters.
    - `{region}`: The hosting region for the database. This value must match the database region configuration.
    - `{dbname}`: The name of the database you're interacting with.
- A **Branch**: Enter the name of the GitHub branch for your database.
- An **API Key**: To generate an API key, go to **Account Settings** and select **+ Add a key**. Refer to Generate an API Key for more information.

Xero credentials

You can use these credentials to authenticate the following nodes:

- Xero

Prerequisites

Create a Xero account.

Supported authentication methods

- OAuth2

Related resources

Refer to Zero's API documentation for more information about the service.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated when you create a new app for a custom connection.
- A **Client Secret**: Generated when you create a new app for a custom connection.

To generate your Client ID and Client Secret, create an OAuth2 custom connection app in your Xero developer portal **My Apps**.

Use these settings for your app:

- Select **Web app** as the **Integration Type**.
- For the **Company or Application URL**, enter the URL of your n8n server or reverse proxy address. For cloud users, for example, this is: `https://your-username.app.n8n.cloud/`.
- Copy the **OAuth Redirect URL** from n8n and add it as an **OAuth 2.0 redirect URI** in your app.
- Select appropriate **scopes** for your app. Refer to OAuth2 Scopes for more information.
    - To use all functionality in the Xero node, add the

`accounting.contacts` and `accounting.transactions` scopes.

Refer to Xero's OAuth Custom Connections documentation for more information.

Yourls credentials

You can use these credentials to authenticate the following nodes:

- Yourls

Prerequisites

Install Yourls on your server.

Supported authentication methods

- API key

Related resources

Refer to Yourl's documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- A **Signature** token: Go to **Tools > Secure passwordless API call** to get your **Signature** token. Refer to Yourl's Passworldess API documentation for more information.
- A **URL**: Enter the URL of your Yourls instance.

Zabbix credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create a Zabbix Cloud account or self-host your own Zabbix server.

Supported authentication methods

- API key

Related resources

Refer to Zabbix's API documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using API key

To configure this credential, you'll need:

- an **API Token**: An API key for your Zabbix user.
- the **URL**: The URL of your Zabbix server. Don't include `/zabbix` as part of the URL.

Refer to Zabbix's API documentation for more information about authenticating to the service.

Zammad credentials

You can use these credentials to authenticate the following nodes:

- Zammad

Prerequisites

- Create a hosted Zammad account or set up your own Zammad instance.
- For token authentication, enable **API Token Access** in **Settings > System > API**. Refer to Setting up a Zammad for more information.

Supported authentication methods

- Basic auth
- Token auth: Zammad recommends using this authentication method.

Related resources

Refer to Zammad's API Authentication documentation for more information about authenticating with the service.

Using basic auth

To configure this credential, you'll need:

- A **Base URL**: Enter the URL of your Zammad instance.
- An **Email** address: Enter the email address you use to log in to Zammad.

- A **Password**: Enter your Zammad password.
- **Ignore SSL Issues**: When turned on, n8n will connect even if SSL certificate validation fails.

Using token auth

To configure this credential, you'll need:

- A **Base URL**: Enter the URL of your Zammad instance.
- An **Access Token**: Once **API Token Access** is enabled for the Zammad instance, any user with the `user_preferences.access_token` permission can generate an **Access Token** by going to your **avatar > Profile > Token Access** and **Create** a new token.
  - The access token permissions depend on what actions you'd like to complete with this credential. For all functionality within the Zammad node, select:
    - `admin.group`
    - `admin.organization`
    - `admin.user`
    - `ticket.agent`
    - `ticket.customer`
- **Ignore SSL Issues**: When turned on, n8n will connect even if SSL certificate validation fails.

Zendesk credentials

You can use these credentials to authenticate the following nodes:

- Zendesk
- Zendesk Trigger

Prerequisites

- Create a Zendesk account.
- For API token authentication, enable token access to the API in Admin Center under **Apps and integrations > APIs > Zendesk APIs**.

Supported authentication methods

- API token
- OAuth2

Related resources

Refer to Zendesk's API documentation for more information about the service.

Using API token

To configure this credential, you'll need:

- Your **Subdomain**: Your Zendesk subdomain is the portion of the URL between `https://` and `.zendesk.com`. For example, if the Zendesk URL is `https://n8n-example.zendesk.com/agent/dashboard`, the subdomain is `n8n-example`.
- An **Email** address: Enter the email address you use to log in to Zendesk.
- An **API Token**: Generate an API token in **Apps and integrations > APIs > Zendesk API**. Refer to API token for more information.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated when you create a new OAuth client.
- A **Client Secret**: Generated when you create a new OAuth client.
- Your **Subdomain**: Your Zendesk subdomain is the portion of the URL between `https://` and `.zendesk.com`. For example, if the Zendesk URL is `https://n8n-example.zendesk.com/agent/dashboard`, the subdomain is `n8n-example`.

To create a new OAuth client, go to **Apps and integrations > APIs > Zendesk API > OAuth Clients**.

Use these settings:

- Copy the **OAuth Redirect URL** from n8n and enter it as a **Redirect URL** in the OAuth client.
- Copy the **Unique identifier** for the Zendesk client and enter this as your n8n **Client ID**.
- Copy the **Secret** from Zendesk and enter this as your n8n **Client Secret**

Refer to Registering your application with Zendesk for more information.

Zep credentials

You can use these credentials to authenticate the following nodes:

- Zep
- Zep Vector Store

Supported authentication methods

- API key

Related resources

Refer to Zep's Cloud SDK documentation for more information about the service. Refer to Zep's REST API documentation for information about the API.

–8<– "_snippets/integrations/builtin/cluster-nodes/langchain-overview-link.md"

Using API key

To configure this credential, you'll need a Zep server with at least one project and:

- An **API URL**
- An **API Key**

Setup depends on whether you're using Zep Cloud or self-hosted Zep Open Source.

Zep Cloud setup

Follow these instructions if you're using Zep Cloud:

1. In Zep, open the **Project Settings**.
2. In the **Project Keys** section, select **Add Key**.
3. Enter a **Key Name**, like `n8n integration`.
4. Select **Create**.
5. Copy the key and enter it in your n8n integration as the **API Key**.
6. Turn on the **Cloud** toggle.

Self-hosted Zep Open Source setup

Follow these instructions if you're self-hosting Zep Open Source:

1. Enter the JWT token for your Zep server as the **API Key** in n8n.
2. Make sure the **Cloud** toggle is off.
3. Enter the URL for your Zep server as the **API URL**.

Zoho credentials

You can use these credentials to authenticate the following nodes:

- Zoho CRM

Prerequisites

Create a Zoho account.

Supported authentication methods

- OAuth2

Related resources

Refer to Zoho's CRM API documentation for more information about the service.

Using OAuth2

To configure this credential, you'll need:

- An **Access Token URL**: Zoho provides region-specific access token URLs. Select the region that best fits your Zoho data center:
  - **AU**: Select this option for Australia data center.
  - **CN**: Select this option for Canada data center.
  - **EU**: Select this option for the European Union data center.
  - **IN**: Select this option for the India data center.
  - **US**: Select this option for the United States data center.

Refer to Multi DC for more information about selecting a data center.

–8<– "_snippets/integrations/builtin/credentials/cloud-oauth-button.md"

If you need to configure OAuth2 from scratch, register an application with Zoho.

Use these settings for your application:

- Select **Server-based Applications** as the **Client Type**.
- Copy the **OAuth Callback URL** from n8n and enter it in the Zoho **Authorized Redirect URIs** field.
- Copy the **Client ID** and **Client Secret** from the application and enter them in your n8n credential.

Zoom credentials

You can use these credentials to authenticate the following nodes:

- Zoom

Prerequisites

Create a Zoom account. Your account must have one of the following permissions:

- Account owner
- Account admin
- Zoom for developers role

Supported authentication methods

- API JWT token
- OAuth2

Related resources

Refer to Zoom's API documentation for more information about the service.

Using API JWT token

This authentication method has been fully deprecated by Zoom. Don't create new credentials with it.

To configure this credential, you'll need:

- A **JWT token**: To create a JWT token, create a new JWT app in the Zoom App Marketplace.

Using OAuth2

To configure this credential, you'll need:

- A **Client ID**: Generated when you create an OAuth app on the Zoom App Marketplace.
- A **Client Secret**: Generated when you create an OAuth app.

To generate your **Client ID** and **Client Secret**, create an OAuth app.

Use these settings for your OAuth app:

- Select **User-managed app** for **Select how the app is managed**.
- Copy the **OAuth Callback URL** from n8n and enter it as an **OAuth Redirect URL** in Zoom.
- If your n8n credential displays a **Whitelist URL**, also enter that URL as a an **OAuth Redirect URL**.
- Enter **Scopes** for the scopes you plan to use. For all functionality in the Zoom node, select:
  - `meeting:read`
  - `meeting:write`
  - Refer to OAuth scopes | Meeting scopes for more information on meeting scopes.
- Copy the **Client ID** and **Client Secret** provided in the Zoom app and enter them in your n8n credential.

Zscaler ZIA credentials

–8<– "_snippets/integrations/builtin/credentials/cred-only-statement.md"

Prerequisites

Create an admin account on a Zscaler Internet Access (ZIA) cloud instance.

Supported authentication methods

- Basic auth and API key combo

Related resources

Refer to Zscaler ZIA's documentation for more information about the service.

This is a credential-only node. Refer to Custom API operations to learn more. View example workflows and related content on n8n's website.

Using basic auth and API key combo

To configure this credential, you'll need:

- A **Base URL**: Enter the base URL of your Zscaler ZIA cloud name. To get your base URL, log in to the ZIA Admin Portal and go to **Administration > Cloud Service API Security**. The base URL is displayed in both the **Cloud Service API Key** tab and the **OAuth 2.0 Authorization Servers** tab.
- A **Username**: Enter your ZIA admin username.
- A **Password**: Enter your ZIA admin password.
- An **Api Key**: Get an API key by creating one from **Administration > Cloud Service API Security > Cloud Service API Key**.

Refer to About Cloud Service API Key for more detailed instructions.

Zulip credentials

You can use these credentials to authenticate the following nodes:

- Zulip

Prerequisites

Create a Zulip account.

Supported authentication methods

- API key

Related resources

Refer to Zulip's API documentation for more information about the service.

Using API key

To configure this credential, you'll need:

- A **URL**: Enter the URL of your Zulip domain.
- An **Email** address: Enter the email address you use to log in to Zulip.
- An **API Key**: Get your API key in the **Gear cog > Personal Settings > Account & privacy > API Key**. Refer to API Keys for more information.

Custom API operations

–8<– “_snippets/integrations/credential-only-intro.md”

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

–8<– “_snippets/integrations/predefined-credential-type-how-to.md”

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:

[Image: The scopes box]

Handling API rate limits

API rate limits are restrictions on request frequency. For example, an API may limit the number of requests you can make per minute, or per day.

APIs can also limits how much data you can send in one request, or how much data the API sends in a single response.

Identify rate limit issues

When an n8n node hits a rate limit, it errors. n8n displays the error message in the node output panel. This includes the error message from the service.

If n8n received error 429 (too many requests) from the service, the error message is **The service is receiving too many requests from you**.

To check the rate limits for the service you’re using, refer to the API documentation for the service.

Handle rate limits for integrations

There are two ways to handle rate limits in n8n's integrations: using the Retry On Fail setting, or using a combination of the Loop Over Items and Wait nodes:

- Retry On Fail adds a pause between API request attempts.
- With Loop Over Items and Wait you can break you request data into smaller chunks, as well as pausing between requests.

Enable Retry On Fail

When you enable Retry On Fail, the node automatically tries the request again if it fails the first time.

1. Open the node.
2. Select **Settings**.
3. Enable the **Retry On Fail** toggle.
4. Configure the retry settings: if using this to work around rate limits, set **Wait Between Tries (ms)** to more than the rate limit. For example, if the API you're using allows one request per second, set **Wait Between Tries (ms)** to `1000` to allow a 1 second wait.

Use Loop Over Items and Wait

Use the Loop Over Items node to batch the input items, and the Wait node to introduce a pause between each request.

1. Add the Loop Over Items node before the node that calls the API. Refer to Loop Over Items for information on how to configure the node.
2. Add the Wait node after the node that calls the API, and connect it back to the Loop Over Items node. Refer to Wait for information on how to configure the node.

For example, to handle rate limits when using OpenAI:

[Image: "Screenshot of a workflow using the Loop Over Items node and Wait node to handle API rate limits for the OpenAI APIs"]

Handle rate limits in the HTTP Request node

The HTTP Request node has built-in settings for handling rate limits and large amounts of data.

Batch requests

Use the Batching option to send more than one request, reducing the request size, and introducing a pause between requests. This is the equivalent of using Loop Over Items and Wait.

1. In the HTTP Request node, select **Add Option** > **Batching**.
2. Set **Items per Batch**: this is the number of input items to include in each request.
3. Set **Batch Interval (ms)** to introduce a delay between requests.

For example, if the API you're using allows one request per second, set **Wait Between Tries (ms)** to `1000` to allow a 1 second wait.

Paginate results

APIs paginate their results when they need to send more data than they can handle in a single response. For more information on pagination in the HTTP Request node, refer to HTTP Request node | Pagination.

Install and manage community nodes

There are three ways to install community nodes:

- Within n8n using the nodes panel (for verified community nodes only).
- Within n8n using the GUI: Use this method to install community nodes from the npm registry.
- Manually from the command line: use this method to install community nodes from npm if your n8n instance doesn't support installation through the in-app GUI.

Install verified community nodes in the n8n app

Install a community node

To install a verified community node:

1. Go to the **Canvas** and open the **nodes panel** (either by selecting '+' or pressing ++tab++).
2. **Search** for the node that you're looking for. If there is a matching verified community node, you will see a **More from the community** section at the bottom of the nodes panel.
3. Select the node you want to install. This takes you to a detailed view of the node, showing all the supported actions.
4. Select **install**. This will install the node for your instance and enable all members to use it in their workflows.
5. You can now add the node to your workflows.

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** Image: Three dots options menu.
3. Select **Uninstall package**.

4. Select **Uninstall Package** in the confirmation modal.

Install community nodes from npm in the n8n app

Install a community node

To install a community node from npm:

1. Go to **Settings** > **Community Nodes**.
2. Select **Install**.
3. Find the node you want to install:
    1. Select **Browse**. n8n takes you to an npm search results page, showing all npm packages tagged with the keyword `n8n-community-node-package`.
    2. Browse the list of results. You can filter the results or add more keywords.
    3. 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.
    4. 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**.

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** Image: Three dots options menu.
3. Select **Uninstall package**.
4. Select **Uninstall Package** in the confirmation modal.

Upgrade a community node

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.

Manually install community nodes from npm

You can manually install community nodes from the npm registry on self-hosted n8n.

You need to manually install community nodes in the following circumstances:

- Your n8n instance runs in queue mode.
- You want to install private packages.

Install a community node

Access your Docker shell:

```
docker exec -it n8n sh
```

Create `~/.n8n/nodes` if it doesn't already exist, and navigate into it:

```
mkdir ~/.n8n/nodes
cd ~/.n8n/nodes
```

Install the node:

```
npm i n8n-nodes-nodeName
```

Then restart n8n.

Uninstall a community node

Access your Docker shell:

```
docker exec -it n8n sh
```

Run npm uninstall:

```
npm uninstall n8n-nodes-nodeName
```

Upgrade a community node

Upgrade to the latest version

Access your Docker shell:

```
docker exec -it n8n sh
```

Run npm update:

```
npm update n8n-nodes-nodeName
```

Upgrade or downgrade to a specific version

Access your Docker shell:

```
docker exec -it n8n sh
```

Run npm uninstall to remove the current version:

```
npm uninstall n8n-nodes-nodeName
```

Run npm install with the version specified:

```
# Replace 2.1.0 with your version number
npm install n8n-nodes-nodeName@2.1.0
```

Risks when using community nodes

Installing community nodes from npm means you are installing unverified code from a public source into your n8n instance. This has some risks.

Risks include:

- System security: community nodes have full access to the machine that n8n runs on, and can do anything, including malicious actions.
- Data security: any community node that you use has access to data in your workflows.
- Breaking changes: 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.

Report bad community nodes

You can report bad community nodes to security@n8n.io

Disable community nodes

If you are self-hosting n8n, you can disable community nodes by setting `N8N_COMMUNITY_PACKAGES_ENABLED` to `false`. On n8n cloud, visit the Cloud Admin Panel and disable community nodes from there. See troubleshooting for more information.

---

n8n community node blocklist

n8n maintains a blocklist of community nodes. You can't install any node on this list.

n8n may add community nodes to the blocklist for a range of reasons, including:

- The node is intentionally malicious
- It's low quality (low enough to be harmful)

If you are a community node creator whose node is on the blocklist, and you believe this is a mistake, contact hello@n8n.io.

Using community nodes

To use community nodes, you first need to install them.

Adding community nodes to your workflow

After installing a community node, you can use it like any other node. n8n displays the node in search results in the **Nodes** panel. n8n marks community nodes with a **Package** Image: Package icon icon in the nodes panel.

Community nodes with duplicate names

It's possible for several community nodes to have the same name. If you use two nodes with the same name in your workflow, they'll look the same, unless they have different icons.

Troubleshooting and errors

Error: Missing packages

n8n installs community nodes directly onto the hard disk. The files must be available at startup for n8n to load them. If the packages aren't available at startup, you get an error warning of missing packages.

If running n8n using Docker: depending on your Docker setup, you may lose the packages when you recreate your container or upgrade your n8n version. You must either:

- Persist the contents of the `~/.n8n/nodes` directory. This is the best option. If you follow the Docker installation guide, the setup steps include persisting this directory.
- Set the `N8N_REINSTALL_MISSING_PACKAGES` environment variable to `true`.

The second option might increase startup time and may cause health checks to fail.

Prevent loading community nodes on n8n cloud

If your n8n cloud instance crashes and fails to start, you can prevent installed community nodes from loading on instance startup. Visit the Cloud Admin Panel > **Manage** and toggle **Disable all community nodes** to **`true`**. This toggle is only visible when you allow community node installation.

Building community nodes

–8<– "_snippets/integrations/submit-community-node.md"

Creating nodes

Learn how to build your own custom nodes.

This section includes:

- Guidance on planning your build, including which style to use.
- Tutorials for different node building styles.
- Instructions for testing your node, including how to use the n8n node linter and troubleshooting support.
- How to share your node with the community, submit it for verification by n8n, or use it as a private node.
- Reference material, including UI elements and information on the individual files that make up a node.

Prerequisites

This section assumes the following:

- Some familiarity with JavaScript and TypeScript.
- Ability to manage your own development environment, including

git.
- Knowledge of npm, including creating and submitting packages.
- Familiarity with n8n, including a good understanding of data structures and item linking.

Plan a node

This section provides guidance on designing your node, including key technical decisions such as choosing your node building style.

When building a node, there are design choices you need to make before you start:

- Which node type you need to build.
- Which node building style to use.
- Your UI design and UX principles
- Your node's file structure.

Node types: Trigger and Action

There are two node types you can build for n8n: trigger nodes and action nodes.

Both types provide integrations with external services.

Trigger nodes

Trigger nodes start a workflow and supply the initial data. A workflow can contain multiple trigger nodes but with each execution, only one of them will execute, depending on the triggering event.

There are three types of trigger nodes in n8n:

Type	Description	Example Nodes
Webhook	Nodes for services that support webhooks. These nodes listen for events and trigger workflows in real time.	Zendesk Trigger, Telegram Trigger, Brevo Trigger
Polling	Nodes for services that don't support webhooks. These nodes periodically check for new data, triggering workflows when they detect updates.	Airtable Trigger, Gmail Trigger, Google Sheet Trigger, RssFeed Read Trigger
	Nodes that handle real-time responses not related to HTTP requests or polling.	AMQP Trigger, RabbitMQ Trigger,

Others	This includes message queue nodes and time-based triggers.	MQTT Trigger, Schedule Trigger, Email Trigger (IMAP)

Action nodes

Action nodes perform operations as part of your workflow. These can include manipulating data, and triggering events in other systems.

Choose your node building approach

n8n has two node-building styles, declarative and programmatic.

You should use the declarative style for most nodes. This style:

- Uses a JSON-based syntax, making it simpler to write, with less risk of introducing bugs.
- Is more future-proof.
- Supports integration with REST APIs.

The programmatic style is more verbose. You must use the programmatic style for:

- Trigger nodes
- Any node that isn't REST-based. This includes nodes that need to call a GraphQL API and nodes that use external dependencies.
- Any node that needs to transform incoming data.
- Full versioning. Refer to Node versioning for more information on types of versioning.

Data handling differences

The main difference between the declarative and programmatic styles is how they handle incoming data and build API requests. The programmatic style requires an `execute()` method, which reads incoming data and parameters, then builds a request. The declarative style handles this using the `routing` key in the `operations` object. Refer to Node base file for more information on node parameters and the `execute()` method.

Syntax differences

To understand the difference between the declarative and programmatic styles, compare the two code snippets below. This example creates a simplified version of the SendGrid integration, called "FriendGrid." The following code snippets aren't complete: they emphasize the differences in the node building styles.

In programmatic style:

```
import {
```

```
            IExecuteFunctions,
            INodeExecutionData,
            INodeType,
            INodeTypeDescription,
            IRequestOptions,
        } from 'n8n-workflow';

        // Create the FriendGrid class
        export class FriendGrid implements INodeType {
          description: INodeTypeDescription = {
            displayName: 'FriendGrid',
            name: 'friendGrid',
            . . .
            properties: [
              {
                displayName: 'Resource',
                . . .
              },
              {
                displayName: 'Operation',
                name: 'operation',
                type: 'options',
                displayOptions: {
                  show: {
                      resource: [
                      'contact',
                      ],
                  },
                },
                options: [
                  {
                    name: 'Create',
                    value: 'create',
                    description: 'Create a contact',
                  },
                ],
                default: 'create',
                description: 'The operation to perform.',
              },
              {
                displayName: 'Email',
                name: 'email',
                . . .
              },
              {
                displayName: 'Additional Fields',
                // Sets up optional fields
              },
            ],
        };

          async execute(this: IExecuteFunctions):
Promise<INodeExecutionData[][]> {
            let responseData;
            const resource = this.getNodeParameter('resource', 0) as string;
            const operation = this.getNodeParameter('operation', 0) as
string;
            //Get credentials the user provided for this node
            const credentials = await this.getCredentials('friendGridApi')
as IDataObject;
```

```
            if (resource === 'contact') {
              if (operation === 'create') {
              // Get email input
              const email = this.getNodeParameter('email', 0) as string;
              // Get additional fields input
              const additionalFields =
this.getNodeParameter('additionalFields', 0) as IDataObject;
              const data: IDataObject = {
                  email,
              };

              Object.assign(data, additionalFields);

              // Make HTTP request as defined in
https://sendgrid.com/docs/api-reference/
              const options: IRequestOptions = {
                headers: {
                    'Accept': 'application/json',
                    'Authorization': `Bearer ${credentials.apiKey}`,
                },
                method: 'PUT',
                body: {
                    contacts: [
                    data,
                    ],
                },
                url: `https://api.sendgrid.com/v3/marketing/contacts`,
                json: true,
              };
              responseData = await this.helpers.httpRequest(options);
              }
            }
            // Map data to n8n data
            return [this.helpers.returnJsonArray(responseData)];
          }
        }
```

In declarative style:

```
        import { INodeType, INodeTypeDescription } from 'n8n-workflow';

        // Create the FriendGrid class
        export class FriendGrid implements INodeType {
          description: INodeTypeDescription = {
            displayName: 'FriendGrid',
            name: 'friendGrid',
            . . .
            // Set up the basic request configuration
            requestDefaults: {
              baseURL: 'https://api.sendgrid.com/v3/marketing'
            },
            properties: [
              {
                displayName: 'Resource',
                . . .
              },
              {
                displayName: 'Operation',
                name: 'operation',
                type: 'options',
```

```
        displayOptions: {
          show: {
            resource: [
              'contact',
            ],
          },
        },
        options: [
          {
            name: 'Create',
            value: 'create',
            description: 'Create a contact',
            // Add the routing object
            routing: {
              request: {
                method: 'POST',
                url: '=/contacts',
                send: {
                  type: 'body',
                  properties: {
                    email: {{$parameter["email"]}}
                  }
                }
              }
            },
            // Handle the response to contact creation
            output: {
              postReceive: [
                {
                  type: 'set',
                  properties: {
                    value: '={{ { "success": $response } }}'
                  }
                }
              ]
            }
          },
        ],
        default: 'create',
        description: 'The operation to perform.',
      },
      {
        displayName: 'Email',
        . . .
      },
      {
        displayName: 'Additional Fields',
        // Sets up optional fields
      },
    ],
  }
  // No execute method needed
}
```

Design your node's user interface

Most nodes are a GUI (graphical user interface) representation of an API. Designing the interface means finding a user-friendly way to represent API endpoints and parameters. Directly translating an entire API into form fields in a node may not result in a good user experience.

This document provides design guidance and standards to follow. These guidelines are the same as those used by n8n. This helps provide a smooth and consistent user experience for users mixing community and built-in nodes.

Design guidance

All node's use n8n's node UI elements, so you don't need to consider style details such as colors, borders, and so on. However, it's still useful to go through a basic design process:

- Review the documentation for the API you're integrating. Ask yourself:
  - What can you leave out?
  - What can you simplify?
  - Which parts of the API are confusing? How can you help users understand them?
- Use a wireframe tool to try out your field layout. If you find your node has a lot of fields and is getting confusing, consider n8n's guidance on showing and hiding fields.

Standards

UI text style

Element	Style
Drop-down value	Title case
Hint	Sentence case
Info box	Sentence case. Don't use a period (.) for one-sentence information. Always use a period if there's more than one sentence. This field can include links, which should open in a new tab.
Node name	Title case
Parameter name	Title case
Subtitle	Title case
Tooltip	Sentence case. Don't use a period (.) for one-sentence tooltips. Always use a period if there's more than one sentence. This field can include links, which should open in a new tab.

UI text terminology

- Use the same terminology as the service the node connects to. For example, a Notion node should refer to Notion blocks, not Notion paragraphs, because Notion calls these elements blocks. There are exceptions to this rule, usually to avoid technical terms (for example, refer to the guidance on name and description for upsert operations).
- Sometimes a service has different terms for something in its API and in its GUI. Use the GUI language in your node, as this is what most users are familiar with. If you think some users may need to refer to the service's API docs, consider including this information in a hint.
- Don't use technical jargon when there are simpler alternatives.
- Be consistent when naming things. For example, choose one of `directory` or `folder` then stick to it.

Node naming conventions

Convention	Correct	Incorrect
If a node is a trigger node, the displayed name should have 'Trigger' at the end, with a space before.	Shopify Trigger	ShopifyTrigger, Shopify trigger
Don't include 'node' in the name.	Asana	Asana Node, Asana node

Showing and hiding fields

Fields can either be:

- Displayed when the node opens: use this for resources and operations, and required fields.
- Hidden in the **Optional fields** section until a user clicks on that section: use this for optional fields.

Progressively disclose complexity: hide a field until any earlier fields it depends on have values. For example, if you have a **Filter by date** toggle, and a **Date to filter by** datepicker, don't display **Date to filter by** until the user enables **Filter by date**.

Conventions by field type

Credentials

n8n automatically displays credential fields as the top fields in the node.

Resources and operations

APIs usually involve doing something to data. For example, "get all tasks." In this example, "task" is the resource, and "get all" is the operation.

When your node has this resource and operation pattern, your first field should be **Resource**, and your second field should be **Operation**.

Required fields

Order fields by:

- Most important to least important.
- Scope: from broad to narrow. For example, you have fields for **Document**, **Page**, and **Text to insert**, put them in that order.

Optional fields

- Order fields alphabetically. To group similar things together, you can rename them. For example, rename **Email** and **Secondary Email** to **Email (primary)** and **Email (secondary)**.
- If an optional field has a default value that the node uses when the value isn't set, load the field with that value. Explain this in the field description. For example, **Defaults to false**.
- Connected fields: if one optional fields is dependent on another, bundle them together. They should both be under a single option that shows both fields when selected.
- If you have a lot of optional fields, consider grouping them by theme.

Help

There are five types of help built in to the GUI:

- Info boxes: yellow boxes that appear between fields. Refer to UI elements | Notice for more information.
  - Use info boxes for essential information. Don't over-use them. By making them rare, they stand out more and grab the user's attention.
- Parameter hints: lines of text displayed beneath a user input field. Use this when there's something the user needs to know, but an info box would be excessive.
- Node hints: provide help in the input panel, output panel, or node details view. Refer to UI elements | Hints for more information.
- Tooltips: callouts that appear when the user hovers over the tooltip icon [Image: "Screenshot of the tooltip icon. The icon is a ? in a grey circle"]. Use tooltips for extra information that the user might need.
  - You don't have to provide a tooltip for every field. Only add one if it contains useful information.
  - When writing tooltips, think about what the user needs. Don't just copy-paste API parameter descriptions. If the description doesn't make sense, or has errors, improve it.
- Placeholder text: n8n can display placeholder text in a field where the user hasn't entered a value. This can help the user know what's expected in that field.

Info boxes, hints, and tooltips can contain links to more information.

Errors

Make it clear which fields are required.

Add validation rules to fields if possible. For example, check for valid email patterns if the field expects an email.

When displaying errors, make sure only the main error message displays in the red error title. More information should go in **Details**.

Refer to Node Error Handling for more information.

Toggles

- Tooltips for binary states should start with something like **Whether to . . . .**
- You may need a list rather than a toggle:
  - Use toggles when it's clear what happens in a false state. For example, **Simplify Output?**. The alternative (don't simplify output) is clear.
  - Use a dropdown list with named options when you need more clarity. For example, **Append?**. What happens if you don't append is unclear (it could be that nothing happens, or information is overwritten, or discarded).

Lists

- Set default values for lists whenever possible. The default should be the most-used option.
- Sort list options alphabetically.
- You can include list option descriptions. Only add descriptions if they provide useful information.
- If there is an option like **All**, use the word **All**, not shorthand like *****.

Trigger node inputs

When a trigger node has a parameter for specifying which events to trigger on:

- Name the parameter **Trigger on**.
- Don't include a tooltip.

Subtitles

Set subtitles based on the values of the main parameters. For example:

```
        subtitle: '={{$parameter["operation"] + ": " +
$parameter["resource"]}}',
```

IDs

When performing an operation on a specific record, such as "update a task comment" you need a way to specify which record you want to change.

- Wherever possible, provide two ways to specify a record:
  - By choosing from a pre-populated list. You can generate this list using the `loadOptions` parameter. Refer to Base files for more information.
  - By entering an ID.
- Name the field `<Record name> name or ID`. For example, **Workspace Name or ID**. Add a tooltip saying "Choose a name from the list, or specify an ID using an expression." Link to n8n's Expressions documentation.
- Build your node so that it can handle users providing more information than required. For example:
  - If you need a relative path, handle the user pasting in the

absolute path.
    - If the user needs to get an ID from a URL, handle the user pasting in the entire URL.

Dates and timestamps

n8n uses ISO timestamp strings for dates and times. Make sure that any date or timestamp field you add supports all ISO 8601 formats.

JSON

You should support two ways of specifying the content of a text input that expects JSON:

- Typing JSON directly into the text input: you need to parse the resulting string into a JSON object.
- Using an expression that returns JSON.

Node icons

–8<– “_snippets/integrations/node-icons.md”

Common patterns and exceptions

This section provides guidance on handling common design patterns, including some edge cases and exceptions to the main standards.

Simplify responses

APIs can return a lot of data that isn’t useful. Consider adding a toggle that allows users to choose to simplify the response data:

- Name: **Simplify Response**
- Description: **Whether to return a simplified version of the response instead of the raw data**

Upsert operations

This should always be a separate operation with:

- Name: **Create or Update**
- Description: **Create a new record, or update the current one if it already exists (upsert)**

Boolean operators

n8n doesn’t have good support for combining boolean operators, such as AND and OR, in the GUI. Whenever possible, provide options for all ANDs or all ORs.

For example, you have a field called **Must match** to test if values match. Include options to test for **Any** and **All**, as separate options.

Source keys or binary properties

Binary data is file data, such as spreadsheets or images. In n8n, you need a named key to reference the data. Don’t use the terms “binary data” or “binary property” for this field. Instead, use a more

descriptive name: **Input data field name** / **Output data field name**.

Node file structure

Following best practices and standards in your node structure makes your node easier to maintain. It's helpful if other people need to work with the code.

The file and directory structure of your node depends on:

- Your node's complexity.
- Whether you use node versioning.
- How many nodes you include in the npm package.

n8n recommends using the `n8n-node` tool to create the expected node file structure. You can customize the generated scaffolding as required to meet more complex needs.

Required files and directories

Your node must include:

- A `package.json` file at the root of the project. Every npm module requires this.
- A `nodes` directory, containing the code for your node:
  - This directory must contain the base file, in the format `<node-name>.node.ts`. For example, `MyNode.node.ts`.
  - n8n recommends including a codex file, containing metadata for your node. The codex filename must match the node base filename. For example, given a node base file named `MyNode.node.ts`, the codex name is `MyNode.node.json`.
  - The `nodes` directory can contain other files and subdirectories, including directories for versions, and node code split across more than one file to create a modular structure.
- A `credentials` directory, containing your credentials code. This code lives in a single credentials file. The filename format is `<node-name>.credentials.ts`. For example, `MyNode.credentials.ts`.

Modular structure

You can choose whether to place all your node's functionality in one file, or split it out into a base file and other modules, which the base file then imports. Unless your node is very simple, it's a best practice to split it out.

A basic pattern is to separate out operations. Refer to the HttpBin starter node for an example of this.

For more complex nodes, n8n recommends a directory structure. Refer to the Airtable node or Microsoft Outlook node as examples.

- `actions`: a directory containing sub-directories that represent resources.
  - Each sub-directory should contain two types of files:
    - An index file with resource description (named either `<resourceName>.resource.ts` or `index.ts`)

        - Files for operations `<operationName>.operation.ts`. These files should have two exports: `description` of the operation and an `execute` function.
- `methods`: an optional directory dynamic parameters' functions.
- `transport`: a directory containing the communication implementation.

Versioning

If your node has more than one version, and you're using full versioning, this makes the file structure more complex. You need a directory for each version, along with a base file that sets the default version. Refer to Node versioning for more information on working with versions, including types of versioning.

Decide how many nodes to include in a package

There are two possible setups when building a node:

- One node in one npm package.
- More than one node in a single npm package.

n8n supports both approaches. If you include more than one node, each node should have its own directory in the `nodes` directory.

A best-practice example for programmatic nodes

n8n's built-in Airtable node implements a modular structure and versioning, following recommended patterns.

Build a node

This section provides tutorials on building nodes. It covers:

- Tutorial: Build a declarative-style node
- Reference material on file structure, parameter definitions for base, codex, and credentials files, node UI elements, and more.

Coming soon:

- More tutorials
- Revised guidance on standards

Set up your development environment

This document lists the essential dependencies for developing a node, as well as guidance on setting up your editor.

Requirements

To build and test a node, you need:

- Node.js and npm. Minimum version Node 18.17.0. You can find instructions on how to install both using nvm (Node Version Manager) for Linux, Mac, and WSL (Windows Subsystem for Linux) here. For Windows users, refer to Microsoft's guide to Install NodeJS on Windows.
- A local instance of n8n. You can install n8n with `npm install n8n -g`, then follow the steps in Run your node locally to test your node.
- When building verified community nodes, you must use the `n8n-node` tool to create and test your node.

You should also have git installed. This allows you to clone and use the n8n-node-starter.

Editor setup

n8n recommends using VS Code as your editor.

Install these extensions:

- ESLint
- EditorConfig
- Prettier

By using VS Code and these extensions, you get access to the n8n node linter's warnings as you code.

Using the n8n-node tool

The `n8n-node` tool is the official CLI for developing community nodes for n8n. You can use it to scaffold out new nodes, build your projects, and run your node as you develop it.

Using `n8n-node`, you can create nodes that adhere to the guidelines for verified community nodes.

Get n8n-node

Run n8n-node without installing

You can create an `n8n-node` project directly without installing by using the `@n8n/create-node` initializer with your package manager:

```
npm create @n8n/node@latest
```

This sets up the initial project files locally (an alternative to installing `n8n-node` locally and explicitly running the new command). Afterward, you run the rest of the `n8n-node` commands through your package

manager's script runner inside the project directory (for example, `npm run dev`).

Install n8n-node globally

You can install `n8n-node` globally with `npm`:

```
npm install --global @n8n/node-cli
```

Verify access to the command by typing:

```
n8n-node --version
```

Command overview

The `n8n-node` tool provides the following commands:

new

The `new` command creates the file system structure and metadata for a new node. This command initializes the same structure as outlined in run n8n-node without installing.

When called, it interactively prompts for details about your project to customize your starting code. You'll provide the project name, choose a node type, and select the starting template that best matches your needs. The `n8n-node` tool will create your project file structure and optionally install your initial project dependencies.

Learn more about how to use the `new` command in the creating a new node section.

build

The `build` command compiles your node and copies all the required assets.

Learn more about how to use the `build` command in the building your node section.

dev

The `dev` command runs n8n with your node. It monitors your project directory and automatically rebuilds the live preview when it detects changes.

Learn more about how to use the `dev` command in the testing your node in n8n section.

lint

The `lint` command checks the code for the node in the current directory. You can optionally use with the `--fix` option to attempt to automatically fix any issues it identifies.

Learn more about how to use the `lint` command in the lint your node section.

release

The `release` command publishes your community node package to npm. It uses `release-it` to clean, check and cleanly build your package before publishing it to npm.

Learn more about how to use the `release` command in the release your node section.

Creating a new node

To create a new node with `n8n-node`, call `n8n-node new`. You can call this command entirely interactively or provide details on the command line.

The command will prompt for any missing information about your node and then generate a project structure to get you started. By default, it will follow up by installing the initial project dependencies (you can disable this by passing the `--skip-install` flag).

Setting node details interactively

When called without arguments, `n8n-node new` prompts you for details about your new node interactively:

```
n8n-node new
```

This will start an interactive prompt where you can define the details of your project:

- **What is your node called?** The name of your node. This impacts the name of your project directory, package name, and the n8n node itself. The name must use one of the following formats:
  - `n8n-nodes-<YOUR_NODE_NAME>`
  - `@<YOUR_ORG>/n8n-nodes-<YOUR_NODE_NAME>`
- **What kind of node are you building?** The node type you want to build:
  - **HTTP API**: A low-code, declarative node structure that's designed for faster approval for n8n Cloud.
  - **Other**: A programmatic style node with full flexibility.
- **What template do you want to use?** When using the HTTP API, you can choose the template to start from:
  - **GitHub Issues API**: A demo node that includes multiple operations and credentials. This can help you get familiar with the node structure and conventions.
  - **Start from scratch**: A blank template that will guide you through your custom setup with some further prompts.

When choosing HTTP API > Start from scratch, `n8n-node` will ask you the following:

- **What's the base URL of the API?** The root URL for the API you plan to integrate with.
- **What type of authentication does your API use?** The authentication your node should provide:
  - **API Key**: Send a secret key using headers, query parameters, or the request body.
  - **Bearer Token**: Send a token using the Authorization header (`Authorization: Bearer <token>`).

    - **OAuth2**: Use an OAuth 2.0 flow to get access tokens on behalf of a user or app.
    - **Basic Auth**: Send the base64-encoded username and password through Authorization headers.
    - **Custom**: Create your own credential logic. This will create an empty credential class that you can customize according to your needs.
    - **None**: No authentication necessary. Don't create a credential class for the node.

Once you've made your selections, `n8n-node` will create a new project directory for your node in the current directory. By default, it will also install the initial project dependencies (you can disable this by passing the `--skip-install` flag).

Providing node details on the command line

You can provide some of your node details on the command line to avoid prompts.

You can include the name you want to use for your node an argument:

```
n8n-node new n8n-nodes-myproject
```

If you know the template you want to use ahead of time, you can also pass the value using the `--template` flag:

```
n8n-node new --template declarative/custom
```

The template must be one of the following:

- `declarative/github-issues`: A demo node that includes multiple operations and credentials. This can help you get familiar with the node structure and conventions.
- `declarative/custom`: A blank template that will guide you through your custom setup with some further prompts.
- `programmatic/example`: A programmatic style node with full flexibility.

Building your node

You can build your node by running the `build` command in your project's root directory:

```
n8n-node build
```

`n8n-node` will compile your TypeScript files and bundle your other project assets. You can also call the `build` script from your package manager. For instance, if you're using `npm`, this works the same:

```
npm run build
```

Lint your node

The `n8n-node` tool automatically creates a `lint` script for your project as well. You can run with your package manager. For example:

```
n8n-node lint
```

You can also run through your package manager's script runner:

```
npm run lint
```

If you include the `--fix` option (also callable with `npm run lint:fix`), `n8n-node` will attempt to fix the issues that it identifies:

```
n8n-node lint --fix
```

Testing your node in n8n

To test your node in n8n, you run the `dev` command in your project's root directory:

```
n8n-node dev
```

As with the `build` command, you can also run this through your package manager. For example:

```
npm run dev
```

`n8n-node` will compile your project and then start up a local n8n instance through `npm` with your node loaded.

Visit your `localhost:5678` to sign in to your n8n instance. If you open a workflow, your node appears in the nodes panel:

[Image: node in nodes panel]

From there, you can add it to your workflow and test the node's functionality as you develop.

Release your node

To publish your node, run the `release` command in your project directory. This command uses `release-it` to build and publish your node.

```
n8n-node release
```

To run with `npm`, type:

```
npm run release
```

When you run the `release` command, `n8n-node` will perform the following actions:

- build the node
- run lint checks against your files
- update the changelog
- create git tags
- create a GitHub release
- publish the package to npm

Build a declarative-style node

This tutorial walks through building a declarative-style node. Before you begin, make sure this is the node style you need. Refer to Choose your node building approach for more information.

Prerequisites

You need the following installed on your development machine:

–8<– “_snippets/integrations/creating-nodes/prerequisites.md”

You need some understanding of:

- JavaScript/TypeScript
- REST APIs
- git

Build your node

In this section, you’ll clone n8n’s node starter repository, and build a node that integrates the NASA API. You’ll create a node that uses two of NASA’s services: APOD (Astronomy Picture of the Day) and Mars Rover Photos. To keep the code examples short, the node won’t implement every available option for the Mars Rover Photos endpoint.

Step 1: Set up the project

n8n provides a starter repository for node development. Using the starter ensures you have all necessary dependencies. It also provides a linter.

Clone the repository and navigate into the directory:

1. Generate a new repository from the template repository.
2. Clone your new repository: `shell git clone https://github.com/<your-organization>/<your-repo-name>.git n8n-nodes-nasa-pics cd n8n-nodes-nasa-pics`

The starter contains example nodes and credentials. Delete the following directories and files:

- `nodes/ExampleNode`
- `nodes/HTTPBin`
- `credentials/ExampleCredentials.credentials.ts`
- `credentials/HttpBinApi.credentials.ts`

Now create the following directories and files:

```
nodes/NasaPics
nodes/NasaPics/NasaPics.node.json
nodes/NasaPics/NasaPics.node.ts
credentials/NasaPicsApi.credentials.ts
```

These are the key files required for any node. Refer to Node file structure for more information on required files and recommended organization.

Now install the project dependencies:

```
npm i
```

Step 2: Add an icon

Save the NASA SVG logo from here as `nasapics.svg` in `nodes/NasaPics/`.

–8<– “_snippets/integrations/creating-nodes/node-icons.md”

Step 3: Create the node

Every node must have a base file. Refer to Node base file for detailed information about base file parameters.

In this example, the file is `NasaPics.node.ts`. To keep this tutorial short, you’ll place all the node functionality in this one file. When building more complex nodes, you should consider splitting out your functionality into modules. Refer to Node file structure for more information.

Step 3.1: Imports

Start by adding the import statements:

```
import { INodeType, INodeTypeDescription } from 'n8n-workflow';
```

Step 3.2: Create the main class

The node must export an interface that implements INodeType. This interface must include a `description` interface, which in turn contains the `properties` array.

```
export class NasaPics implements INodeType {
    description: INodeTypeDescription = {
        // Basic node details will go here
        properties: [
        // Resources and operations will go here
        ]
    };
}
```

Step 3.3: Add node details

All nodes need some basic parameters, such as their display name, icon, and the basic information for making a request using the node. Add the following to the `description`:

```
        displayName: 'NASA Pics',
        name: 'nasaPics',
        icon: 'file:nasapics.svg',
        group: ['transform'],
        version: 1,
        subtitle: '={{$parameter["operation"] + ": " +
$parameter["resource"]}}',
        description: 'Get data from NASAs API',
        defaults: {
            name: 'NASA Pics',
        },
        inputs: ['main'],
        outputs: ['main'],
        credentials: [
            {
```

```
            name: 'NasaPicsApi',
            required: true,
        },
    ],
    requestDefaults: {
        baseURL: 'https://api.nasa.gov',
        headers: {
            Accept: 'application/json',
            'Content-Type': 'application/json',
        },
    },
```

n8n uses some of the properties set in `description` to render the node in the Editor UI. These properties are `displayName`, `icon`, `description`, and `subtitle`.

Step 3.4: Add resources

The resource object defines the API resource that the node uses. In this tutorial, you're creating a node to access two of NASA's API endpoints: `planetary/apod` and `mars-photos`. This means you need to define two resource options in `NasaPics.node.ts`. Update the `properties` array with the resource object:

```
    properties: [
        {
            displayName: 'Resource',
            name: 'resource',
            type: 'options',
            noDataExpression: true,
            options: [
                {
                    name: 'Astronomy Picture of the Day',
                    value: 'astronomyPictureOfTheDay',
                },
                {
                    name: 'Mars Rover Photos',
                    value: 'marsRoverPhotos',
                },
            ],
            default: 'astronomyPictureOfTheDay',
        },
        // Operations will go here

    ]
```

`type` controls which UI element n8n displays for the resource, and tells n8n what type of data to expect from the user. `options` results in n8n adding a dropdown that allows users to choose one option. Refer to Node UI elements for more information.

Step 3.5: Add operations

The operations object defines the available operations on a resource.

In a declarative-style node, the operations object includes `routing` (within the `options` array). This sets up the details of the API call.

Add the following to the `properties` array, after the `resource` object:

```
        {
```

```
        displayName: 'Operation',
        name: 'operation',
        type: 'options',
        noDataExpression: true,
        displayOptions: {
            show: {
                resource: [
                    'astronomyPictureOfTheDay',
                ],
            },
        },
        options: [
            {
                name: 'Get',
                value: 'get',
                action: 'Get the APOD',
                description: 'Get the Astronomy Picture of the day',
                routing: {
                    request: {
                        method: 'GET',
                        url: '/planetary/apod',
                    },
                },
            },
        ],
        default: 'get',
    },
    {
        displayName: 'Operation',
        name: 'operation',
        type: 'options',
        noDataExpression: true,
        displayOptions: {
            show: {
                resource: [
                    'marsRoverPhotos',
                ],
            },
        },
        options: [
            {
                name: 'Get',
                value: 'get',
                action: 'Get Mars Rover photos',
                description: 'Get photos from the Mars Rover',
                routing: {
                    request: {
                        method: 'GET',
                    },
                },
            },
        ],
        default: 'get',
    },
    {
        displayName: 'Rover name',
        description: 'Choose which Mars Rover to get a photo from',
        required: true,
        name: 'roverName',
        type: 'options',
```

```
            options: [
                {name: 'Curiosity', value: 'curiosity'},
                {name: 'Opportunity', value: 'opportunity'},
                {name: 'Perseverance', value: 'perseverance'},
                {name: 'Spirit', value: 'spirit'},
            ],
            routing: {
                request: {
                    url: '=/mars-photos/api/v1/rovers/{{$value}}/photos',
                },
            },
            default: 'curiosity',
            displayOptions: {
                show: {
                    resource: [
                        'marsRoverPhotos',
                    ],
                },
            },
        },
        {
            displayName: 'Date',
            description: 'Earth date',
            required: true,
            name: 'marsRoverDate',
            type: 'dateTime',
            default:'',
            displayOptions: {
                show: {
                    resource: [
                        'marsRoverPhotos',
                    ],
                },
            },
            routing: {
                request: {
                    // You've already set up the URL. qs appends the value
of the field as a query string
                    qs: {
                        earth_date: '={{ new
Date($value).toISOString().substr(0,10) }}',
                    },
                },
            },
        },
        // Optional/additional fields will go here
```

This code creates two operations: one to get today's APOD image, and another to send a get request for photos from one of the Mars Rovers. The object named `roverName` requires the user to choose which Rover they want photos from. The `routing` object in the Mars Rover operation references this to create the URL for the API call.

Step 3.6: Optional fields

Most APIs, including the NASA API that you're using in this example, have optional fields you can use to refine your query.

To avoid overwhelming users, n8n displays these under **Additional Fields** in the UI.

For this tutorial, you'll add one additional field, to allow users to pick a date to use with the APOD endpoint. Add the following to the properties array:

```
        {
            displayName: 'Additional Fields',
            name: 'additionalFields',
            type: 'collection',
            default: {},
            placeholder: 'Add Field',
            displayOptions: {
                show: {
                    resource: [
                        'astronomyPictureOfTheDay',
                    ],
                    operation: [
                        'get',
                    ],
                },
            },
            options: [
                {
                    displayName: 'Date',
                    name: 'apodDate',
                    type: 'dateTime',
                    default: '',
                    routing: {
                        request: {
                            // You've already set up the URL. qs appends the
value of the field as a query string
                            qs: {
                                date: '={{ new
Date($value).toISOString().substr(0,10) }}',
                            },
                        },
                    },
                },
            ],
        }
```

Step 4: Set up authentication

The NASA API requires users to authenticate with an API key.

Add the following to `nasaPicsApi.credentials.ts`:

```
        import {
            IAuthenticateGeneric,
            ICredentialType,
            INodeProperties,
        } from 'n8n-workflow';

        export class NasaPicsApi implements ICredentialType {
            name = 'NasaPicsApi';
            displayName = 'NASA Pics API';
            // Uses the link to this tutorial as an example
            // Replace with your own docs links when building your own nodes
            documentationUrl = 'https://docs.n8n.io/integrations/creating-
nodes/build/declarative-style-node/';
            properties: INodeProperties[] = [
```

```
            {
                displayName: 'API Key',
                name: 'apiKey',
                type: 'string',
                default: '',
            },
        ];
        authenticate = {
            type: 'generic',
            properties: {
                qs: {
                    'api_key': '={{$credentials.apiKey}}'
                }
            },
        } as IAuthenticateGeneric;
    }
```

For more information about credentials files and options, refer to Credentials file.

Step 5: Add node metadata

Metadata about your node goes in the JSON file at the root of your node. n8n refers to this as the codex file. In this example, the file is `NasaPics.node.json`.

Add the following code to the JSON file:

```
{
    "node": "n8n-nodes-base.NasaPics",
    "nodeVersion": "1.0",
    "codexVersion": "1.0",
    "categories": [
        "Miscellaneous"
    ],
    "resources": {
        "credentialDocumentation": [
            {
                "url": ""
            }
        ],
        "primaryDocumentation": [
            {
                "url": ""
            }
        ]
    }
}
```

For more information on these parameters, refer to Node codex files.

Step 6: Update the npm package details

Your npm package details are in the `package.json` at the root of the project. It's essential to include the `n8n` object with links to the credentials and base node file. Update this file to include the following information:

```
{
    // All node names must start with "n8n-nodes-"
```

```
            "name": "n8n-nodes-nasapics",
            "version": "0.1.0",
            "description": "n8n node to call NASA's APOD and Mars Rover
Photo services.",
            "keywords": [
                // This keyword is required for community nodes
                "n8n-community-node-package"
            ],
            "license": "MIT",
            "homepage": "https://n8n.io",
            "author": {
                "name": "Test",
                "email": "test@example.com"
            },
            "repository": {
                "type": "git",
                // Change the git remote to your own repository
                // Add the new URL here
                "url": "git+<your-repo-url>"
            },
            "main": "index.js",
            "scripts": {
                // don't change
            },
            "files": [
                "dist"
            ],
            // Link the credentials and node
            "n8n": {
                "n8nNodesApiVersion": 1,
                "credentials": [
                    "dist/credentials/NasaPicsApi.credentials.js"
                ],
                "nodes": [
                    "dist/nodes/NasaPics/NasaPics.node.js"
                ]
            },
            "devDependencies": {
                // don't change
            },
            "peerDependencies": {
                // don't change
            }
        }
```

You need to update the `package.json` to include your own information, such as your name and repository URL. For more information on npm `package.json` files, refer to npm's package.json documentation.

Test your node

–8<– "_snippets/integrations/creating-nodes/testing.md"

Next steps

- Deploy your node.
- View an example of a declarative node: n8n's Brevo node. Note that the main node is declarative, while the trigger node is in

programmatic style.
- Learn about node versioning.

Build a programmatic-style node

This tutorial walks through building a programmatic-style node. Before you begin, make sure this is the node style you need. Refer to Choose your node building approach for more information.

Prerequisites

You need the following installed on your development machine:

–8<– “_snippets/integrations/creating-nodes/prerequisites.md”

You need some understanding of:

- JavaScript/TypeScript
- REST APIs
- git
- Expressions in n8n

Build your node

In this section, you’ll clone n8n’s node starter repository, and build a node that integrates the SendGrid. You’ll create a node that implements one piece of SendGrid functionality: create a contact.

Step 1: Set up the project

n8n provides a starter repository for node development. Using the starter ensures you have all necessary dependencies. It also provides a linter.

Clone the repository and navigate into the directory:

1. Generate a new repository from the template repository.
2. Clone your new repository: `shell git clone https://github.com/<your-organization>/<your-repo-name>.git n8n-nodes-friendgrid cd n8n-nodes-friendgrid`

The starter contains example nodes and credentials. Delete the following directories and files:

- `nodes/ExampleNode`
- `nodes/HTTPBin`
- `credentials/ExampleCredentials.credentials.ts`
- `credentials/HttpBinApi.credentials.ts`

Now create the following directories and files:

```
nodes/FriendGrid
nodes/FriendGrid/FriendGrid.node.json
nodes/FriendGrid/FriendGrid.node.ts
credentials/FriendGridApi.credentials.ts
```

These are the key files required for any node. Refer to Node file structure for more information on required files and recommended organization.

Now install the project dependencies:

```
npm i
```

Step 2: Add an icon

Save the SendGrid SVG logo from here as `friendGrid.svg` in `nodes/FriendGrid/`.

–8<– "_snippets/integrations/creating-nodes/node-icons.md"

Step 3: Define the node in the base file

Every node must have a base file. Refer to Node base file for detailed information about base file parameters.

In this example, the file is `FriendGrid.node.ts`. To keep this tutorial short, you'll place all the node functionality in this one file. When building more complex nodes, you should consider splitting out your functionality into modules. Refer to Node file structure for more information.

Step 3.1: Imports

Start by adding the import statements:

```
import {
    IExecuteFunctions,
} from 'n8n-core';

import {
    IDataObject,
    INodeExecutionData,
    INodeType,
    INodeTypeDescription,
    NodeConnectionType
} from 'n8n-workflow';

import {
    OptionsWithUri,
} from 'request';
```

Step 3.2: Create the main class

The node must export an interface that implements `INodeType`. This interface must include a `description` interface, which in turn contains the `properties` array.

```
export class FriendGrid implements INodeType {
    description: INodeTypeDescription = {
        // Basic node details will go here
        properties: [
            // Resources and operations will go here
        ],
    };
```

```
            // The execute method will go here
            async execute(this: IExecuteFunctions):
Promise<INodeExecutionData[][]> {
            }
        }
```

Step 3.3: Add node details

All programmatic nodes need some basic parameters, such as their display name and icon. Add the following to the `description`:

```
        displayName: 'FriendGrid',
        name: 'friendGrid',
        icon: 'file:friendGrid.svg',
        group: ['transform'],
        version: 1,
        description: 'Consume SendGrid API',
        defaults: {
            name: 'FriendGrid',
        },
        inputs: [NodeConnectionType.Main],
        outputs: [NodeConnectionType.Main],
        credentials: [
            {
                name: 'friendGridApi',
                required: true,
            },
        ],
```

n8n uses some of the properties set in `description` to render the node in the Editor UI. These properties are `displayName`, `icon`, and `description`.

Step 3.4: Add the resource

The resource object defines the API resource that the node uses. In this tutorial, you're creating a node to access one of SendGrid's API endpoints: `/v3/marketing/contacts`. This means you need to define a resource for this endpoint. Update the `properties` array with the resource object:

```
        {
            displayName: 'Resource',
            name: 'resource',
            type: 'options',
            options: [
                {
                    name: 'Contact',
                    value: 'contact',
                },
            ],
            default: 'contact',
            noDataExpression: true,
            required: true,
            description: 'Create a new contact',
        },
```

`type` controls which UI element n8n displays for the resource, and tells n8n what type of data to expect from the user. `options` results in n8n adding a dropdown that allows users to choose one option. Refer to

Node UI elements for more information.

Step 3.5: Add operations

The operations object defines what you can do with a resource. It usually relates to REST API verbs (GET, POST, and so on). In this tutorial, there's one operation: create a contact. It has one required field, the email address for the contact the user creates.

Add the following to the `properties` array, after the `resource` object:

```
{
    displayName: 'Operation',
    name: 'operation',
    type: 'options',
    displayOptions: {
        show: {
            resource: [
                'contact',
            ],
        },
    },
    options: [
        {
            name: 'Create',
            value: 'create',
            description: 'Create a contact',
            action: 'Create a contact',
        },
    ],
    default: 'create',
    noDataExpression: true,
},
{
    displayName: 'Email',
    name: 'email',
    type: 'string',
    required: true,
    displayOptions: {
        show: {
            operation: [
                'create',
            ],
            resource: [
                'contact',
            ],
        },
    },
    default:'',
    placeholder: 'name@email.com',
    description:'Primary email for the contact',
},
```

Step 3.6: Add optional fields

Most APIs, including the SendGrid API that you're using in this example, have optional fields you can use to refine your query.

To avoid overwhelming users, n8n displays these under **Additional Fields** in the UI.

For this tutorial, you'll add two additional fields, to allow users to enter the contact's first name and last name. Add the following to the properties array:

```
{
    displayName: 'Additional Fields',
    name: 'additionalFields',
    type: 'collection',
    placeholder: 'Add Field',
    default: {},
    displayOptions: {
        show: {
            resource: [
                'contact',
            ],
            operation: [
                'create',
            ],
        },
    },
    options: [
        {
            displayName: 'First Name',
            name: 'firstName',
            type: 'string',
            default: '',
        },
        {
            displayName: 'Last Name',
            name: 'lastName',
            type: 'string',
            default: '',
        },
    ],
},
```

Step 4: Add the execute method

You've set up the node UI and basic information. It's time to map the node UI to API requests, and make the node actually do something.

The `execute` method runs every time the node runs. In this method, you have access to the input items and to the parameters that the user set in the UI, including the credentials.

Add the following the `execute` method in the `FriendGrid.node.ts`:

```
// Handle data coming from previous nodes
const items = this.getInputData();
let responseData;
const returnData = [];
const resource = this.getNodeParameter('resource', 0) as string;
const operation = this.getNodeParameter('operation', 0) as string;

// For each item, make an API call to create a contact
for (let i = 0; i < items.length; i++) {
    if (resource === 'contact') {
        if (operation === 'create') {
            // Get email input
            const email = this.getNodeParameter('email', i) as
```

```
string;
                    // Get additional fields input
                    const additionalFields =
this.getNodeParameter('additionalFields', i) as IDataObject;
                    const data: IDataObject = {
                        email,
                    };

                    Object.assign(data, additionalFields);

                    // Make HTTP request according to
https://sendgrid.com/docs/api-reference/
                    const options: OptionsWithUri = {
                        headers: {
                            'Accept': 'application/json',
                        },
                        method: 'PUT',
                        body: {
                            contacts: [
                                data,
                            ],
                        },
                        uri:
`https://api.sendgrid.com/v3/marketing/contacts`,
                        json: true,
                    };
                    responseData = await
this.helpers.requestWithAuthentication.call(this, 'friendGridApi',
options);
                    returnData.push(responseData);
                }
            }
        }
        // Map data to n8n data structure
        return [this.helpers.returnJsonArray(returnData)];
```

Note the following lines of this code:

```
        const items = this.getInputData();
        ...
        for (let i = 0; i < items.length; i++) {
            ...
            const email = this.getNodeParameter('email', i) as string;
            ...
        }
```

Users can provide data in two ways:

- Entered directly in the node fields
- By mapping data from earlier nodes in the workflow

`getInputData()`, and the subsequent loop, allows the node to handle situations where data comes from a previous node. This includes supporting multiple inputs. This means that if, for example, the previous node outputs contact information for five people, your FriendGrid node can create five contacts.

Step 5: Set up authentication

The SendGrid API requires users to authenticate with an API key.

Add the following to `FriendGridApi.credentials.ts`

```
import {
    IAuthenticateGeneric,
    ICredentialTestRequest,
    ICredentialType,
    INodeProperties,
} from 'n8n-workflow';

export class FriendGridApi implements ICredentialType {
    name = 'friendGridApi';
    displayName = 'FriendGrid API';
    properties: INodeProperties[] = [
        {
            displayName: 'API Key',
            name: 'apiKey',
            type: 'string',
            default: '',
        },
    ];

    authenticate: IAuthenticateGeneric = {
        type: 'generic',
        properties: {
            headers: {
                Authorization: '=Bearer {{$credentials.apiKey}}',
            },
        },
    };

    test: ICredentialTestRequest = {
        request: {
            baseURL: 'https://api.sendgrid.com/v3',
            url: '/marketing/contacts',
        },
    };
}
```

For more information about credentials files and options, refer to Credentials file.

Step 6: Add node metadata

Metadata about your node goes in the JSON file at the root of your node. n8n refers to this as the codex file. In this example, the file is `FriendGrid.node.json`.

Add the following code to the JSON file:

```
{
    "node": "n8n-nodes-base.FriendGrid",
    "nodeVersion": "1.0",
    "codexVersion": "1.0",
    "categories": [
        "Miscellaneous"
    ],
    "resources": {
        "credentialDocumentation": [
            {
                "url": ""
```

```
            }
        ],
        "primaryDocumentation": [
            {
                "url": ""
            }
        ]
    }
}
```

For more information on these parameters, refer to Node codex files.

Step 7: Update the npm package details

Your npm package details are in the `package.json` at the root of the project. It's essential to include the `n8n` object with links to the credentials and base node file. Update this file to include the following information:

```
{
    // All node names must start with "n8n-nodes-"
    "name": "n8n-nodes-friendgrid",
    "version": "0.1.0",
    "description": "n8n node to create contacts in SendGrid",
    "keywords": [
        // This keyword is required for community nodes
        "n8n-community-node-package"
    ],
    "license": "MIT",
    "homepage": "https://n8n.io",
    "author": {
        "name": "Test",
        "email": "test@example.com"
    },
    "repository": {
        "type": "git",
        // Change the git remote to your own repository
        // Add the new URL here
        "url": "git+<your-repo-url>"
    },
    "main": "index.js",
    "scripts": {
        // don't change
    },
    "files": [
        "dist"
    ],
    // Link the credentials and node
    "n8n": {
        "n8nNodesApiVersion": 1,
        "credentials": [
            "dist/credentials/FriendGridApi.credentials.js"
        ],
        "nodes": [
            "dist/nodes/FriendGrid/FriendGrid.node.js"
        ]
    },
    "devDependencies": {
        // don't change
    },
```

```
        "peerDependencies": {
            // don't change
        }
    }
```

You need to update the `package.json` to include your own information, such as your name and repository URL. For more information on npm `package.json` files, refer to npm's package.json documentation.

Test your node

–8<– "_snippets/integrations/creating-nodes/testing.md"

Next steps

- Deploy your node.
- View an example of a programmatic node: n8n's Mattermost node. This is an example of a more complex programmatic node structure.
- Learn about node versioning.
- Make sure you understand key concepts: item linking and data structures.

Node building reference

This section contains reference information, including details about:

- Node UI elements
- Organizing your node files
- Key parameters in your node's base file and credentials file.
- UX guidelines and verification guidelines for submitting your node for verification by n8n.

Node user interface elements

n8n provides a set of predefined UI components (based on a JSON file) that allows users to input all sorts of data types. The following UI elements are available in n8n.

String

Basic configuration:

```
    {
        displayName: Name, // The value the user sees in the UI
        name: name, // The name used to reference the element UI within
the code
        type: string,
        required: true, // Whether the field is required or not
        default: 'n8n',
        description: 'The name of the user',
```

```
            displayOptions: { // the resources and operations to display
this element with
                show: {
                    resource: [
                        // comma-separated list of resource names
                    ],
                    operation: [
                        // comma-separated list of operation names
                    ]
                }
            },
        }
```

[Image: String]

String field for inputting passwords:

```
        {
            displayName: 'Password',
            name: 'password',
            type: 'string',
            required: true,
            typeOptions: {
                password: true,
            },
            default: '',
            description: `User's password`,
            displayOptions: { // the resources and operations to display
this element with
                show: {
                    resource: [
                        // comma-separated list of resource names
                    ],
                    operation: [
                        // comma-separated list of operation names
                    ]
                }
            },
        }
```

[Image: Password]

String field with more than one row:

```
        {
            displayName: 'Description',
            name: 'description',
            type: 'string',
            required: true,
            typeOptions: {
                rows: 4,
            },
            default: '',
            description: 'Description',
            displayOptions: { // the resources and operations to display
this element with
                show: {
                    resource: [
                        // comma-separated list of resource names
                    ],
                    operation: [
```

```
                    // comma-separated list of operation names
                ]
            }
        },
    }
```

[Image: Multiple rows]

Support drag and drop for data keys

Users can drag and drop data values to map them to fields. Dragging and dropping creates an expression to load the data value. n8n supports this automatically.

You need to add an extra configuration option to support dragging and dropping data keys:

- `requiresDataPath: 'single'`: for fields that require a single string.
- `requiresDataPath: 'multiple'`: for fields that can accept a comma-separated list of string.

The Compare Datasets node code has examples.

Number

Number field with decimal points:

```
    {
        displayName: 'Amount',
        name: 'amount',
        type: 'number',
        required: true,
        typeOptions: {
            maxValue: 10,
            minValue: 0,
            numberPrecision: 2,
        },
        default: 10.00,
        description: 'Your current amount',
        displayOptions: { // the resources and operations to display
this element with
            show: {
                resource: [
                    // comma-separated list of resource names
                ],
                operation: [
                    // comma-separated list of operation names
                ]
            }
        },
    }
```

[Image: Decimal]

Collection

Use the `collection` type when you need to display optional fields.

```
        {
            displayName: 'Filters',
            name: 'filters',
            type: 'collection',
            placeholder: 'Add Field',
            default: {},
            options: [
                {
                    displayName: 'Type',
                    name: 'type',
                    type: 'options',
                    options: [
                        {
                            name: 'Automated',
                            value: 'automated',
                        },
                        {
                            name: 'Past',
                            value: 'past',
                        },
                        {
                            name: 'Upcoming',
                            value: 'upcoming',
                        },
                    ],
                    default: '',
                },
            ],
            displayOptions: { // the resources and operations to display
this element with
                show: {
                    resource: [
                        // comma-separated list of resource names
                    ],
                    operation: [
                        // comma-separated list of operation names
                    ]
                }
            },
        }
```

[Image: Collection]

DateTime

The `dateTime` type provides a date picker.

```
        {
            displayName: 'Modified Since',
            name: 'modified_since',
            type: 'dateTime',
            default: '',
            description: 'The date and time when the file was last
modified',
            displayOptions: { // the resources and operations to display
this element with
                show: {
                    resource: [
                        // comma-separated list of resource names
```

```
                ],
                operation: [
                    // comma-separated list of operation names
                ]
            }
        },
    }
```

[Image: DateTime]

Boolean

The `boolean` type adds a toggle for entering true or false.

```
    {
        displayName: 'Wait for Image',
        name: 'waitForImage',
        type: 'boolean',
        default: true, // Initial state of the toggle
        description: 'Whether to wait for the image or not',
        displayOptions: { // the resources and operations to display
this element with
            show: {
                resource: [
                    // comma-separated list of resource names
                ],
                operation: [
                    // comma-separated list of operation names
                ]
            }
        },
    }
```

[Image: Boolean]

Color

The `color` type provides a color selector.

```
    {
        displayName: 'Background Color',
        name: 'backgroundColor',
        type: 'color',
        default: '', // Initially selected color
        displayOptions: { // the resources and operations to display
this element with
            show: {
                resource: [
                    // comma-separated list of resource names
                ],
                operation: [
                    // comma-separated list of operation names
                ]
            }
        },
    }
```

[Image: Color]

Options

The `options` type adds an options list. Users can select a single value.

```
        {
            displayName: 'Resource',
            name: 'resource',
            type: 'options',
            options: [
                {
                    name: 'Image',
                    value: 'image',
                },
                {
                    name: 'Template',
                    value: 'template',
                },
            ],
            default: 'image', // The initially selected option
            description: 'Resource to consume',
            displayOptions: { // the resources and operations to display
this element with
                show: {
                    resource: [
                        // comma-separated list of resource names
                    ],
                    operation: [
                        // comma-separated list of operation names
                    ]
                }
            },
        }
```

[Image: Options]

Multi-options

The `multiOptions` type adds an options list. Users can select more than one value.

```
        {
            displayName: 'Events',
            name: 'events',
            type: 'multiOptions',
            options: [
                {
                    name: 'Plan Created',
                    value: 'planCreated',
                },
                {
                    name: 'Plan Deleted',
                    value: 'planDeleted',
                },
            ],
            default: [], // Initially selected options
            description: 'The events to be monitored',
            displayOptions: { // the resources and operations to display
this element with
                show: {
```

```
                    resource: [
                        // comma-separated list of resource names
                    ],
                    operation: [
                        // comma-separated list of operation names
                    ]
                }
            },
        }
```

[Image: Multi-options]

Filter

Use this component to evaluate, match, or filter incoming data.

This is the code from n8n's own If node. It shows a filter component working with a collection component where users can configure the filter's behavior.

```
        {
            displayName: 'Conditions',
            name: 'conditions',
            placeholder: 'Add Condition',
            type: 'filter',
            default: {},
            typeOptions: {
                filter: {
                    // Use the user options (below) to determine filter
behavior
                    caseSensitive: '={{!$parameter.options.ignoreCase}}',
                    typeValidation: '=
{{$parameter.options.looseTypeValidation ? "loose" : "strict"}}',
                },
            },
        },
        {
        displayName: 'Options',
        name: 'options',
        type: 'collection',
        placeholder: 'Add option',
        default: {},
        options: [
            {
                displayName: 'Ignore Case',
                description: 'Whether to ignore letter case when evaluating
conditions',
                name: 'ignoreCase',
                type: 'boolean',
                default: true,
            },
            {
                displayName: 'Less Strict Type Validation',
                description: 'Whether to try casting value types based on
the selected operator',
                name: 'looseTypeValidation',
                type: 'boolean',
                default: true,
            },
```

```
        ],
        },
```

[Image: Filter]

Assignment collection (drag and drop)

Use the drag and drop component when you want users to pre-fill name and value parameters with a single drag interaction.

```
        {
            displayName: 'Fields to Set',
            name: 'assignments',
            type: 'assignmentCollection',
            default: {},
        },
```

You can see an example in n8n's Edit Fields (Set) node:

[Image: A gif showing the drag and drop action, as well as changing a field to fixed]

Fixed collection

Use the `fixedCollection` type to group fields that are semantically related.

```
        {
            displayName: 'Metadata',
            name: 'metadataUi',
            placeholder: 'Add Metadata',
            type: 'fixedCollection',
            default: '',
            typeOptions: {
                multipleValues: true,
            },
            description: '',
            options: [
                {
                    name: 'metadataValues',
                    displayName: 'Metadata',
                    values: [
                        {
                            displayName: 'Name',
                            name: 'name',
                            type: 'string',
                            default: 'Name of the metadata key to add.',
                        },
                        {
                            displayName: 'Value',
                            name: 'value',
                            type: 'string',
                            default: '',
                            description: 'Value to set for the metadata
key.',
                        },
                    ],
                },
```

```
            ],
            displayOptions: { // the resources and operations to display
this element with
                show: {
                    resource: [
                        // comma-separated list of resource names
                    ],
                    operation: [
                        // comma-separated list of operation names
                    ]
                }
            },
        }
```

[Image: Fixed collection]

Resource locator

[Image: Resource locator]

The resource locator element helps users find a specific resource in an external service, such as a card or label in Trello.

The following options are available:

- ID
- URL
- List: allows users to select or search from a prepopulated list. This option requires more coding, as you must populate the list, and handle searching if you choose to support it.

You can choose which types to include.

Example:

```
        {
            displayName: 'Card',
            name: 'cardID',
            type: 'resourceLocator',
            default: '',
            description: 'Get a card',
            modes: [
                {
                    displayName: 'ID',
                    name: 'id',
                    type: 'string',
                    hint: 'Enter an ID',
                    validation: [
                        {
                            type: 'regex',
                            properties: {
                                regex: '^[0-9]',
                                errorMessage: 'The ID must start with a
number',
                            },
                        },
                    ],
                    placeholder: '12example',
                    // How to use the ID in API call
                    url: '=http://api-base-url.com/?id={{$value}}',
```

```
                },
                {
                    displayName: 'URL',
                    name: 'url',
                    type: 'string',
                    hint: 'Enter a URL',
                    validation: [
                        {
                            type: 'regex',
                            properties: {
                                regex: '^http',
                                errorMessage: 'Invalid URL',
                            },
                        },
                    ],
                    placeholder: 'https://example.com/card/12example/',
                    // How to get the ID from the URL
                    extractValue: {
                        type: 'regex',
                        regex: 'example.com/card/([0-9]*.*)/',
                    },
                },
                {
                    displayName: 'List',
                    name: 'list',
                    type: 'list',
                    typeOptions: {
                        // You must always provide a search method
                        // Write this method within the methods object in
your base file
                        // The method must populate the list, and handle
searching if searchable: true
                        searchListMethod: 'searchMethod',
                        // If you want users to be able to search the list
                        searchable: true,
                        // Set to true if you want to force users to search
                        // When true, users can't browse the list
                        // Or false if users can browse a list
                        searchFilterRequired: true,
                    },
                },
            ],
            displayOptions: {
                // the resources and operations to display this element with
                show: {
                    resource: [
                        // comma-separated list of resource names
                    ],
                    operation: [
                        // comma-separated list of operation names
                    ],
                },
            },
        },
```

Refer to the following for live examples:

- Refer to `CardDescription.ts` and `Trello.node.ts` in n8n's Trello node for an example of a list with search that includes `searchFilterRequired: true`.
- Refer to `GoogleDrive.node.ts` for an example where users can

browse the list or search.

Resource mapper

If your node performs insert, update, or upsert operations, you need to send data from the node in a format supported by the service you're integrating with. A common pattern is to use a Set node before the node that sends data, to convert the data to match the schema of the service you're connecting to. The resource mapper UI component provides a way to get data into the required format directly within the node, rather than using a Set node. The resource mapper component can also validate input data against the schema provided in the node, and cast input data into the expected type.

```
        {
            displayName: 'Columns',
            name: 'columns', // The name used to reference the element UI
within the code
            type: 'resourceMapper', // The UI element type
            default: {
                // mappingMode can be defined in the component (mappingMode:
'defineBelow')
                // or you can attempt automatic mapping (mappingMode:
'autoMapInputData')
                mappingMode: 'defineBelow',
                // Important: always set default value to null
                value: null,
            },
            required: true,
            // See "Resource mapper type options interface" below for the
full typeOptions specification
            typeOptions: {
                resourceMapper: {
                    resourceMapperMethod: 'getMappingColumns',
                    mode: 'update',
                    fieldWords: {
                        singular: 'column',
                        plural: 'columns',
                    },
                    addAllFields: true,
                    multiKeyMatch: true,
                    supportAutoMap: true,
                    matchingFieldsLabels: {
                        title: 'Custom matching columns title',
                        description: 'Help text for custom matching
columns',
                        hint: 'Below-field hint for custom matching
columns',
                    },
                },
            },
        },
```

Refer to the Postgres node (version 2) for a live example using a database schema.

Refer to the Google Sheets node (version 2) for a live example using a schema-less service.

Resource mapper type options interface

The `typeOptions` section must implement the following interface:

```
        export interface ResourceMapperTypeOptions {
            // The name of the method where you fetch the schema
            // Refer to the Resource mapper method section for more detail
            resourceMapperMethod: string;
            // Choose the mode for your operation
            // Supported modes: add, update, upsert
            mode: 'add' | 'update' | 'upsert';
            // Specify labels for fields in the UI
            fieldWords?: { singular: string; plural: string };
            // Whether n8n should display a UI input for every field when
node first added to workflow
            // Default is true
            addAllFields?: boolean;
            // Specify a message to show if no fields are fetched from the
service
            // (the call is successful but the response is empty)
            noFieldsError?: string;
            // Whether to support multi-key column matching
            // multiKeyMatch is for update and upsert only
            // Default is false
            // If true, the node displays a multi-select dropdown for the
matching column selector
            multiKeyMatch?: boolean;
            // Whether to support automatic mapping
            // If false, n8n hides the mapping mode selector field and sets
mappingMode to defineBelow
            supportAutoMap?: boolean;
            // Custom labels for the matching columns selector
            matchingFieldsLabels?: {
                title?: string;
                description?: string;
                hint?: string;
            };
        }
```

Resource mapper method

This method contains your node-specific logic for fetching the data schema. Every node must implement its own logic for fetching the schema, and setting up each UI field according to the schema.

It must return a value that implements the `ResourceMapperFields` interface:

```
        interface ResourceMapperField {
            // Field ID as in the service
            id: string;
            // Field label
            displayName: string;
            // Whether n8n should pre-select the field as a matching field
            // A matching field is a column used to identify the rows to
modify
            defaultMatch: boolean;
            // Whether the field can be used as a matching field
            canBeUsedToMatch?: boolean;
            // Whether the field is required by the schema
```

```
            required: boolean;
            // Whether to display the field in the UI
            // If false, can't be used for matching or mapping
            display: boolean;
            // The data type for the field
            // These correspond to UI element types
            // Supported types: string, number, dateTime, boolean, time,
array, object, options
            type?: FieldType;
            // Added at runtime if the field is removed from mapping by the
user
            removed?: boolean;
            // Specify options for enumerated types
            options?: INodePropertyOptions[];
        }
```

Refer to the Postgres resource mapping method and Google Sheets resource mapping method for live examples.

JSON

```
        {
            displayName: 'Content (JSON)',
            name: 'content',
            type: 'json',
            default: '',
            description: '',
            displayOptions: { // the resources and operations to display
this element with
                show: {
                    resource: [
                        // comma-separated list of resource names
                    ],
                    operation: [
                        // comma-separated list of operation names
                    ]
                }
            },
        }
```

[Image: JSON]

HTML

The HTML editor allows users to create HTML templates in their workflows. The editor supports standard HTML, CSS in `<style>` tags, and expressions wrapped in `{{}}`. Users can add `<script>` tags to pull in additional JavaScript. n8n doesn't run this JavaScript during workflow execution.

```
        {
            displayName: 'HTML Template', // The value the user sees in the
UI
            name: 'html', // The name used to reference the element UI
within the code
            type: 'string',
            typeOptions: {
                editor: 'htmlEditor',
```

```
            },
            default: placeholder, // Loads n8n's placeholder HTML template
            noDataExpression: true, // Prevent using an expression for the
field
            description: 'HTML template to render',
        },
```

Refer to `Html.node.ts` for a live example.

Notice

Display a yellow box with a hint or extra info. Refer to Node UI design for guidance on writing good hints and info text.

```
        {
          displayName: 'Your text here',
          name: 'notice',
          type: 'notice',
          default: '',
        },
```

[Image: Notice]

Hints

There are two types of hints: parameter hints and node hints:

- Parameter hints are small lines of text below a user input field.
- Node hints are a more powerful and flexible option than Notice. Use them to display longer hints, in the input panel, output panel, or node details view.

Add a parameter hint

Add the `hint` parameter to a UI element:

```
        {
            displayName: 'URL',
            name: 'url',
            type: 'string',
            hint: 'Enter a URL',
            ...
        }
```

Add a node hint

Define the node's hints in the `hints` property within the node `description`:

```
        description: INodeTypeDescription = {
            ...
            hints: [
                {
                    // The hint message. You can use HTML.
                    message: "This node has many input items. Consider
enabling <b>Execute Once</b> in the node\'s settings.",
                    // Choose from: info, warning, danger. The default is
```

```
'info'.
                    // Changes the color. info (grey), warning (yellow),
danger (red)
                    type: 'info',
                    // Choose from: inputPane, outputPane, ndv. By default
n8n displays the hint in both the input and output panels.
                    location: 'outputPane',
                    // Choose from: always, beforeExecution, afterExecution.
The default is 'always'
                    whenToDisplay: 'beforeExecution',
                    // Optional. An expression. If it resolves to true, n8n
displays the message. Defaults to true.
                    displayCondition: '={{ $parameter["operation"] ===
"select" && $input.all().length > 1 }}'
                }
            ]
            ...
        }
```

Add a dynamic hint to a programmatic-style node

In programmatic-style nodes you can create a dynamic message that includes information from the node execution. As it relies on the node output data, you can't display this type of hint until after execution.

```
        if (operation === 'select' && items.length > 1 && !node.executeOnce)
{
            // Expects two parameters: NodeExecutionData and an array of
hints
            return new NodeExecutionOutput(
                [returnData],
                [
                    {
                        message: `This node ran ${items.length} times, once
for each input item. To run for the first item only, enable
<b>Execute once</b> in the node settings.`,
                        location: 'outputPane',
                    },
                ],
            );
        }
        return [returnData];
```

For a live example of a dynamic hint in a programmatic-style node, view the Split Out node code.

Code standards

Following defined code standards when building your node makes your code more readable and maintainable, and helps avoid errors. This document provides guidance on good code practices for node building. It focuses on code details. For UI standards and UX guidance, refer to Node UI design.

Use the linter

The n8n node linter provides automatic checking for many of the node-building standards. You should ensure your node passes the linter's checks before publishing it. Refer to the n8n node linter documentation for more information.

Use the starter

The n8n node starter project includes a recommended setup, dependencies (including the linter), and examples to help you get started. Begin new projects with the starter.

Write in TypeScript

All n8n code is TypeScript. Writing your nodes in TypeScript can speed up development and reduce bugs.

Detailed guidelines for writing a node

These guidelines apply to any node you build.

Resources and operations

If your node can perform several operations, call the parameter that sets the operation `Operation`. If your node can do these operations on more than one resource, create a `Resource` parameter. The following code sample shows a basic resource and operations setup:

```
export const ExampleNode implements INodeType {
    description: {
        displayName: 'Example Node',
        ...
        properties: [
            {
                displayName: 'Resource',
                name: 'resource',
                type: 'options',
                options: [
                    {
                        name: 'Resource One',
                        value: 'resourceOne'
                    },
                    {
                        name: 'Resource Two',
                        value: 'resourceTwo'
                    }
                ],
                default: 'resourceOne'
            },
            {
                displayName: 'Operation',
                name: 'operation',
                type: 'options',
                // Only show these operations for Resource One
                displayOptions: {
                    show: {
                        resource: [
```

```
                                    'resourceOne'
                                ]
                            }
                        },
                        options: [
                            {
                                name: 'Create',
                                value: 'create',
                                description: 'Create an instance of Resource
One'
                            }
                        ]
                    }
                ]
            }
        }
```

Reuse internal parameter names

All resource and operation fields in an n8n node have two settings: a display name, set using the `name` parameter, and an internal name, set using the `value` parameter. Reusing the internal name for fields allows n8n to preserve user-entered data if a user switches operations.

For example: you're building a node with a resource named 'Order'. This resource has several operations, including Get, Edit, and Delete. Each of these operations uses an order ID to perform the operation on the specified order. You need to display an ID field for the user. This field has a display label, and an internal name. By using the same internal name (set in `value`) for the operation ID field on each resource, a user can enter the ID with the Get operation selected, and not lose it if they switch to Edit.

When reusing the internal name, you must ensure that only one field is visible to the user at a time. You can control this using `displayOptions`.

Detailed guidelines for writing a programmatic-style node

These guidelines apply when building nodes using the programmatic node-building style. They aren't relevant when using the declarative style. For more information on different node-building styles, refer to Choose your node building approach.

Don't change incoming data

Never change the incoming data a node receives (data accessible with `this.getInputData()`) as all nodes share it. If you need to add, change, or delete data, clone the incoming data and return the new data. If you don't do this, sibling nodes that execute after the current one will operate on the altered data and process incorrect data.

It's not necessary to always clone all the data. For example, if a node changes the binary data but not the JSON data, you can create a new item that reuses the reference to the JSON item.

Use the built in request library

Some third-party services have their own libraries on npm, which make it easier to create an integration. The problem with these packages is that you add another dependency (plus all the dependencies of the dependencies). This adds more and more code, which has to be loaded, can introduce security vulnerabilities, bugs, and so on. Instead, use the built-in module:

```
        // If no auth needed
        const response = await this.helpers.httpRequest(options);

        // If auth needed
        const response = await
this.helpers.httpRequestWithAuthentication.call(
            this,
            'credentialTypeName', // For example: pipedriveApi
            options,
        );
```

This uses the npm package Axios.

Refer to HTTP helpers for more information, and for migration instructions for the removed `this.helpers.request`.

Error handling in n8n nodes

Proper error handling is crucial for creating robust n8n nodes that provide clear feedback to users when things go wrong. n8n provides two specialized error classes to handle different types of failures in node implementations:

- **`NodeApiError`**: For API-related errors and external service failures
- **`NodeOperationError`**: For operational errors, validation failures, and configuration issues

NodeApiError

Use `NodeApiError` when dealing with external API calls and HTTP requests. This error class is specifically designed to handle API response errors and provides enhanced features for parsing and presenting API-related failures such as:

- HTTP request failures
- external API errors
- authentication/authorization failures
- rate limiting errors
- service unavailable errors

Initialize new `NodeApiError` instances using the following pattern:

```
        new NodeApiError(node: INode, errorResponse: JsonObject, options?:
NodeApiErrorOptions)
```

Common usage patterns

For basic API request failures, catch the error and wrap it in `NodeApiError`:

```
        try {
            const response = await
this.helpers.requestWithAuthentication.call(
                this,
                credentialType,
                options
            );
            return response;
        } catch (error) {
            throw new NodeApiError(this.getNode(), error as JsonObject);
        }
```

Handle specific HTTP status codes with custom messages:

```
        try {
            const response = await
this.helpers.requestWithAuthentication.call(
                this,
                credentialType,
                options
            );
            return response;
        } catch (error) {
            if (error.httpCode === "404") {
                const resource = this.getNodeParameter("resource", 0) as
string;
                const errorOptions = {
                    message: `${
                        resource.charAt(0).toUpperCase() + resource.slice(1)
                    } not found`,
                    description:
                        "The requested resource could not be found. Please
check your input parameters.",
                };
                throw new NodeApiError(
                    this.getNode(),
                    error as JsonObject,
                    errorOptions
                );
            }

            if (error.httpCode === "401") {
                throw new NodeApiError(this.getNode(), error as JsonObject,
{
                    message: "Authentication failed",
                    description: "Please check your credentials and try
again.",
                });
            }

            throw new NodeApiError(this.getNode(), error as JsonObject);
        }
```

NodeOperationError

Use `NodeOperationError` for:

- operational errors
- validation failures
- configuration issues that aren't related to external API calls
- input validation errors
- missing required parameters
- data transformation errors
- workflow logic errors

Initialize new `NodeOperationError` instances using the following pattern:

```
        new NodeOperationError(node: INode, error: Error | string |
JsonObject, options?: NodeOperationErrorOptions)
```

Common usage patterns

Use `NodeOperationError` for validating user inputs:

```
        const email = this.getNodeParameter("email", itemIndex) as string;

        if (email.indexOf("@") === -1) {
            const description = `The email address '${email}' in the 'email'
field isn't valid`;
            throw new NodeOperationError(this.getNode(), "Invalid email
address", {
                description,
                itemIndex, // for multiple items, this will link the error
to the specific item
            });
        }
```

When processing multiple items, include the item index for better error context:

```
        for (let i = 0; i < items.length; i++) {
            try {
                // Process item
                const result = await processItem(items[i]);
                returnData.push(result);
            } catch (error) {
                if (this.continueOnFail()) {
                    returnData.push({
                        json: { error: error.message },
                        pairedItem: { item: i },
                    });
                    continue;
                }

                throw new NodeOperationError(this.getNode(), error as Error,
{
                    description: error.description,
                    itemIndex: i,
                });
            }
        }
```

Node versioning

n8n supports node versioning. You can make changes to existing nodes without breaking the existing behavior by introducing a new version.

Be aware of how n8n decides which node version to load:

- If a user builds and saves a workflow using version 1, n8n continues to use version 1 in that workflow, even if you create and publish a version 2 of the node.
- When a user creates a new workflow and browses for nodes, n8n always loads the latest version of the node.

Light versioning

This is available for all node types.

One node can contain more than one version, allowing small version increments without code duplication. To use this feature:

1. Change the main `version` parameter to an array, and add your version numbers, including your existing version.
2. You can then access the version parameter with `@version` in your `displayOptions` in any object (to control which versions n8n displays the object with). You can also query the version from a function using `const nodeVersion = this.getNode().typeVersion;`.

As an example, say you want to add versioning to the NasaPics node from the Declarative node tutorial, then configure a resource so that n8n only displays it in version 2 of the node. In your base `NasaPics.node.ts` file:

```
{
    displayName: 'NASA Pics',
    name: 'NasaPics',
    icon: 'file:nasapics.svg',
    // List the available versions
    version: [1,2,3],
    // More basic parameters here
    properties: [
        // Add a resource that's only displayed for version2
        {
            displayName: 'Resource name',
            // More resource parameters
            displayOptions: {
                show: {
                    '@version': 2,
                },
            },
        },
    ],
}
```

Full versioning

This isn't available for declarative-style nodes.

As an example, refer to the Mattermost node.

Full versioning summary:

- The base node file should extend `NodeVersionedType` instead of `INodeType`.
- The base node file should contain a description including the `defaultVersion` (usually the latest), other basic node metadata such as name, and a list of versions. It shouldn't contain any node functionality.
- n8n recommends using `v1`, `v2`, and so on, for version folder names.

Node file structure

Following best practices and standards in your node structure makes your node easier to maintain. It's helpful if other people need to work with the code.

The file and directory structure of your node depends on:

- Your node's complexity.
- Whether you use node versioning.
- How many nodes you include in the npm package.

n8n recommends using the `n8n-node` tool to create the expected node file structure. You can customize the generated scaffolding as required to meet more complex needs.

Required files and directories

Your node must include:

- A `package.json` file at the root of the project. Every npm module requires this.
- A `nodes` directory, containing the code for your node:
  - This directory must contain the base file, in the format `<node-name>.node.ts`. For example, `MyNode.node.ts`.
  - n8n recommends including a codex file, containing metadata for your node. The codex filename must match the node base filename. For example, given a node base file named `MyNode.node.ts`, the codex name is `MyNode.node.json`.
  - The `nodes` directory can contain other files and subdirectories, including directories for versions, and node code split across more than one file to create a modular structure.
- A `credentials` directory, containing your credentials code. This code lives in a single credentials file. The filename format is `<node-name>.credentials.ts`. For example, `MyNode.credentials.ts`.

Modular structure

You can choose whether to place all your node's functionality in one file, or split it out into a base file and other modules, which the base file then imports. Unless your node is very simple, it's a best practice to split it out.

A basic pattern is to separate out operations. Refer to the HttpBin starter node for an example of this.

For more complex nodes, n8n recommends a directory structure. Refer to the Airtable node or Microsoft Outlook node as examples.

- `actions`: a directory containing sub-directories that represent resources.
  - Each sub-directory should contain two types of files:
    - An index file with resource description (named either `<resourceName>.resource.ts` or `index.ts`)
    - Files for operations `<operationName>.operation.ts`. These files should have two exports: `description` of the operation and an `execute` function.
- `methods`: an optional directory dynamic parameters' functions.
- `transport`: a directory containing the communication implementation.

Versioning

If your node has more than one version, and you're using full versioning, this makes the file structure more complex. You need a directory for each version, along with a base file that sets the default version. Refer to Node versioning for more information on working with versions, including types of versioning.

Decide how many nodes to include in a package

There are two possible setups when building a node:

- One node in one npm package.
- More than one node in a single npm package.

n8n supports both approaches. If you include more than one node, each node should have its own directory in the `nodes` directory.

A best-practice example for programmatic nodes

n8n's built-in Airtable node implements a modular structure and versioning, following recommended patterns.

Node base file

The node base file contains the core code of your node. All nodes must have a base file. The contents of this file are different depending on whether you're building a declarative-style or programmatic-style node. For guidance on which style to use, refer to Choose your node building approach.

These documents give short code snippets to help understand the code structure and concepts. For full walk-throughs of building a node, including real-world code examples, refer to Build a declarative-style node or Build a programmatic-style node.

You can also explore the n8n-nodes-starter and n8n's own nodes for a wider range of examples. The starter contains basic examples that you can build on. The n8n Mattermost node is a good example of a more

complex programmatic-style node, including versioning.

For all nodes, refer to the:

- Structure of the node base file
- Standard parameters

For declarative-style nodes, refer to the:

- Declarative-style parameters

For programmatic-style nodes, refer to the:

- Programmatic-style parameters
- Programmatic-style execute() method

Structure of the node base file

The node base file follows this basic structure:

1. Add import statements.
2. Create a class for the node.
3. Within the node class, create a `description` object, which defines the node.

A programmatic-style node also has an `execute()` method, which reads incoming data and parameters, then builds a request. The declarative style handles this using the `routing` key in the `properties` object, within `descriptions`.

Outline structure for a declarative-style node

This code snippet gives an outline of the node structure.

```
import { INodeType, INodeTypeDescription } from 'n8n-workflow';

export class ExampleNode implements INodeType {
    description: INodeTypeDescription = {
        // Basic node details here
        properties: [
            // Resources and operations here
        ]
    };
}
```

Refer to Standard parameters for information on parameters available to all node types. Refer to Declarative-style parameters for the parameters available for declarative-style nodes.

Outline structure for a programmatic-style node

This code snippet gives an outline of the node structure.

```
        import { IExecuteFunctions } from 'n8n-core';
        import { INodeExecutionData, INodeType, INodeTypeDescription } from
'n8n-workflow';

        export class ExampleNode implements INodeType {
            description: INodeTypeDescription = {
            // Basic node details here
            properties: [
              // Resources and operations here
            ]
          };

          async execute(this: IExecuteFunctions):
Promise<INodeExecutionData[][]> {
            // Process data and return
          }
        };
```

Refer to Standard parameters for information on parameters available to all node types. Refer to Programmatic-style parameters and Programmatic-style execute method for more information on working with programmatic-style nodes.

Standard parameters

These are the standard parameters for the node base file. They're the same for all node types.

`displayName`

String | *Required*

This is the name users see in the n8n GUI.

`name`

String | *Required*

The internal name of the object. Used to reference it from other places in the node.

`icon`

String or *Object* | *Required*

Specifies an icon for a particular node. n8n recommends uploading your own image file.

You can provide the icon file name as a string, or as an object to handle different icons for light and dark modes. If the icon works in both light and dark modes, use a string that starts with `file:`, indicating the path to the icon file. For example:

```
icon: 'file:exampleNodeIcon.svg'
```

To provide different icons for light and dark modes, use an object with `light` and `dark` properties. For example:

```
icon: {
  light: 'file:exampleNodeIcon.svg',
  dark: 'file:exampleNodeIcon.dark.svg'
}
```

–8<– “_snippets/integrations/creating-nodes/node-icons.md”

`group`

Array of strings | *Required*

Tells n8n how the node behaves when the workflow runs. Options are:

- `trigger`: node waits for a trigger.
- `schedule`: node waits for a timer to expire.
- `input`, `output`, `transform`: these currently have no effect.
- An empty array, `[]`. Use this as the default option if you don’t need `trigger` or `schedule`.

`description`

String | *Required*

A short description of the node. n8n uses this in the GUI.

`defaults`

Object | *Required*

Contains essential brand and name settings.

The object can include:

- `name`: String. Used as the node name on the canvas if the `displayName` is too long.
- `color`: String. Hex color code. Provide the brand color of the integration for use in n8n.

`forceInputNodeExecution`

Boolean | *Optional*

When building a multi-input node, you can choose to force all preceding nodes on all branches to execute before the node runs. The default is `false` (requiring only one input branch to run).

`inputs`

Array of strings | *Required*

Names the input connectors. Controls the number of connectors the node has on the input side. If you need only one connector, use `input: ['main']`.

outputs

Array of strings | *Required*

Names the output connectors. Controls the number of connectors the node has on the output side. If you need only one connector, use `output: ['main']`.

requiredInputs

Integer or *Array* | *Optional*

Used for multi-input nodes. Specify inputs by number that must have data (their branches must run) before the node can execute.

credentials

Array of objects | *Required*

This parameter tells n8n the credential options. Each object defines an authentication type.

The object must include:

- `name`: the credential name. Must match the `name` property in the credential file. For example, `name: 'asanaApi'` in `Asana.node.ts` links to `name = 'asanaApi'` in `AsanaApi.credential.ts`.
- `required`: Boolean. Specify whether authentication is required to use this node.

requestDefaults

Object | *Required*

Set up the basic information for the API calls the node makes.

This object must include:

- `baseURL`: The API base URL.

You can also add:

- `headers`: an object describing the API call headers, such as content type.
- `url`: string. Appended to the `baseURL`. You can usually leave this out. It's more common to provide this in the `operations`.

properties

Array of objects | *Required*

This contains the resource and operations objects that define node behaviors, as well as objects to set up mandatory and optional fields that can receive user input.

Resource objects

A resource object includes the following parameters:

- `displayName`: String. This should always be `Resource`.
- `name`: String. This should always be `resource`.
- `type`: String. Tells n8n which UI element to use, and what input type to expect. For example, `options` results in n8n adding a dropdown that allows users to choose one option. Refer to Node UI elements for more information.
- `noDataExpression`: Boolean. Prevents using an expression for the parameter. Must always be `true` for `resource`.

Operations objects

The operations object defines the available operations on a resource.

- `displayName`: String. This should always be `Options`.
- `name`: String. This should always be `option`.
- `type`: String. Tells n8n which UI element to use, and what input type to expect. For example, `dateTime` results in n8n adding a date picker. Refer to Node UI elements for more information.
- `noDataExpression`: Boolean. Prevents using an expression for the parameter. Must always be `true` for `operation`.
- `options`: Array of objects. Each objects describes an operation's behavior, such as its routing, the REST verb it uses, and so on. An `options` object includes:
  - `name`. String.
  - `value`. String.
  - `action`: String. This parameter combines the resource and operation. You should always include it, as n8n will use it in future versions. For example, given a resource called `"Card"` and an operation `"Get all"`, your action is `"Get all cards"`.
  - `description`: String.
  - `routing`: Object containing request details.

Additional fields objects

These objects define optional parameters. n8n displays them under **Additional Fields** in the GUI. Users can choose which parameters to set.

The objects must include:

```
displayName: 'Additional Fields',
name: 'additionalFields',
// The UI element type
type: ''
placeholder: 'Add Field',
default: {},
displayOptions: {
  // Set which resources and operations this field is available for
  show: {
    resource: [
      // Resource names
```

```
            ],
            operation: [
              // Operation names
            ]
          },
        }
```

For more information about UI element types, refer to UI elements.

Declarative-style parameters

These are the parameters available for node base file of declarative-style nodes.

This document gives short code snippets to help understand the code structure and concepts. For a full walk-through of building a node, including real-world code examples, refer to Build a declarative-style node.

Refer to Standard parameters for parameters available to all nodes.

`methods` and `loadOptions`

Object | *Optional*

`methods` contains the `loadOptions` object. You can use `loadOptions` to query the service to get user-specific settings, then return them and render them in the GUI so the user can include them in subsequent queries. The object must include routing information for how to query the service, and output settings that define how to handle the returned options. For example:

```
        methods : {
            loadOptions: {
                routing: {
                    request: {
                        url: '/webhook/example-option-parameters',
                        method: 'GET',
                    },
                    output: {
                        postReceive: [
                            {
                                // When the returned data is nested under another property
                                // Specify that property key
                                type: 'rootProperty',
                                properties: {
                                    property: 'responseData',
                                },
                            },
                            {
                                type: 'setKeyValue',
                                properties: {
                                    name: '={{$responseItem.key}} ({{$responseItem.value}})',
                                    value: '={{$responseItem.value}}',
                                },
```

```
                        },
                        {
                            // If incoming data is an array of objects,
sort alphabetically by key
                            type: 'sort',
                            properties: {
                                key: 'name',
                            },
                        },
                    ],
                },
            },
        }
    },
```

routing

Object | *Required*

`routing` is an object used within an `options` array in operations and input field objects. It contains the details of an API call.

The code example below comes from the Declarative-style tutorial. It sets up an integration with a NASA API. It shows how to use `requestDefaults` to set up the basic API call details, and `routing` to add information for each operation.

```
description: INodeTypeDescription = {
  // Other node info here
  requestDefaults: {
            baseURL: 'https://api.nasa.gov',
            url: '',
            headers: {
                Accept: 'application/json',
                'Content-Type': 'application/json',
            },
        },
    properties: [
      // Resources here
      {
        displayName: 'Operation'
        // Other operation details
        options: [
          {
            name: 'Get'
            value: 'get',
            description: '',
            routing: {
              request: {
                method: 'GET',
                url: '/planetary/apod'
              }
            }
          }
        ]
      }
    ]
}
```

`version`

Number or *Array* | *Optional*

If you have one version of your node, this can be a number. If you want to support more than one version, turn this into an array, containing numbers for each node version.

n8n supports two methods of node versioning, but declarative-style nodes must use the light versioning approach. Refer to Node versioning for more information.

Programmatic-style parameters

These are the parameters available for node base file of programmatic-style nodes.

This document gives short code snippets to help understand the code structure and concepts. For a full walk-through of building a node, including real-world code examples, refer to Build a programmatic-style node.

Programmatic-style nodes also use the `execute()` method. Refer to Programmatic-style execute method for more information.

Refer to Standard parameters for parameters available to all nodes.

`defaultVersion`

Number | *Optional*

Use `defaultVersion` when using the full versioning approach.

n8n support two methods of node versioning. Refer to Node versioning for more information.

`methods` and `loadOptions`

Object | *Optional*

Contains the `loadOptions` method for programmatic-style nodes. You can use this method to query the service to get user-specific settings (such as getting a user's email labels from Gmail), then return them and render them in the GUI so the user can include them in subsequent queries.

For example, n8n's Gmail node uses `loadOptions` to get all email labels:

```
methods = {
    loadOptions: {
        // Get all the labels and display them
        async getLabels(
            this: ILoadOptionsFunctions,
        ): Promise<INodePropertyOptions[]> {
            const returnData: INodePropertyOptions[] = [];
            const labels = await googleApiRequestAllItems.call(
```

```
                    this,
                    'labels',
                    'GET',
                    '/gmail/v1/users/me/labels',
                );
                for (const label of labels) {
                    const labelName = label.name;
                    const labelId = label.id;
                    returnData.push({
                        name: labelName,
                        value: labelId,
                    });
                }
                return returnData;
            },
        },
    };
```

`version`

Number or *Array* | *Optional*

Use `version` when using the light versioning approach.

If you have one version of your node, this can be a number. If you want to support multiple versions, turn this into an array, containing numbers for each node version.

n8n support two methods of node versioning. Programmatic-style nodes can use either. Refer to Node versioning for more information.

Programmatic-style execute() method

The main difference between the declarative and programmatic styles is how they handle incoming data and build API requests. The programmatic style requires an `execute()` method, which reads incoming data and parameters, then builds a request. The declarative style handles requests using the `routing` key in the `operations` object.

The `execute()` method creates and returns an instance of `INodeExecutionData`.

Node codex files

The codex file contains metadata about your node. This file is the JSON file at the root of your node. For example, the `HttpBin.node.json` file in the n8n starter.

The codex filename must match the node base filename. For example, given a node base file named `MyNode.node.ts`, the codex would be named `MyNode.node.json`.

Parameter	Description
`node`	Includes the node name. Must start with `n8n-nodes-base.`. For example, `n8n-nodes-base.openweatherapi`.
`nodeVersion`	The node version. This should have the same value as the `version` parameter in your main node file. For example, `"1.0"`.
`codexVersion`	The codex file version. The current version is `"1.0"`.
`categories`	The settings in the `categories` array determine which category n8n adds your node to in the GUI. See Node categories for more information.
`resources`	The `resources` object contains links to your node documentation. n8n automatically adds help links to credentials and nodes in the GUI.

Node categories

You can define one or more categories in your node configuration JSON. This helps n8n put the node in the correct category in the nodes panel.

Choose from these categories:

- Data & Storage
- Finance & Accounting
- Marketing & Content
- Productivity
- Miscellaneous
- Sales
- Development
- Analytics
- Communication
- Utility

You must match the syntax. For example, `Data & Storage` not `data and storage`.

Credentials file

The credentials file defines the authorization methods for the node. The settings in this file affect what n8n displays in the **Credentials** modal, and must reflect the authentication requirements of the service you're connecting to.

In the credentials file, you can use all the n8n UI elements. n8n encrypts the data that's stored using credentials using an encryption key.

Structure of the credentials file

The credentials file follows this basic structure:

1. Import statements
2. Create a class for the credentials
3. Within the class, define the properties that control authentication for the node.

Outline structure

```
import {
    IAuthenticateGeneric,
    ICredentialTestRequest,
    ICredentialType,
    INodeProperties,
} from 'n8n-workflow';

export class ExampleNode implements ICredentialType {
    name = 'exampleNodeApi';
    displayName = 'Example Node API';
    documentationUrl = '';
    properties: INodeProperties[] = [
        {
            displayName: 'API Key',
            name: 'apiKey',
            type: 'string',
            default: '',
        },
    ];
    authenticate: IAuthenticateGeneric = {
        type: 'generic',
        properties: {
            // Can be body, header, qs or auth
            qs: {
                // Use the value from `apiKey` above
                'api_key': '={{$credentials.apiKey}}'
            }

        },
    };
    test: ICredentialTestRequest = {
        request: {
            baseURL: '={{$credentials?.domain}}',
            url: '/bearer',
        },
    };
}
```

Parameters

`name`

String. The internal name of the object. Used to reference it from other places in the node.

`displayName`

String. The name n8n uses in the GUI.

`documentationUrl`

String. URL to your credentials documentation.

`properties`

Each object contains:

- `displayName`: the name n8n uses in the GUI.
- `name`: the internal name of the object. Used to reference it from other places in the node.
- `type`: the data type expected, such as `string`.
- `default`: the URL that n8n should use to test credentials.

`authenticate`

- `authenticate`: Object. Contains objects that tell n8n how to inject the authentication data as part of the API request.

`type`

String. If you're using an authentication method that sends data in the header, body, or query string, set this to `'generic'`.

`properties`

Object. Defines the authentication methods. Options are:

- `body`: Object. Sends authentication data in the request body. Can contain nested objects.

```
authenticate: IAuthenticateGeneric = {
    type: 'generic',
    properties: {
        body: {
            username: '={{$credentials.username}}',
            password: '={{$credentials.password}}',
        },
    },
};
```

- `header`: Object. Send authentication data in the request header.

```
authenticate: IAuthenticateGeneric = {
    type: 'generic',
    properties: {
        header: {
            Authorization: '=Bearer {{$credentials.authToken}}',
        },
    },
};
```

- `qs`: Object. Stands for "query string." Send authentication data in the request query string.

```
authenticate: IAuthenticateGeneric = {
    type: 'generic',
    properties: {
        qs: {
```

```
                token: '={{$credentials.token}}',
            },
        },
    };
```

- `auth`: Object. Used for Basic Auth. Requires `username` and `password` as the key names.

```
    authenticate: IAuthenticateGeneric = {
        type: 'generic',
        properties: {
            auth: {
                username: '={{$credentials.username}}',
                password: '={{$credentials.password}}',
            },
        },
    };
```

`test`

Provide a `request` object containing a URL and authentication type that n8n can use to test the credential.

```
    test: ICredentialTestRequest = {
            request: {
                baseURL: '={{$credentials?.domain}}',
                url: '/bearer',
            },
        };
```

HTTP request helper for node builders

n8n provides a flexible helper for making HTTP requests, which abstracts away most of the complexity.

Usage

Call the helper inside the `execute` function.

```
        // If no auth needed
        const response = await this.helpers.httpRequest(options);

        // If auth needed
        const response = await
this.helpers.httpRequestWithAuthentication.call(
            this,
            'credentialTypeName', // For example: pipedriveApi
            options,
        );
```

`options` is an object:

```
        {
            url: string;
            headers?: object;
```

```
            method?: 'GET' | 'POST' | 'PUT' | 'DELETE' | 'HEAD';
            body?: FormData | Array | string | number | object | Buffer |
URLSearchParams;
            qs?: object;
            arrayFormat?: 'indices' | 'brackets' | 'repeat' | 'comma';
            auth?: {
                username: string,
                password: string,
            };
            disableFollowRedirect?: boolean;
            encoding?: 'arraybuffer' | 'blob' | 'document' | 'json' | 'text'
| 'stream';
            skipSslCertificateValidation?: boolean;
            returnFullResponse?: boolean;
            proxy?: {
                host: string;
                port: string | number;
                auth?: {
                    username: string;
                    password: string;
                },
                protocol?: string;
            };
            timeout?: number;
            json?: boolean;
        }
```

`url` is required. The other fields are optional. The default method is `GET`.

Some notes about the possible fields:

- `body`: you can use a regular JavaScript object for JSON payload, a buffer for file uploads, an instance of FormData for `multipart/form-data`, and `URLSearchParams` for `application/x-www-form-urlencoded`.
- `headers`: a key-value pair.
  - If `body` is an instance of `FormData` then n8n adds `content-type: multipart/form-data` automatically.
  - If `body` is an instance of `URLSearchParams`, then n8n adds `content-type: application/x-www-form-urlencoded`.
  - To override this behavior, set a `content-type` header.
- `arrayFormat`: if your query string contains an array of data, such as `const qs = {IDs: [15,17]}`, the value of `arrayFormat` defines how n8n formats it.
  - `indices` (default): `{ a: ['b', 'c'] }` as `a[0]=b&a[1]=c`
  - `brackets`: `{ a: ['b', 'c'] }` as `a[]=b&a[]=c`
  - `repeat`: `{ a: ['b', 'c'] }` as `a=b&a=c`
  - `comma`: `{ a: ['b', 'c'] }` as `a=b,c`
- `auth`: Used for Basic auth. Provide `username` and `password`. n8n recommends omitting this, and using `helpers.httpRequestWithAuthentication(...)` instead.
- `disableFollowRedirect`: By default, n8n follows redirects. You can set this to true to prevent this from happening.
- `skipSslCertificateValidation`: Used for calling HTTPS services without proper certificate
- `returnFullResponse`: Instead of returning just the body, returns an object with more data in the following format: `{body: body, headers: object, statusCode: 200, statusMessage: 'OK'}`
- `encoding`: n8n can detect the content type, but you can specify `arrayBuffer` to receive a Buffer you can read from and interact with.

Example

For an example, refer to the Mattermost node.

Deprecation of the previous helper

The previous helper implementation using `this.helpers.request(options)` used and exposed the `request-promise` library. This was removed in version 1.

To minimize incompatibility, n8n made a transparent conversion to another library called `Axios`.

If you are having issues, please report them in the Community Forums or on GitHub.

Migration guide to the new helper

The new helper is much more robust, library agnostic, and easier to use.

New nodes should all use the new helper. You should strongly consider migrating existing custom nodes to the new helper. These are the main considerations when migrating:

- Accepts `url`. Doesn't accept `uri`.
- `encoding: null` now must be `encoding: arrayBuffer`.
- `rejectUnauthorized: false` is now `skipSslCertificateValidation: true`
- Use `body` according to `content-type` headers to clarify the payload.
- `resolveWithFullResponse` is now `returnFullResponse` and has similar behavior

Item linking

–8<– "_snippets/data/data-mapping/item-linking-node-creators.md"

UX guidelines for community nodes

Your node's UI must conform to these guidelines to be a verified community node candidate.

Credentials

API key and sensitive credentials should always be password fields.

OAuth

Always include the OAuth credential if available.

Node structure

Operations to include

Try to include **CRUD** operations for each resource type.

Try to include common operations in nodes for each resource. n8n uses some CRUD operations to keep the experience consistent and allow users to perform basic operations on the resource. The suggested operations are:

- **Create**
- **Create or Update (Upsert)**
- **Delete**
- **Get**
- **Get Many:** also used when some filtering or search is available
- **Update**

Notes:

1. These operations can apply to the resource itself or an entity inside of the resource (for example, a row inside a Google Sheet). When operating on an entity inside of the resource, you must specify the **name of the entity** in the operations name.
2. The naming could change depending on the node and the resource. Check the following guidelines for details.

Resource Locator

- Use a Resource Locator component whenever possible. This provides a much better UX for users. The Resource Locator Component is most often useful when you have to select a single item.
- The default option for the Resource Locator Component should be `From list` (if available).

Consistency with other nodes

- Maintain UX consistency: n8n tries to keep its UX consistent. This means following existing UX patterns, in particular, those used in the latest new or overhauled nodes.
- Check similar nodes: For example, if you're working on a database node, it's worth checking the Postgres node.

Sorting options

- You can enhance certain "Get Many" operations by providing users with sorting options.
- Add sorting in a dedicated collection (below the "Options" collection). Follow the example of Airtable Record:Search.

Node functionality

Deleting operations output

When deleting an item (like a record or a row), return an array with a single object: `{"deleted": true}`. This is a confirmation for the user that the deletion was successful and the item will trigger the following node.

Simplifying output fields

Normal nodes: ‘Simplify’ parameter

When an endpoint returns data with more than 10 fields, add the “Simplify” boolean parameter to return a simplified version of the output with max 10 fields.

- One of the main issues with n8n can be the size of data and the Simplify parameter limits that problem by reducing data size.
- Select the most useful fields to output in the simplified node and sort them to have the most used ones at the top.
- In the Simplify mode, it’s often best to flatten nested fields
- Display Name: `Simplify`
- Description: `Whether to return a simplified version of the response instead of the raw data`

AI tool nodes: ‘Output’ parameter

When an endpoint returns data with more than 10 fields, add the ‘Output’ option parameter with 3 modes.

In AI tool nodes, allow the user to be more granular and select the fields to output. The rationale is that tools may run out of context window and they can get confused by too many fields, so it’s better to pass only the ones they need.

Options:

- **Simplified:** Works the same as the “Simplify” parameter described above.
- **Raw:** Returns all the available fields.
- **Selected fields:** Shows a multi-option parameter for selecting the fields to add to the output and send to the AI agent. By default, this option always returns the ID of the record/entity.

Copy

Text Case

Use **Title Case** for the node `name`, `parameters display names` (labels), `dropdown titles`. Title Case is when you capitalize the first letter of each word, except for certain small words, such as articles and short prepositions.

Use **Sentence case** for node `action` names, node `descriptions`, `parameters descriptions` (tooltips), `hints`, `dropdown descriptions`.

Terminology

- **Use the third-party service terminology:** Try to use the same terminology as the service you're interfacing with (for example, Notion 'blocks', not Notion 'paragraphs').
- **Use the terminology used in the UI:** Stick to the terminology used in the user interface of the service, rather than that used in the APIs or technical documentation (for example, in Trello you "archive" cards, but in the API they show up as "closed". In this case, you might want to use "archive").
- **No tech jargon:** Don't use technical jargon where simple words will do. For example, use "field" instead of "key".
- **Consistent naming:** Choose one term for something and stick to it. For example, don't mix "directory" and "folder".

Placeholders

It's often helpful to insert examples of content in parameters placeholders. These should start with "e.g." and use **camel case** for the demo content in fields.

Placeholder examples to copy:

- image: `e.g. https://example.com/image.png`
- video: `e.g. https://example.com/video.mp4`
- search term: `e.g. automation`
- email: `e.g. nathan@example.com`
- Twitter user (or similar): `e.g. n8n`
- Name and last name: `e.g. Nathan Smith`
- First name: `e.g. Nathan`
- Last name: `e.g. Smith`

Operations name, action, and description

- **Name:** This is the name displayed in the select when the node is open on the canvas. It must use title case and doesn't have to include the resource (for example, "Delete").
- **Action:** This is the name of the operation displayed in the panel where the user selects the node. It must be in sentence case and must include the resource (for example, "Delete record").
- **Description:** This is the sub-text displayed below the name in the select when the node is open on the canvas. It must use sentence case and must include the resource. It can add a bit of information and use alternative words than the basic resource/operation (for example, "Retrieve a list of users").
- If the operation acts on an entity that's not the Resource (for example, a row in a Google Sheet), specify that in the operation name (for example, "Delete Row").

As a general rule, is important to understand what the **object** of an operation is. Sometimes, the object of an Operation is the resource itself (for example, `Sheet:Delete` to delete a Sheet).

In other cases, the object of the operation isn't the resource, but something contained inside the resource (for example, `Table:Delete rows`, here the resource is the table, but what you are operating on are the rows inside of it).

Naming `name`

This is the name displayed in the select when the node is open on the canvas.

- Parameter: `name`
- Case: Title Case

Naming guidelines:

- **Don’t repeat the resource (if the resource selection is above):** The resource is often displayed above the operation, so it’s not necessary to repeat it in the operation (this is the case if the object of the operation is the resource itself).
  - For example: `Sheet:Delete` → No need to repeat `Sheet` in `Delete`, because n8n displays `Sheet` in the field above and what you’re deleting is the Sheet.
- **Specify the resource if there’s no resource selection above:** In some nodes, you won’t have a resource selection (because there’s only one resource). In these cases, specify the resource in the operation.
  - For example: `Delete Records` → In Airtable, there’s no resource selection, so it’s better to specify that the Delete operation will delete records.
- **Specify the object of the operation if it’s not the resource:** Sometimes, the object of the operation isn’t the resource. In these cases, specify the object in the operation as well.
  - For example: `Table:Get Columns` → Specify `Columns` because the resource is `Table`, while the object of the operation is `Columns`.

Naming `action`

This is the name of the operation displayed in the panel where the user selects the node. * Parameter: `action` * Case: Sentence case

Naming guidelines:

- **Omit articles:** To keep the text shorter, get rid of articles (a, an, the…).
  - **correct**: `Update row in sheet`
  - **incorrect**: `Update a row in a sheet`
- **Repeat the resource:** In this case, it’s okay to repeat the resource. Even if the resource is visible in the list, the user might not notice and it’s useful to repeat it in the operation label.
- **Specify the object of the operation if it is not the resource:** Same as for the operation name. In this case, you don’t need to repeat the resource.
  - For example: `Append Rows` → You have to specify `Rows` because rows are what you’re actually appending to. Don’t add the resource (`Sheet`) since you aren’t appending to the resource.

Naming `description`

This is the subtext displayed below the name in the selection when the node is open on the canvas.

- Parameter: `description`
- Case: Sentence case

Naming guidelines:

- If possible, add more information than that specified in the

operation `name`
- Use alternative wording to help users better understand what the operation is doing. Some people might not understand the text used in the operation (maybe English isn't their native language), and using alternative working could help them.

Vocabulary

n8n uses a general vocabulary and some context-specific vocabulary for groups of similar applications (for example, databases or spreadsheets).

The general vocabulary takes inspiration from CRUD operations:

- **Clear**
  - Delete all the contents of the resource (empty the resource).
  - Description: `Delete all the <CHILD_ELEMENT>s inside the <RESOURCE>`
- **Create**
  - Create a new instance of the resource.
  - Description: `Create a new <RESOURCE>`
- **Create or Update**
  - Create or update an existing instance of the resource.
  - Description: `Create a new <RESOURCE> or update an existing one (upsert)`
- **Delete**
  - You can use "Delete" in two different ways:
    1. Delete a resource:
       - Description: `Delete a <RESOURCE> permanently` (use "permanently" only if that's the case)
    2. Delete something **inside** of the resource (for example, a row):
       - In this case, **always specify the object of the operation**: for example, `Delete Rows` or `Delete Records`.
       - Description: `Delete a <CHILD_ELEMENT> permanently`
- **Get**
  - You can use "Get" in two different ways:
    1. Get a resource:
       - Description: `Retrieve a <RESOURCE>`
    2. Get an item **inside** of the resource (for example, records):
       - In this case, **always specify the object of the operation**: for example, `Get Row` or `Get Record`.
       - Description: `Retrieve a <CHILD_ELEMENT> from the/a <RESOURCE>`
- **Get Many**
  - You can use "Get Many" in two different ways:
    1. Get a list of resources (without filtering):
       - Description: `Retrieve a list of <RESOURCE>s`
    2. Get a list of items **inside** of the resource (for example, records):
       - In this case, **always specify the object of the operation**: for example, `Get Many Rows` or `Get Many Records`.
       - You can omit `Many`: `Get Many Rows` can be `Get Rows`.
       - Description: `List all <CHILD_ELEMENT>s in the/a <RESOURCE>`
- **Insert** or **Append**
  - Add something inside of a resource.
  - Use `insert` for database nodes.

    - Description: `Insert <CHILD_ELEMENT>(s) in a <RESOURCE>`
- **Insert or Update** or **Append or Update**
    - Add or update something inside of a resource.
    - Use `insert` for database nodes.
    - Description: `Insert <CHILD_ELEMENT>(s) or update an existing one(s) (upsert)`
- **Update**
    - You can use “Update” in two different ways:
        1. Update a resource:
            - Description: `Update one or more <RESOURCE>s`
        2. Update something **inside** of a resource (for example, a row):
            - In this case, **always specify the object of the operation**: for example, `Update Rows` or `Update Records`.
            - Description: `Update <CHILD_ELEMENT>(s) inside a <RESOURCE>`

Referring to parameter and field name

When you need to refer to parameter names or field names in copy, wrap them in single quotation marks (for example, “Please fill the `'name'` parameter).

Boolean description

Start the description of boolean components with ‘Whether…’

Errors

General philosophy

Errors are sources of pain for users. For this reason, n8n always wants to tell the user:

- **What happened**: a description of the error and what went wrong.
- **How to solve the problem**: or at least how to get unstuck and continue using n8n without problems. n8n doesn’t want users to remain blocked, so use this as an opportunity to guide them to success.

Error structure in the Output panel

Error Message - What happened

This message explains to the user what happened, and the current issue that prevents the execution completing.

- If you have the `displayName` of the parameter that triggered the error, include it in the error message or description (or both).
- Item index: if you have the ID of the item that triggered the error, append `[Item X]` to the error message. For example, `The ID of the release in the parameter “Release ID” for could not be found [item 2]`.
- Avoid using words like “error”, “problem”, “failure”, “mistake”.

Error Description - How to solve or get unstuck

The description explains to users how to solve the problem, what to change in the node configuration (if that's the case), or how to get unstuck. Here, you should guide them to the next step and unblock them.

Avoid using words like "error", "problem", "failure", "mistake".

Community node verification guidelines

Use the n8n-node tool

All verified community node authors should strongly consider using the `n8n-node` tool to create and check their package. This helps n8n ensure quality and consistency by:

- Generating the expected package file structure
- Adding the required metadata and configuration to the `package.json` file
- Making it easy to lint your code against n8n's standards
- Allowing you to load your node in a local n8n instance for testing

Package source verification

- Verify that your npm package repository URL matches the expected GitHub (or other platform) repository.
- Confirm that the package author / maintainer matches between npm and the repository.
- Confirm that the git link in npm works and that the repository is public.
- Make sure your package has proper documentation (README, usage examples, etc.).
- Make sure your package license is MIT.

No external dependencies

- Ensure that your package does **not** include any external dependencies to keep it lightweight and easy to maintain.

Proper documentation

- Provide clear documentation, whether it's a **README** on GitHub or links to relevant **API documentation**.
- Include usage instructions, example workflows, and any necessary authentication details.

No access to environment variables or file system

- The code **must not** interact with environment variables or attempt

to read/write files.
- Pass all necessary data through node parameters.

Follow n8n best practices

- Maintain a clear and consistent coding style.
- Use **TypeScript** and follow n8n's **node development guidelines**.
- Ensure proper error handling and validation.
- Make sure the linter passes (in other words, make sure running `npx @n8n/scan-community-package n8n-nodes-PACKAGE` passes).

Use English language only

- Both the node interface and all documentation must be in **English** only.
- This includes parameter names, descriptions, help text, error messages and **README** content.

Test a node

This section contains information about testing your node.

There are two ways to test your node:

- Manually, by running it on your own machine within a local n8n instance.
- Automatically, using the linter.

You should use both methods before publishing your node.

Run your node locally

–8<– "_snippets/integrations/creating-nodes/testing.md"

---

n8n node linter

n8n's node linter, `eslint-plugin-n8n-nodes-base`, statically analyzes ("lints") the source code of n8n nodes and credentials in the official repository and in community packages. The linter detects issues and automatically fixes them to help you follow best practices.

`eslint-plugin-n8n-nodes-base` contains a collection of rules for node files (`*.node.ts`), resource description files (`*Description.ts`), credential files (`*.credentials.ts`), and the `package.json` of a community package.

Setup

If using the n8n node starter: Run `npm install` in the starter project to install all dependencies. Once the installation finishes, the linter is available to you.

If using VS Code, install the ESLint VS Code extension. For other IDEs, refer to their ESLint integrations.

Usage

You can use the linter in a community package or in the main n8n repository.

Linting

In a community package, the linter runs automatically after installing dependencies and before publishing the package to npm. In the main n8n repository, the linter runs automatically using GitHub Actions whenever you push to your pull request.

In both cases, VS Code lints in the background as you work on your project. Hover over a detected issue to see a full description of the linting and a link to further information.

You can also run the linter manually:

- Run `npm run lint` to lint and view detected issues in your console.
- Run `npm run lintfix` to lint and automatically fix issues. The linter fixes violations of rules marked as automatically fixable.

Both commands can run in the root directory of your community package, or in `/packages/nodes-base/` in the main repository.

Exceptions

Instead of fixing a rule violation, you can also make an exception for it, so the linter doesn't flag it.

To make a lint exception from VS Code: hover over the issue and click on `Quick fix` (or `cmd+.` in macOS) and select **Disable {rule} for this line**. Only disable rules for a line where you have good reason to. If you think the linter is incorrectly reporting an issue, please report it in the linter repository.

To add a lint exception to a single file, add a code comment. In particular, TSLint rules may not show up in VS Code and may need to be turned off using code comments. Refer to the TSLint documentation for more guidance.

Troubleshooting

Credentials

Error message: 'Credentials of type "*" aren't known'

Check that the name in the credentials array matches the name used in the property name of the credentials' class.

[Image: Troubleshooting credentials]

Editor UI

Error message: 'There was a problem loading init data: API-Server can not be reached. It's probably down'

- Check that the names of the node file, node folder, and class match the path added to `packages/nodes-base/package.json`.
- Check that the names used in the `displayOptions` property are names used by UI elements in the node.

Node icon doesn't show up in the Add Node menu and the Editor UI

- Check that the icon is in the same folder as the node.
- Check that it's either in PNG or SVG format.
- When the `icon` property references the icon file, check that it includes the logo extension (`.png` or `.svg`) and that it prefixes it with `file:`. For example, `file:friendGrid.png` or `file:friendGrid.svg`.

Node icon doesn't fit

- If you use an SVG file, make sure the canvas size is square. You can find instructions to change the canvas size of an SVG file using GIMP here.
- If you use a PNG file, make sure that it's 60x60 pixels.

Node doesn't show up in the Add Node menu

Check that you registered the node in the `package.json` file in your project.

Changes to the description properties don't show in the UI on refreshing

Every time you change the description properties, you have to stop the current n8n process (`ctrl + c`) and run it again. You may also need to re-run `npm link`.

Linter incorrectly warning about file name case

The node linter has rules for file names, including what case they should be. Windows users may encounter an issue when renaming files that causes the linter to continue giving warnings, even after you rename the files. This is due to a known Windows issue with changing case when renaming files.

Deploy a node

This section contains details on how to deploy and share your node.

You can choose to:

- Submit your node to the community node repository. This makes it available for everyone to use, and allows you to install and use it like any other community node. This is the only way to use custom nodes on cloud.
- Install the node into your n8n instance as a private node.

Submit community nodes

–8<– “_snippets/integrations/submit-community-node.md”

Install private nodes

You can build your own nodes and install them in your n8n instance without publishing them on npm. This is useful for nodes that you create for internal use only at your company.

Install your node in a Docker n8n instance

If you’re running n8n using Docker, you need to create a Docker image with the node installed in n8n.

1. Create a Dockerfile and paste the code from this Dockerfile.

   Your Dockerfile should look like this:

```
        FROM node:16-alpine

        ARG N8N_VERSION

        RUN if [ -z "$N8N_VERSION" ] ; then echo "The N8N_VERSION
argument is missing!" ; exit 1; fi

        # Update everything and install needed dependencies
        RUN apk add --update graphicsmagick tzdata git tini su-exec

        # Set a custom user to not have n8n run as root
        USER root

        # Install n8n and the packages it needs to build it correctly.
        RUN apk --update add --virtual build-dependencies python3 build-
base ca-certificates && \
            npm config set python "$(which python3)" && \
            npm_config_user=root npm install -g full-icu
n8n@${N8N_VERSION} && \
            apk del build-dependencies \
            && rm -rf /root /tmp/* /var/cache/apk/* && mkdir /root;
```

```
        # Install fonts
        RUN apk --no-cache add --virtual fonts msttcorefonts-installer
fontconfig && \
            update-ms-fonts && \
            fc-cache -f && \
            apk del fonts && \
            find  /usr/share/fonts/truetype/msttcorefonts/ -type l -exec
unlink {} \; \
            && rm -rf /root /tmp/* /var/cache/apk/* && mkdir /root

        ENV NODE_ICU_DATA /usr/local/lib/node_modules/full-icu

        WORKDIR /data

        COPY docker-entrypoint.sh /docker-entrypoint.sh
        ENTRYPOINT ["tini", "--", "/docker-entrypoint.sh"]

        EXPOSE 5678/tcp
```

2. Compile your custom node code (`npm run build` if you are using nodes starter). Copy the **node** and **credential** folders from within the **dist** folder into your container's `~/.n8n/custom/` directory. This makes them available to Docker.

3. Download the docker-entrypoint.sh file, and place it in the same directory as your Dockerfile.

4. Build your Docker image:

```
        # Replace <n8n-version-number> with the n8n release version
number.
        # For example, N8N_VERSION=0.177.0
        docker build --build-arg N8N_VERSION=<n8n-version-number> --
tag=customizedn8n .
```

You can now use your node in Docker.

Install your node in a global n8n instance

If you've installed n8n globally, make sure that you install your node inside n8n. n8n will find the module and load it automatically.

Self-hosting n8n

This section provides guidance on setting up n8n for both the Enterprise and Community self-hosted editions. The Community edition is free, the Enterprise edition isn't.

See Community edition features for a list of available features.

- **Installation and server setups**

  Install n8n on any platform using npm or Docker. Or follow our guides to popular hosting platforms.

  :octicons-arrow-right-24: Docker installation guide

- **Configuration**

  Learn how to configure n8n with environment variables.

  :octicons-arrow-right-24: Environment Variables

- **Users and authentication**

  Choose and set up user authentication for your n8n instance.

  :octicons-arrow-right-24: Authentication

- **Scaling**

  Manage data, modes, and processes to keep n8n running smoothly at scale.

  :octicons-arrow-right-24: Scaling

- **Securing n8n**

  Secure your n8n instance by setting up SSL, SSO, or 2FA or blocking or opting out of some data collection or features.

  :octicons-arrow-right-24: Securing n8n guide

- **Starter kits**

  New to n8n or AI? Try our Self-hosted AI Starter Kit. Curated by n8n, it combines the self-hosted n8n platform with compatible AI products and components to get you started building self-hosted AI workflows.

  :octicons-arrow-right-24: Starter kits

–8<– “_snippets/self-hosting/warning.md”

Community Edition Features

The community edition includes almost the complete feature set of n8n, except for the features listed here.

The community edition doesn’t include these features:

- Custom Variables
- Environments
- External secrets
- External storage for binary data
- Log streaming (Logging *is* included)
- Multi-main mode (Queue mode *is* included)
- Projects
- SSO (SAML, LDAP)
- Sharing (workflows, credentials) (Only the instance owner and the user who creates them can access workflows and credentials)
- Version control using Git

These features are available on the Enterprise Cloud plan, including the self-hosted Enterprise edition. Some of these features are available on the Starter and Pro Cloud plan.

See pricing for reference.

Registered Community Edition

You can unlock extra features by registering your n8n community edition. You register with your email and receive a license key.

Registering unlocks these features for the community edition:

- Folders: Organize your workflows into tidy folders
- Debug in editor: Copy and pin execution data when working on a workflow
- Custom execution data: Save, find, and annotate execution metadata

To register a new community edition instance, select the option during your initial account creation.

To register an existing community edition instance:

1. Select the **three dots icon** Image: three dots icon in the lower-left corner.
2. Select **Settings** and then **Usage and plan**.
3. Select **Unlock** to enter your email and then select **Send me a free license key**.
4. Check your email for the account you entered.

Once you have a license key, activate it by clicking the button in the license email or by visiting **Options > Settings > Usage and plan** and selecting **Enter activation key**.

Once activated, your license will not expire. We may change the unlocked features in the future. This will not impact previously unlocked features.

---

npm

npm is a quick way to get started with n8n on your local machine. You must have Node.js installed. n8n requires a Node.js version between 20.19 and 24.x, inclusive.

–8<– “_snippets/self-hosting/installation/latest-next-version.md”

Try n8n with npx

You can try n8n without installing it using npx.

From the terminal, run:

```
npx n8n
```

This command will download everything that’s needed to start n8n. You can then access n8n and start building workflows by opening http://localhost:5678.

Install globally with npm

To install n8n globally, use npm:

```
npm install n8n -g
```

To install or update to a specific version of n8n use the `@` syntax to specify the version. For example:

```
npm install -g n8n@0.126.1
```

To install `next`:

```
npm install -g n8n@next
```

After the installation, start n8n by running:

```
n8n
# or
n8n start
```

Next steps

Try out n8n using the Quickstarts.

Updating

To update your n8n instance to the `latest` version, run:

```
npm update -g n8n
```

To install the `next` version:

```
npm install -g n8n@next
```

–8<– “_snippets/self-hosting/installation/tunnel.md”

Start n8n with `--tunnel` by running:

```
n8n start --tunnel
```

Reverting an upgrade

Install the older version that you want to go back to.

If the upgrade involved a database migration:

1. Check the feature documentation and release notes to see if there are any manual changes you need to make.
2. Run `n8n db:revert` on your current version to roll back the database. If you want to revert more than one database migration, you need to repeat this process.

Windows troubleshooting

If you are experiencing issues running n8n on Windows, make sure your Node.js environment is correctly set up. Follow Microsoft’s guide to Install NodeJS on Windows.

Docker Installation

n8n recommends using Docker for most self-hosting needs. It provides a clean, isolated environment, avoids operating system and tooling incompatibilities, and makes database and environment management simpler.

You can also use n8n in Docker with Docker Compose. You can find Docker Compose configurations for various architectures in the n8n-hosting repository.

–8<– “_snippets/self-hosting/warning.md”

You can also follow along with our video guide here:

Prerequisites

Before proceeding, install Docker:

- Docker Desktop is available for Mac, Windows, and Linux. Docker Desktop includes the Docker Engine and Docker Compose.
- Docker Engine and Docker Compose are also available as separate packages for Linux. Use this for Linux machines without a graphical environment or when you don’t want the Docker Desktop UI.

–8<– “_snippets/self-hosting/installation/latest-next-version.md”

Starting n8n

From your terminal, run the following commands, replacing the `<YOUR_TIMEZONE>` placeholders with your timezone:

```
docker volume create n8n_data

docker run -it --rm \
 --name n8n \
 -p 5678:5678 \
 -e GENERIC_TIMEZONE="<YOUR_TIMEZONE>" \
 -e TZ="<YOUR_TIMEZONE>" \
 -e N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true \
 -e N8N_RUNNERS_ENABLED=true \
 -v n8n_data:/home/node/.n8n \
 docker.n8n.io/n8nio/n8n
```

This command creates a volume to store persistent data, downloads the required n8n image, and starts the container with the following settings:

- Maps and exposes port `5678` on the host.
- Sets the timezone for the container:
  - the `TZ` environment variable sets the system timezone to control what scripts and commands like `date` return.
  - the `GENERIC_TIMEZONE` environment variable sets the correct timezone for schedule-oriented nodes like the Schedule Trigger node.

- Enforces secure file permissions for the n8n configuration file.
- Enables task runners, the recommended way of executing tasks in n8n.
- Mounts the `n8n_data` volume to the `/home/node/.n8n` directory to persist your data across container restarts.

Once running, you can access n8n by opening: http://localhost:5678

Using with PostgreSQL

By default, n8n uses SQLite to save credentials, past executions, and workflows. n8n also supports PostgreSQL, configurable using environment variables as detailed below.

To use n8n with PostgreSQL, execute the following commands, replacing the placeholders (depicted within angled brackets, for example `<POSTGRES_USER>`) with your actual values:

```
docker volume create n8n_data

docker run -it --rm \
 --name n8n \
 -p 5678:5678 \
 -e GENERIC_TIMEZONE="<YOUR_TIMEZONE>" \
 -e TZ="<YOUR_TIMEZONE>" \
 -e N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true \
 -e N8N_RUNNERS_ENABLED=true \
 -e DB_TYPE=postgresdb \
 -e DB_POSTGRESDB_DATABASE=<POSTGRES_DATABASE> \
 -e DB_POSTGRESDB_HOST=<POSTGRES_HOST> \
 -e DB_POSTGRESDB_PORT=<POSTGRES_PORT> \
 -e DB_POSTGRESDB_USER=<POSTGRES_USER> \
 -e DB_POSTGRESDB_SCHEMA=<POSTGRES_SCHEMA> \
 -e DB_POSTGRESDB_PASSWORD=<POSTGRES_PASSWORD> \
 -v n8n_data:/home/node/.n8n \
 docker.n8n.io/n8nio/n8n
```

You can find a complete `docker-compose` file for PostgreSQL in the n8n hosting repository.

Updating

To update n8n, in Docker Desktop, navigate to the **Images** tab and select **Pull** from the context menu to download the latest n8n image:

[Image: Docker Desktop]

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

```
# Pull latest (stable) version
docker pull docker.n8n.io/n8nio/n8n

# Pull specific version
docker pull docker.n8n.io/n8nio/n8n:1.81.0

# Pull next (unstable) version
docker pull docker.n8n.io/n8nio/n8n:next
```

After pulling the updated image, stop your n8n container and start it again. You can also use the command line. Replace `<container_id>` in the commands below with the container ID you find in the first command:

```
# Find your container ID
docker ps -a

# Stop the container with the `<container_id>`
docker stop <container_id>

# Remove the container with the `<container_id>`
docker rm <container_id>

# Start the container
docker run --name=<container_name> [options] -d
docker.n8n.io/n8nio/n8n
```

Updating Docker Compose

–8<– “_snippets/self-hosting/installation/docker-compose-updating.md”

–8<– “_snippets/self-hosting/installation/tunnel.md”

Start n8n with `--tunnel` by running:

```
docker volume create n8n_data

docker run -it --rm \
 --name n8n \
 -p 5678:5678 \
 -e GENERIC_TIMEZONE="<YOUR_TIMEZONE>" \
 -e TZ="<YOUR_TIMEZONE>" \
 -e N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS=true \
 -e N8N_RUNNERS_ENABLED=true \
 -v n8n_data:/home/node/.n8n \
 docker.n8n.io/n8nio/n8n \
 start --tunnel
```

Next steps

- Find more information about Docker setup in the README file for the Docker image. –8<– “_snippets/self-hosting/installation/server-setups-next-steps.md”

Server setups

Self-host with Docker Compose:

- Digital Ocean
- Heroku
- Hetzner Cloud

Self-host with Google Cloud Run (with access to n8n workflow tools for Google Workspace, e.g. Gmail, Drive):

- Google Cloud Run

Starting points for a Kubernetes setup:

- AWS
- Azure
- Google Kubernetes Engine (GKE)

Configuration guides to help you get started on other platforms:

- Docker Compose

Hosting n8n on DigitalOcean

This hosting guide shows you how to self-host n8n on a DigitalOcean droplet. It uses:

- Caddy (a reverse proxy) to allow access to the Droplet from the internet. Caddy will also automatically create and manage SSL / TLS certificates for your n8n instance.
- Docker Compose to create and define the application components and how they work together.

–8<– “_snippets/self-hosting/warning.md”

–8<– “_snippets/self-hosting/installation/latest-next-version.md”

Create a Droplet

1. Log in to DigitalOcean.
2. Select the project to host the Droplet, or create a new project.
3. In your project, select **Droplets** from the **Manage** menu.
4. Create a new Droplet using the Docker image available on the **Marketplace** tab.

Log in to your Droplet and create new user

The rest of this guide requires you to log in to the Droplet using a terminal with SSH. Refer to How to Connect to Droplets with SSH for more information.

You should create a new user, to avoid working as the root user:

1. Log in as root.
2. Create a new user: `shell adduser <username>`
3. Follow the prompts in the CLI to finish creating the user.
4. Grant the new user administrative privileges: `shell usermod -aG sudo <username>` You can now run commands with superuser privileges by using `sudo` before the command.
5. Follow the steps to set up SSH for the new user: Add Public Key Authentication.
6. Log out of the droplet.
7. Log in using SSH as the new user.

Clone configuration repository

Docker Compose, n8n, and Caddy require a series of folders and configuration files. You can clone these from this repository into the home folder of the logged-in user on your Droplet. The following steps will tell you which file to change and what changes to make.

Clone the repository with the following command:

```
git clone https://github.com/n8n-io/n8n-docker-caddy.git
```

And change directory to the root of the repository you cloned:

```
cd n8n-docker-caddy
```

Default folders and files

The host operating system (the DigitalOcean Droplet) copies the two folders you created to Docker containers to make them available to Docker. The two folders are:

- `caddy_config`: Holds the Caddy configuration files.
- `local_files`: A folder for files you upload or add using n8n.

Create Docker volumes

To persist the Caddy cache between restarts and speed up start times, create a Docker volume that Docker reuses between restarts:

```
sudo docker volume create caddy_data
```

Create a Docker volume for the n8n data:

```
sudo docker volume create n8n_data
```

Set up DNS

n8n typically operates on a subdomain. Create a DNS record with your provider for the subdomain and point it to the IP address of the Droplet. The exact steps for this depend on your DNS provider, but typically you need to create a new “A” record for the n8n subdomain. DigitalOcean provide An Introduction to DNS Terminology, Components, and Concepts.

Open ports

n8n runs as a web application, so the Droplet needs to allow incoming access to traffic on port 80 for non-secure traffic, and port 443 for secure traffic.

Open the following ports in the Droplet’s firewall by running the following two commands:

```
sudo ufw allow 80
sudo ufw allow 443
```

Configure n8n

n8n needs some environment variables set to pass to the application running in the Docker container. The example `.env` file contains placeholders you need to replace with values of your own.

Open the file with the following command:

```
nano .env
```

The file contains inline comments to help you know what to change.

Refer to Environment variables for n8n environment variables details.

The Docker Compose file

The Docker Compose file (`docker-compose.yml`) defines the services the application needs, in this case Caddy and n8n.

- The Caddy service definition defines the ports it uses and the local volumes to copy to the containers.
- The n8n service definition defines the ports it uses, the environment variables n8n needs to run (some defined in the `.env` file), and the volumes it needs to copy to the containers.

The Docker Compose file uses the environment variables set in the `.env` file, so you shouldn't need to change it's content, but to take a look, run the following command:

```
nano docker-compose.yml
```

Configure Caddy

Caddy needs to know which domains it should serve, and which port to expose to the outside world. Edit the `Caddyfile` file in the `caddy_config` folder.

```
nano caddy_config/Caddyfile
```

Change the placeholder domain to yours. If you followed the steps to name the subdomain n8n, your full domain is similar to `n8n.example.com`. The `n8n` in the `reverse_proxy` setting tells Caddy to use the service definition defined in the `docker-compose.yml` file:

```
n8n.<domain>.<suffix> {
    reverse_proxy n8n:5678 {
      flush_interval -1
    }
}
```

If you were to use `automate.example.com`, your `Caddyfile` may look something like:

```
automate.example.com {
    reverse_proxy n8n:5678 {
      flush_interval -1
    }
}
```

Start Docker Compose

Start n8n and Caddy with the following command:

```
sudo docker compose up -d
```

This may take a few minutes.

Test your setup

In your browser, open the URL formed of the subdomain and domain name defined earlier. Enter the user name and password defined earlier, and you should be able to access n8n.

Stop n8n and Caddy

You can stop n8n and Caddy with the following command:

```
sudo docker compose stop
```

Updating

–8<– “_snippets/self-hosting/installation/docker-compose-updating.md”

Next steps

–8<– “_snippets/self-hosting/installation/server-setups-next-steps.md”

Hosting n8n on Heroku

This hosting guide shows you how to self-host n8n on Heroku. It uses:

- Docker Compose to create and define the application components and how they work together.
- Heroku’s PostgreSQL service to host n8n’s data storage.
- A **Deploy to Heroku** button offering a one click, with minor configuration, deployment.

–8<– “_snippets/self-hosting/warning.md”

–8<– “_snippets/self-hosting/installation/latest-next-version.md”

Use the deployment template to create a Heroku project

The quickest way to get started with deploying n8n to Heroku is using the **Deploy to Heroku** button:

[Image: Deploy]

This opens the **Create New App** page on Heroku. Set a name for the project, and choose the region to deploy the project to.

Configure environment variables

Heroku pre-fills the configuration options defined in the `env` section of the `app.json` file, which also sets default values for the environment variables n8n uses.

You can change any of these values to suit your needs. You must change the following values:

- **N8N_ENCRYPTION_KEY**, which n8n uses to encrypt user account details before saving to the database.
- **WEBHOOK_URL** should match the application name you create to ensure that webhooks have the correct URL.

Deploy n8n

Select **Deploy app**.

After Heroku builds and deploys the app it provides links to **Manage App** or **View** the application.

Changing the deployment template

You can make changes to the deployment template by forking the repository and deploying from you fork.

The Dockerfile

By default the Dockerfile pulls the latest n8n image, if you want to use a different or fixed version, then update the image tag on the top line of the `Dockerfile`.

Heroku and exposing ports

Heroku doesn't allow Docker-based applications to define an exposed port with the `EXPOSE` command. Instead, Heroku provides a `PORT` environment variable that it dynamically populates at application runtime. The `entrypoint.sh` file overrides the default Docker image command to instead set the port variable that Heroku provides. You can then access n8n on port 80 in a web browser.

Configuring Heroku

The `heroku.yml` file defines the application you want to create on Heroku. It consists of two sections:

- `setup` > `addons` defines the Heroku addons to use. In this case, the PostgreSQL database addon.
- The `build` section defines how Heroku builds the application. In this case it uses the Docker buildpack to build a `web` service based on the supplied `Dockerfile`.

Next steps

–8<– “_snippets/self-hosting/installation/server-setups-next-steps.md”

Hosting n8n on Hetzner cloud

This hosting guide shows you how to self-host n8n on a Hetzner cloud server. It uses:

- Caddy (a reverse proxy) to allow access to the Server from the internet.
- Docker Compose to create and define the application components and how they work together.

–8<– “_snippets/self-hosting/warning.md”

–8<– “_snippets/self-hosting/installation/latest-next-version.md”

Create a server

1. Log in to the Hetzner Cloud Console.
2. Select the project to host the server, or create a new project by selecting **+ NEW PROJECT**.
3. Select **+ CREATE SERVER** on the project tile you want to add it to.

You can change most of the settings to suit your needs, but as this guide uses Docker to run the application, under the **Image** section, select “Docker CE” from the **APPS** tab.

Log in to your server

The rest of this guide requires you to log in to the server using a terminal with SSH. Refer to Access with SSH/rsync/BorgBackup for more information. You can find the public IP in the listing of the servers in your project.

Install Docker Compose

The Hetzner Docker app image doesn’t have Docker compose installed. Install it with the following commands:

```
apt update && apt -y upgrade
apt install docker-compose-plugin
```

Clone configuration repository

Docker Compose, n8n, and Caddy require a series of folders and configuration files. You can clone these from this repository into the root user folder of the server. The following steps will tell you which file to change and what changes to make.

Clone the repository with the following command:

```
git clone https://github.com/n8n-io/n8n-docker-caddy.git
```

And change directory to the root of the repository you cloned:

```
cd n8n-docker-caddy
```

Default folders and files

The host operating system (the server) copies the two folders you created to Docker containers to make them available to Docker. The two folders are:

- `caddy_config`: Holds the Caddy configuration files.
- `local_files`: A folder for files you upload or add using n8n.

Create Docker volume

To persist the Caddy cache between restarts and speed up start times, create a Docker volume that Docker reuses between restarts:

```
docker volume create caddy_data
```

Create a Docker volume for the n8n data:

```
sudo docker volume create n8n_data
```

Set up DNS

n8n typically operates on a subdomain. Create a DNS record with your provider for the subdomain and point it to the IP address of the server. The exact steps for this depend on your DNS provider, but typically you need to create a new “A” record for the n8n subdomain. DigitalOcean provide An Introduction to DNS Terminology, Components, and Concepts.

Open ports

n8n runs as a web application, so the server needs to allow incoming access to traffic on port 80 for non-secure traffic, and port 443 for secure traffic.

Open the following ports in the server’s firewall by running the following two commands:

```
sudo ufw allow 80
sudo ufw allow 443
```

Configure n8n

n8n needs some environment variables set to pass to the application running in the Docker container. The example `.env` file contains placeholders you need to replace with values of your own.

Open the file with the following command:

```
nano .env
```

The file contains inline comments to help you know what to change.

Refer to Environment variables for n8n environment variables details.

The Docker Compose file

The Docker Compose file (`docker-compose.yml`) defines the services the application needs, in this case Caddy and n8n.

- The Caddy service definition defines the ports it uses and the local volumes to copy to the containers.
- The n8n service definition defines the ports it uses, the environment variables n8n needs to run (some defined in the `.env` file), and the volumes it needs to copy to the containers.

The Docker Compose file uses the environment variables set in the `.env` file, so you shouldn't need to change it's content, but to take a look, run the following command:

```
nano docker-compose.yml
```

Configure Caddy

Caddy needs to know which domains it should serve, and which port to expose to the outside world. Edit the `Caddyfile` file in the `caddy_config` folder.

```
nano caddy_config/Caddyfile
```

Change the placeholder subdomain to yours. If you followed the steps to name the subdomain n8n, your full domain is similar to `n8n.example.com`. The `n8n` in the `reverse_proxy` setting tells Caddy to use the service definition defined in the `docker-compose.yml` file:

```
n8n.<domain>.<suffix> {
    reverse_proxy n8n:5678 {
      flush_interval -1
    }
}
```

Start Docker Compose

Start n8n and Caddy with the following command:

```
docker compose up -d
```

This may take a few minutes.

Test your setup

In your browser, open the URL formed of the subdomain and domain name defined earlier. Enter the user name and password defined earlier, and you should be able to access n8n.

Stop n8n and Caddy

You can stop n8n and Caddy with the following command:

```
sudo docker compose stop
```

Updating

–8<– “_snippets/self-hosting/installation/docker-compose-updating.md”

Next steps

–8<– “_snippets/self-hosting/installation/server-setups-next-steps.md”

Hosting n8n on Amazon Web Services

This hosting guide shows you how to self-host n8n with Amazon Web Services (AWS). It uses n8n with Postgres as a database backend using Kubernetes to manage the necessary resources and reverse proxy.

Hosting options

AWS offers several ways suitable for hosting n8n, including EC2 (virtual machines), and EKS (containers running with Kubernetes).

This guide uses EKS as the hosting option. Using Kubernetes requires some additional complexity and configuration, but is the best method for scaling n8n as demand changes.

Prerequisites

The steps in this guide use a mix of the AWS UI and the eksctl CLI tool for EKS.

While not mentioned in the documentation for eksctl, you also need to install the AWS CLI tool, and configure authentication of the tool.

–8<– “_snippets/self-hosting/warning.md”

–8<– “_snippets/self-hosting/installation/latest-next-version.md”

Create a cluster

Use the eksctl tool to create a cluster specifying a name and a region with the following command:

```
eksctl create cluster --name n8n --region <your-aws-region>
```

This can take a while to create the cluster.

Once the cluster is created, eksctl automatically sets the kubectl context to the cluster.

Clone configuration repository

Kubernetes and n8n require a series of configuration files. You can clone these from this repository. The following steps tell you what each file does, and what settings you need to change.

Clone the repository with the following command:

```
git clone https://github.com/n8n-io/n8n-hosting.git
```

And change directory:

```
cd n8n-hosting/kubernetes
```

Configure Postgres

For larger scale n8n deployments, Postgres provides a more robust database backend than SQLite.

Configure volume for persistent storage

To maintain data between pod restarts, the Postgres deployment needs a persistent volume. The default AWS storage class, gp3, is suitable for this purpose. This is defined in the `postgres-claim0-persistentvolumeclaim.yaml` manifest.

```
…
spec:
  storageClassName: gp3
  accessModes:
    - ReadWriteOnce
…
```

Postgres environment variables

Postgres needs some environment variables set to pass to the application running in the containers.

The example `postgres-secret.yaml` file contains placeholders you need to replace with values of your own for user details and the database to use.

PostgreSQL uses a root user (`POSTGRES_USER`) for setup and administration, but it’s best practice to create a separate non-root user (`POSTGRES_NON_ROOT_USER`) for n8n. The root user has full control, while n8n only needs the non-root user permissions to run. Configuring both improves security and helps prevent accidental changes to the database system.

The `postgres-deployment.yaml` manifest then uses the values from this manifest file to send to the application pods.

Configure n8n

Create a volume for file storage

While not essential for running n8n, using persistent volumes helps maintain files uploaded while using n8n and if you want to persist manual n8n encryption keys between restarts, which saves a file containing the key into file storage during startup.

The `n8n-claim0-persistentvolumeclaim.yaml` manifest creates this, and the n8n Deployment mounts that claim in the `volumes` section of the `n8n-deployment.yaml` manifest.

```
…
volumes:
  - name: n8n-claim0
    persistentVolumeClaim:
      claimName: n8n-claim0
…
```

Pod resources

Kubernetes lets you specify the minimum resources application containers need and the limits they can run to. The example YAML files cloned above contain the following in the `resources` section of the `n8n-deployment.yaml` file:

```
…
resources:
  requests:
    memory: "250Mi"
  limits:
    memory: "500Mi"
…
```

This defines a minimum of 250mb per container, a maximum of 500mb, and lets Kubernetes handle CPU. You can change these values to match your own needs. As a guide, here are the resources values for the n8n cloud offerings:

–8<– “_snippets/self-hosting/installation/suggested-pod-resources.md”

Optional: Environment variables

You can configure n8n settings and behaviors using environment variables.

Create an `n8n-secret.yaml` file. Refer to Environment variables for n8n environment variables details.

Deployments

The two deployment manifests (`n8n-deployment.yaml` and `postgres-deployment.yaml`) define the n8n and Postgres applications to Kubernetes.

The manifests define the following:

- Send the environment variables defined to each application pod
- Define the container image to use
- Set resource consumption limits
- The `volumes` defined earlier and `volumeMounts` to define the path in the container to mount volumes.
- Scaling and restart policies. The example manifests define one instance of each pod. You should change this to meet your needs.

Services

The two service manifests (`postgres-service.yaml` and `n8n-service.yaml`) expose the services to the outside world using the Kubernetes load balancer using ports 5432 and 5678 respectively by default.

Send to Kubernetes cluster

Send all the manifests to the cluster by running the following command in the `n8n-kubernetes-hosting` directory:

```
kubectl apply -f .
```

Set up DNS

n8n typically operates on a subdomain. Create a DNS record with your provider for the subdomain and point it to a static address of the instance.

To find the address of the n8n service running on the instance:

1. Open the **Clusters** section of the **Amazon Elastic Kubernetes Service** page in the AWS console.
2. Select the name of the cluster to open its configuration page.
3. Select the **Resources** tab, then **Service and networking** > **Services**.
4. Select the **n8n** service and copy the **Load balancer URLs** value. Use this value suffixed with the n8n service port (5678) for DNS.

Delete resources

If you need to delete the setup, you can remove the resources created by the manifests with the following command:

```
kubectl delete -f .
```

Next steps

–8<– “_snippets/self-hosting/installation/server-setups-next-steps.md”

Hosting n8n on Azure

This hosting guide shows you how to self-host n8n on Azure. It uses n8n with Postgres as a database backend using Kubernetes to manage the necessary resources and reverse proxy.

Prerequisites

You need The Azure command line tool

–8<– “_snippets/self-hosting/warning.md”

–8<– “_snippets/self-hosting/installation/latest-next-version.md”

Hosting options

Azure offers several ways suitable for hosting n8n, including Azure Container Instances (optimized for running containers), Linux Virtual Machines, and Azure Kubernetes Service (containers running with Kubernetes).

This guide uses the Azure Kubernetes Service (AKS) as the hosting option. Using Kubernetes requires some additional complexity and configuration, but is the best method for scaling n8n as demand changes.

The steps in this guide use a mix of the Azure UI and command line tool, but you can use either to accomplish most tasks.

Open the Azure Kubernetes Service

From the Azure portal select **Kubernetes services**.

Create a cluster

From the Kubernetes services page, select **Create > Create a Kubernetes cluster**.

You can select any of the configuration options that suit your needs, then select **Create** when done.

Set Kubectl context

The remainder of the steps in this guide require you to set the Azure instance as the Kubectl context. You can find the connection details for a cluster instance by opening its details page and then the **Connect** button. The resulting code snippets shows the steps to paste and run into a terminal to change your local Kubernetes settings to use the new cluster.

Clone configuration repository

Kubernetes and n8n require a series of configuration files. You can clone these from this repository. The following steps tell you which file configures what and what you need to change.

Clone the repository with the following command:

```
git clone https://github.com/n8n-io/n8n-hosting.git
```

And change directory:

```
cd n8n-hosting/kubernetes
```

Configure Postgres

For larger scale n8n deployments, Postgres provides a more robust database backend than SQLite.

Configure volume for persistent storage

To maintain data between pod restarts, the Postgres deployment needs a persistent volume. The default storage class is suitable for this purpose and is defined in the `postgres-claim0-persistentvolumeclaim.yaml` manifest.

Postgres environment variables

Postgres needs some environment variables set to pass to the application running in the containers.

The example `postgres-secret.yaml` file contains placeholders you need to replace with your own values. Postgres will use these details when creating the database..

The `postgres-deployment.yaml` manifest then uses the values from this manifest file to send to the application pods.

Configure n8n

Create a volume for file storage

While not essential for running n8n, using persistent volumes is required for:

- Using nodes that interact with files, such as the binary data node.
- If you want to persist manual n8n encryption keys between restarts. This saves a file containing the key into file storage during startup.

The `n8n-claim0-persistentvolumeclaim.yaml` manifest creates this, and the n8n Deployment mounts that claim in the `volumes` section of the `n8n-deployment.yaml` manifest.

```
…
volumes:
  - name: n8n-claim0
    persistentVolumeClaim:
      claimName: n8n-claim0
…
```

Pod resources

Kubernetes lets you optionally specify the minimum resources application containers need and the limits they can run to. The example YAML files cloned above contain the following in the `resources` section of the `n8n-deployment.yaml` file:

```
…
resources:
  requests:
    memory: "250Mi"
  limits:
    memory: "500Mi"
…
```

This defines a minimum of 250mb per container, a maximum of 500mb, and lets Kubernetes handle CPU. You can change these values to match your own needs. As a guide, here are the resources values for the n8n cloud offerings:

–8<– “_snippets/self-hosting/installation/suggested-pod-resources.md”

Optional: Environment variables

You can configure n8n settings and behaviors using environment variables.

Create an `n8n-secret.yaml` file. Refer to Environment variables for n8n environment variables details.

Deployments

The two deployment manifests (`n8n-deployment.yaml` and `postgres-deployment.yaml`) define the n8n and Postgres applications to Kubernetes.

The manifests define the following:

- Send the environment variables defined to each application pod
- Define the container image to use
- Set resource consumption limits with the `resources` object
- The `volumes` defined earlier and `volumeMounts` to define the path in the container to mount volumes.
- Scaling and restart policies. The example manifests define one instance of each pod. You should change this to meet your needs.

Services

The two service manifests (`postgres-service.yaml` and `n8n-service.yaml`) expose the services to the outside world using the Kubernetes load balancer using ports 5432 and 5678 respectively.

Send to Kubernetes cluster

Send all the manifests to the cluster with the following command:

```
kubectl apply -f .
```

Set up DNS

n8n typically operates on a subdomain. Create a DNS record with your provider for the subdomain and point it to the IP address of the n8n service. Find the IP address of the n8n service from the **Services & ingresses** menu item of the cluster you want to use under the **External IP** column. You need to add the n8n port, “5678” to the URL.

Delete resources

Remove the resources created by the manifests with the following command:

```
kubectl delete -f .
```

Next steps

–8<– “_snippets/self-hosting/installation/server-setups-next-steps.md”

Hosting n8n on Google Cloud Run

This hosting guide shows you how to self-host n8n on Google Cloud Run, a serverless container runtime. If you’re just getting started with n8n and don’t need a production-grade deployment, you can go with the “easy mode” option below for deployment. Otherwise, if you intend to use this n8n deployment at-scale, refer to the “durable mode” instructions further down.

You can also enable access via OAuth to Google Workspace, such as Gmail and Drive, to use these services as n8n workflow tools. Instructions for granting n8n access to these services are at the end of of this documentation.

If you want to deploy to Google Kubernetes Engine (GKE) instead, you can refer to these instructions.

–8<– “_snippets/self-hosting/warning.md”

–8<– “_snippets/self-hosting/installation/latest-next-version.md”

Before you begin: get a Google Cloud project

If you have not yet created a Google Cloud project, do this first (and ensure you have billing enabled on the project; even if your Cloud Run service runs for free you must have billing activated to deploy). Otherwise, navigate to the project where you want to deploy n8n.

Easy mode

This is the fastest way to deploy n8n on Cloud Run. For this deployment, n8n's data is in-memory so this is only recommended for demo purposes. **Anytime this Cloud Run service scales to zero or is redeployed, the n8n data will be lost.** Refer to the durable mode instructions below if you need a production-grade deployment.

If you have not yet created a Google Cloud project, do this first (and ensure you have billing enabled on the project; even if your Cloud Run service will run for free you must have billing enabled to activated to deploy). Otherwise, navigate to the project where you want to deploy n8n.

Open the Cloud Shell Terminal (on the Google Cloud console, either type "G" then "S" or click on the terminal icon on the upper right).

Once your session is open, you may need to run this command first to login (and follow the steps it asks you to complete):

```
gcloud auth login
```

You can also explicitly enable the Cloud Run API (even if you don't do this, it will ask if you want this enabled when you deploy):

```
gcloud services enable run.googleapis.com
```

To deploy n8n:

```
gcloud run deploy n8n \
    --image=n8nio/n8n \
    --region=us-west1 \
    --allow-unauthenticated \
    --port=5678 \
    --no-cpu-throttling \
    --memory=2Gi
```

(you can specify whichever region you prefer, instead of "us-west1")

Once the deployment finishes, open another tab to navigate to the Service URL. n8n may still be loading and you will see a "n8n is starting up. Please wait" message, but shortly thereafter you should see the n8n login screen.

Optional: If you want to keep this n8n service running for as long as possible to avoid data loss, you can also set manual scale to 1 to prevent it from autoscaling to 0.

```
gcloud run deploy n8n \
    --image=n8nio/n8n \
    --region=us-west1 \
    --allow-unauthenticated \
    --port=5678 \
    --no-cpu-throttling \
    --memory=2Gi \
    --scaling=1
```

This does not prevent data loss completely, such as whenever the Cloud Run service is re-deployed/updated. If you want truly persistant data, you should refer to the instructions below for how to attach a database.

Durable mode

The following instructions are intended for a more durable, production-grade deployment of n8n on Cloud Run. It includes resources such as a database for persistance and secret manager for sensitive data.

If you want to deploy the following setup via Terraform, refer to this example which deploys the same setup as the following (without the OAuth setup for Google Workspace tools).

Enable APIs and set env vars

Open the Cloud Shell Terminal (on the Google Cloud console, either type “G” then “S” or click on the terminal icon on the upper right) and run these commands in the terminal session:

```
## You may need to login first
gcloud auth login

gcloud services enable run.googleapis.com
gcloud services enable sqladmin.googleapis.com
gcloud services enable secretmanager.googleapis.com
```

You’ll also want to set some environment variables for the remainder of these instructions:

```
export PROJECT_ID=your-project
export REGION=region-where-you-want-this-deployed
```

Setup your Postgres database

Run this command to create the Postgres DB instance (it will take a few minutes to complete; also ensure you update the root-password field with your own desired password):

```
gcloud sql instances create n8n-db \
    --database-version=POSTGRES_13 \
    --tier=db-f1-micro \
    --region=$REGION \
    --root-password="change-this-password" \
    --storage-size=10GB \
    --availability-type=ZONAL \
    --no-backup \
    --storage-type=HDD
```

Once complete, you can add the database that n8n will use:

```
gcloud sql databases create n8n --instance=n8n-db
```

Create the DB user for n8n (change the password value, of course):

```
gcloud sql users create n8n-user \
    --instance=n8n-db \
    --password="change-this-password"
```

You can save the password you set for this n8n-user to a file for the next step of saving the password in Secret Manager. Be sure to delete this file later.

Store sensitive data in Secret Manager

While not required, it's absolutely recommended to store your sensitive data in Secrets Manager.

Create a secret for the database password (replace "/your/password/file" with the file you created above for the n8n-user password):

```
gcloud secrets create n8n-db-password \
    --data-file=/your/password/file \
    --replication-policy="automatic"
```

Create an encryption key (you can use your own, this example generates a random one):

```
openssl rand -base64 -out my-encryption-key 42
```

Create a secret for this encryption key (replace "my-encryption-key" if you are supplying your own):

```
gcloud secrets create n8n-encryption-key \
    --data-file=my-encryption-key \
    --replication-policy="automatic"
```

Now you can delete my-encryption-key and the database password files you created. These values are now securely stored in Secret Manager.

Create a service account for Cloud Run

You want this Cloud Run service to be restricted to access only the resources it needs. The following commands create the service account and adds the permissions necessary to access secrets and the database:

```
gcloud iam service-accounts create n8n-service-account \
    --display-name="n8n Service Account"

gcloud secrets add-iam-policy-binding n8n-db-password \
    --member="serviceAccount:n8n-service-
account@$PROJECT_ID.iam.gserviceaccount.com" \
    --role="roles/secretmanager.secretAccessor"

gcloud secrets add-iam-policy-binding n8n-encryption-key \
    --member="serviceAccount:n8n-service-
account@$PROJECT_ID.iam.gserviceaccount.com" \
    --role="roles/secretmanager.secretAccessor"

gcloud projects add-iam-policy-binding $PROJECT_ID \
    --member="serviceAccount:n8n-service-
account@$PROJECT_ID.iam.gserviceaccount.com" \
    --role="roles/cloudsql.client"
```

Deploy the Cloud Run service

Now you can deploy your n8n service:

```
gcloud run deploy n8n \
```

```
            --image=n8nio/n8n:latest \
            --command="/bin/sh" \
            --args="-c,sleep 5;n8n start" \
            --region=$REGION \
            --allow-unauthenticated \
            --port=5678 \
            --memory=2Gi \
            --no-cpu-throttling \
            --set-env-
vars="N8N_PORT=5678,N8N_PROTOCOL=https,DB_TYPE=postgresdb,DB_POSTGRE
SDB_DATABASE=n8n,DB_POSTGRESDB_USER=n8n-
user,DB_POSTGRESDB_HOST=/cloudsql/$PROJECT_ID:$REGION:n8n-
db,DB_POSTGRESDB_PORT=5432,DB_POSTGRESDB_SCHEMA=public,GENERIC_TIMEZ
ONE=UTC,QUEUE_HEALTH_CHECK_ACTIVE=true" \
            --set-secrets="DB_POSTGRESDB_PASSWORD=n8n-db-
password:latest,N8N_ENCRYPTION_KEY=n8n-encryption-key:latest" \
            --add-cloudsql-instances=$PROJECT_ID:$REGION:n8n-db \
            --service-account=n8n-service-
account@$PROJECT_ID.iam.gserviceaccount.com
```

Once the deployment finishes, open another tab to navigate to the Service URL. You should see the n8n login screen.

Troubleshooting

If you see a “Cannot GET /” screen this usually indicates that n8n is still starting up. You can refresh the page and it should eventually load.

(Optional) Enabling Google Workspace services as n8n tools

If you want to use Google Workspace services (Gmail, Calendar, Drive, etc.) as tools in n8n, it’s recommended to setup OAuth to access these services.

First ensure the respective APIs you want are enabled:

```
        ## Enable whichever APIs you need
        ## Note: If you want Sheets/Docs, it's not enough to just enable
Drive; these services each have their own API
        gcloud services enable gmail.googleapis.com
        gcloud services enable drive.googleapis.com
        gcloud services enable sheets.googleapis.com
        gcloud services enable docs.googleapis.com
        gcloud services enable calendar-json.googleapis.com
```

Re-deploy n8n on Cloud Run with the necessary OAuth callback URLs as environment variables:

```
        export SERVICE_URL="your-n8n-service-URL"
        ## e.g. https://n8n-12345678.us-west1.run.app

        gcloud run services update n8n \
            --region=$REGION \
            --update-env-vars="N8N_HOST=$(echo $SERVICE_URL | sed
's/https:\/\///'),WEBHOOK_URL=$SERVICE_URL,N8N_EDITOR_BASE_URL=$SERV
ICE_URL"
```

Lastly, you must setup OAuth for these services. Visit `https://console.cloud.google.com/auth` and follow these steps:

1. Click “Get Started” if this button shows (when you have not yet setup OAuth in this Cloud project).
2. For “App Information”, enter whichever “App Name” and “User Support Email” you prefer.
3. For “Audience”, select “Internal” if you intend to only enable access to your user(s) within this same Google Workspace. Otherwise, you can select “External”.
4. Enter “Contact Information”.
5. If you selected “External”, then click “Audience” and add any test users you need to grant access.
6. Click “Clients” > “Create client”, select “Web application” for “Application type”, enter your n8n service URL into “Authorized JavaScript origins”, and “/rest/oauth2-credential/callback” into “Authorized redirect URIs” where your YOUR-N8N-URL is also the n8n service URL (e.g. `https://n8n-12345678.us-west1.run.app/rest/oauth2-credential/callback`). Make sure you download the created client’s JSON file since it contains the client secret which you will not be able to see later in the Console.
7. Click “Data Access” and add the scopes you want n8n to have access for (e.g. to access Google Sheets, you need `https://googleapis.com/auth/drive.file` and `https://googleapis.com/auth/spreadsheets`)
8. Now you should be able to use these workspace services. You can test if it works by logging into n8n, add a Tool for the respective service and add its credentials using the information in the OAuth client JSON file from step 6.

Hosting n8n on Google Kubernetes Engine

Google Cloud offers several options suitable for hosting n8n, including Cloud Run (optimized for running containers), Compute Engine (VMs), and Kubernetes Engine (containers running with Kubernetes).

This guide uses the Google Kubernetes Engine (GKE) as the hosting option. If you want to use Cloud Run, refer to these instructions.

Most of the steps in this guide use the Google Cloud UI, but you can also use the gcloud command line tool instead to undertake all the steps.

Prerequisites

- The gcloud command line tool
- The gke-gcloud-auth-plugin (install the gcloud CLI first)

–8<– “_snippets/self-hosting/warning.md”

–8<– “_snippets/self-hosting/installation/latest-next-version.md”

Create project

GCP encourages you to create projects to logically organize resources and configuration. Create a new project for your n8n deployment from your Google Cloud Console: select the project dropdown menu and then the **NEW PROJECT** button. Then select the newly created project. As you follow the other steps in this guide, make sure you have the correct project selected.

Enable the Kubernetes Engine API

GKE isn't enabled by default. Search for "Kubernetes" in the top search bar and select "Kubernetes Engine" from the results.

Select **ENABLE** to enable the Kubernetes Engine API for this project.

Create a cluster

From the GKE service page, select **Clusters > CREATE**. Make sure you select the "Standard" cluster option, n8n doesn't work with an "Autopilot" cluster. You can leave the cluster configuration on defaults unless there's anything specifically you need to change, such as location.

Set Kubectl context

The rest of the steps in this guide require you to set the GCP instance as the Kubectl context. You can find the connection details for a cluster instance by opening its details page and selecting **CONNECT**. The displayed code snippet shows a connection string for the gcloud CLI tool. Paste and run the code snippet in the gcloud CLI to change your local Kubernetes settings to use the new gcloud cluster.

Clone configuration repository

Kubernetes and n8n require a series of configuration files. You can clone these from this repository locally. The following steps explain the file configuration and how to add your information.

Clone the repository with the following command:

```
git clone https://github.com/n8n-io/n8n-hosting.git
```

And change directory:

```
cd n8n-hosting/kubernetes
```

Configure Postgres

For larger scale n8n deployments, Postgres provides a more robust database backend than SQLite.

Create a volume for persistent storage

To maintain data between pod restarts, the Postgres deployment needs a persistent volume. Running Postgres on GCP requires a specific Kubernetes Storage Class. You can read this guide for specifics, but the `storage.yaml` manifest creates it for you. You may want to change the regions to create the storage in under the `allowedTopologies` > `matchedLabelExpressions` > `values` key. By default, they're set to `us-central`.

```
…
allowedTopologies:
  - matchLabelExpressions:
      - key: failure-domain.beta.kubernetes.io/zone
        values:
          - us-central1-b
          - us-central1-c
```

Postgres environment variables

Postgres needs some environment variables set to pass to the application running in the containers.

The example `postgres-secret.yaml` file contains placeholders you need to replace with your own values. Postgres will use these details when creating the database..

The `postgres-deployment.yaml` manifest then uses the values from this manifest file to send to the application pods.

Configure n8n

Create a volume for file storage

While not essential for running n8n, using persistent volumes is required for:

- Using nodes that interact with files, such as the binary data node.
- If you want to persist manual n8n encryption keys between restarts. This saves a file containing the key into file storage during startup.

The `n8n-claim0-persistentvolumeclaim.yaml` manifest creates this, and the n8n Deployment mounts that claim in the `volumes` section of the `n8n-deployment.yaml` manifest.

```
…
volumes:
  - name: n8n-claim0
    persistentVolumeClaim:
      claimName: n8n-claim0
…
```

Pod resources

Kubernetes lets you optionally specify the minimum resources application containers need and the limits they can run to. The example YAML files cloned above contain the following in the `resources` section of the `n8n-deployment.yaml` and `postgres-deployment.yaml` files:

```
…
resources:
  requests:
    memory: "250Mi"
  limits:
    memory: "500Mi"
…
```

This defines a minimum of 250mb per container, a maximum of 500mb, and lets Kubernetes handle CPU. You can change these values to match your own needs. As a guide, here are the resources values for the n8n cloud offerings:

–8<– “_snippets/self-hosting/installation/suggested-pod-resources.md”

Optional: Environment variables

You can configure n8n settings and behaviors using environment variables.

Create an `n8n-secret.yaml` file. Refer to Environment variables for n8n environment variables details.

Deployments

The two deployment manifests (`n8n-deployment.yaml` and `postgres-deployment.yaml`) define the n8n and Postgres applications to Kubernetes.

The manifests define the following:

- Send the environment variables defined to each application pod
- Define the container image to use
- Set resource consumption limits with the `resources` object
- The `volumes` defined earlier and `volumeMounts` to define the path in the container to mount volumes.
- Scaling and restart policies. The example manifests define one instance of each pod. You should change this to meet your needs.

Services

The two service manifests (`postgres-service.yaml` and `n8n-service.yaml`) expose the services to the outside world using the Kubernetes load balancer using ports 5432 and 5678 respectively.

Send to Kubernetes cluster

Send all the manifests to the cluster with the following command:

```
kubectl apply -f .
```

Set up DNS

n8n typically operates on a subdomain. Create a DNS record with your provider for the subdomain and point it to the IP address of the n8n service. Find the IP address of the n8n service from the **Services & Ingress** menu item of the cluster you want to use under the **Endpoints** column.

Delete resources

Remove the resources created by the manifests with the following command:

```
kubectl delete -f .
```

Next steps

–8<– “_snippets/self-hosting/installation/server-setups-next-steps.md”

Docker-Compose

These instructions cover how to run n8n on a Linux server using Docker Compose.

If you have already installed Docker and Docker-Compose, then you can start with step 3.

You can find Docker Compose configurations for various architectures in the n8n-hosting repository.

–8<– “_snippets/self-hosting/warning.md”

–8<– “_snippets/self-hosting/installation/latest-next-version.md”

1. Install Docker and Docker Compose

The way that you install Docker and Docker Compose depends on your Linux distribution. You can find specific instructions for each component in the links below:

- Docker Engine
- Docker Compose

After following the installation instructions, verify that Docker and Docker Compose are available by typing:

```
docker --version
docker compose version
```

2. Optional: Non-root user access

You can optionally grant access to run Docker without the `sudo` command.

To grant access to the user that you're currently logged in with (assuming they have `sudo` access), run:

```
sudo usermod -aG docker ${USER}
# Register the `docker` group membership with current session
without changing your primary group
exec sg docker newgrp
```

To grant access to a different user, type the following, substituting `<USER_TO_RUN_DOCKER>` with the appropriate username:

```
sudo usermod -aG docker <USER_TO_RUN_DOCKER>
```

You will need to run `exec sg docker newgrp` from any of that user's existing sessions for it to access the new group permissions.

You can verify that your current session recognizes the `docker` group by typing:

```
groups
```

3. DNS setup

To host n8n online or on a network, create a dedicated subdomain pointed at your server.

Add an A record to route the subdomain accordingly:

Record type	Name	Destination
A	`n8n` (or your desired subdomain)	`<your_server_IP_address>`

4. Create an `.env` file

Create a project directory to store your n8n environment configuration and Docker Compose files and navigate inside:

```
mkdir n8n-compose
cd n8n-compose
```

Inside the `n8n-compose` directory, create an `.env` file to customize your n8n instance's details. Change it to match your own information:

```shell title=".env file" # DOMAIN_NAME and SUBDOMAIN together determine where n8n will be reachable from # The top level domain to serve from DOMAIN_NAME=example.com

# The subdomain to serve from

SUBDOMAIN=n8n

# The above example serve n8n at: https://n8n.example.com
```

Optional timezone to set which gets used by Cron and other scheduling nodes

New York is the default value if not set

GENERIC_TIMEZONE=Europe/Berlin

The email address to use for the TLS/SSL certificate creation

SSL_EMAIL=user@example.com

```
## 5. Create local files directory

Inside your project directory, create a directory called `local-
files` for sharing files between the n8n instance and the host
system (for example, using the [Read/Write Files from Disk node]
(/integrations/builtin/core-nodes/n8n-nodes-base.readwritefile.md)):

```shell
mkdir local-files
```

The Docker Compose file below can automatically create this directory, but doing it manually ensures that it’s created with the right ownership and permissions.

6. Create Docker Compose file

Create a `compose.yaml` file. Paste the following in the file:

```yaml title=“compose.yaml file” services: traefik: image: “traefik” restart: always command: - “–api.insecure=true” - “–providers.docker=true” - “–providers.docker.exposedbydefault=false” - “–entrypoints.web.address=:80” - “–entrypoints.web.http.redirections.entryPoint.to=websecure” - “–entrypoints.web.http.redirections.entrypoint.scheme=https” - “–entrypoints.websecure.address=:443” - “–certificatesresolvers.mytlschallenge.acme.tlschallenge=true” - “–certificatesresolvers.mytlschallenge.acme.email=${SSL_EMAIL}” - “–certificatesresolvers.mytlschallenge.acme.storage=/letsencrypt/acme.json” ports: - “80:80” - “443:443” volumes: - traefik_data:/letsencrypt - /var/run/docker.sock:/var/run/docker.sock:ro

n8n: image: docker.n8n.io/n8nio/n8n restart: always ports: - “127.0.0.1:5678:5678” labels: - traefik.enable=true - traefik.http.routers.n8n.rule=Host(${SUBDOMAIN}.${DOMAIN_NAME}) - traefik.http.routers.n8n.tls=true - traefik.http.routers.n8n.entrypoints=web,websecure -
```
```

traefik.http.routers.n8n.tls.certresolver=mytlschallenge - traefik.http.middlewares.n8n.headers.SSLRedirect=true - traefik.http.middlewares.n8n.headers.STSSeconds=315360000 - traefik.http.middlewares.n8n.headers.browserXSSFilter=true - traefik.http.middlewares.n8n.headers.contentTypeNosniff=true - traefik.http.middlewares.n8n.headers.forceSTSHeader=true - traefik.http.middlewares.n8n.headers.SSLHost=$DOMAIN_NAME$ − *traefik.http.middlewares.n8n.headers.STSIncludeSubdomains* = *true* − *traefik.http.middlewares.n8n.headers.STSPreload* = *true* − *traefik.http.routers.n8n.middlewares* = *n8n@dockerenvironment* : − $N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS$ = *true* − $N8N_HOST$= {SUBDOMAIN}.$DOMAIN_NAME$ − $N8N_PORT$ = 5678 − $N8N_PROTOCOL$ = *https* − $N8N_RUNNERS_ENABLED$ = *true* − $NODE_ENV$ = *production* − $WEBHOOK_URL$ = *https* : //{SUBDOMAIN}.$DOMAIN_NAME$/ − $GENERIC_TIMEZONE$= {GENERIC_TIMEZONE} - TZ=${GENERIC_TIMEZONE} volumes: - n8n_data:/home/node/.n8n - ./local-files:/files

volumes: n8n_data: traefik_data:

```
The Docker Compose file above configures two containers: one for
n8n, and one to run [traefik](https://github.com/traefik/traefik),
an application proxy to manage TLS/SSL certificates and handle
routing.

It also creates and mounts two [Docker Volumes]
(https://docs.docker.com/engine/storage/volumes/) and mounts the
`local-files` directory you created earlier:

| Name            | Type
| Container mount   | Description
|
|-----------------|-----------------------------------------------
------------|------------------|--------------------------------
-------------------------------------------------------------------
-----------------------------|
| `n8n_data`      | [Volume]
(https://docs.docker.com/engine/storage/volumes/)   |
`/home/node/.n8n` | Where n8n saves its SQLite database file and
encryption key.
|
| `traefik_data`  | [Volume]
(https://docs.docker.com/engine/storage/volumes/)   | `/letsencrypt`
| Where traefik saves TLS/SSL certificate data.
|
| `./local-files` | [Bind]
(https://docs.docker.com/engine/storage/bind-mounts/) | `/files`
| A local directory shared between the n8n instance and host. In
n8n, use the `/files` path to read from and write to this directory.
|

## 7. Start Docker Compose

Start n8n by typing:

```shell
sudo docker compose up -d
```

To stop the containers, type:
```

```
sudo docker compose stop
```

8. Done

You can now reach n8n using the subdomain + domain combination you defined in your `.env` file configuration. The above example would result in `https://n8n.example.com`.

n8n is only accessible using secure HTTPS, not over plain HTTP.

If you have trouble reaching your instance, check your server's firewall settings and your DNS configuration.

Next steps

–8<– "_snippets/self-hosting/installation/server-setups-next-steps.md"

Update self-hosted n8n

It's important to keep your n8n version up to date. This ensures you get the latest features and fixes.

Some tips when updating:

–8<– "_snippets/manage-cloud/updating-best-practices.md"

For instructions on how to update, refer to the documentation for your installation method:

- Installed with npm
- Installed with Docker

Environment variables overview

This section lists the environment variables that you can use to change n8n's configuration settings when self-hosting n8n.

For the complete and most up-to-date list of all environment variables, see the full reference in the n8n docs repository: Full environment variables index.

AI Assistant environment variables

Variable	Type	Default	Description
`N8N_AI_ASSISTANT_BASE_URL`	String	(empty)	Base URL of the AI assistant service, specified as `https://ai-assistant.n8n.io`.

	Required if you self-host n8n and want to enable the AI Assistant.

Binary data environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

By default, n8n uses memory to store binary data. Enterprise users can choose to use an external service instead. Refer to External storage for more information on using external storage for binary data.

Variable	Type	Default
`N8N_AVAILABLE_BINARY_DATA_MODES`	String	`filesystem`
`N8N_BINARY_DATA_STORAGE_PATH`	String	`N8N_USER_FOLDER/binaryData`
`N8N_DEFAULT_BINARY_DATA_MODE`	String	`default`

Credentials environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

Enable credential overwrites using the following environment variables. Refer to Credential overwrites for details.

Variable	Type	Default	Descripti
`CREDENTIALS_OVERWRITE_DATA_FILE`	*	-	Overwrites credentials
`CREDENTIALS_OVERWRITE_ENDPOINT`	String	-	The API en to fetch credentials
`CREDENTIALS_OVERWRITE_PERSISTENCE`	Boolean	`false`	Enable dat persistenc credential overwrites Required f multiinstar queue moc propagate overwrites workers th a publish/su approach.
`CREDENTIALS_DEFAULT_NAME`	String	`My credentials`	The defaul for creden

Database environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

By default, n8n uses SQLite. n8n also supports PostgreSQL. n8n deprecated support for MySQL and MariaDB in v1.0.

This page outlines environment variables to configure your chosen database for your self-hosted n8n instance.

Variable	Type	Default	Description
`DB_TYPE_FILE`	Enum string: `sqlite`,	`sqlite`	The database to use.

	`postgresdb`		
`DB_TABLE_PREFIX`	*	-	Prefix to use for table names.
`DB_PING_INTERVAL_SECONDS`	Number	2	The interval, in seconds, between pings to the database to check if the connection is still alive.

PostgreSQL

Variable	Type	Default	Descrip
`DB_POSTGRESDB_DATABASE` `/_FILE`	String	`n8n`	The nam Postgre
`DB_POSTGRESDB_HOST` `/_FILE`	String	`localhost`	The Post
`DB_POSTGRESDB_PORT` `/_FILE`	Number	`5432`	The Post
`DB_POSTGRESDB_USER` `/_FILE`	String	`postgres`	The Post
`DB_POSTGRESDB_PASSWORD` `/_FILE`	String	-	The Post passwor
`DB_POSTGRESDB_POOL_SIZE` `/_FILE`	Number	2	Control parallel connecti should h Increasi with res utilizatio many co degrade
`DB_POSTGRESDB_CONNECTION_TIMEOUT` `/_FILE`	Number	`20000`	Postgres timeout
`DB_POSTGRESDB_IDLE_CONNECTION_TIMEOUT` `/_FILE`	Number	`30000`	Amount an idle c eligible for being
`DB_POSTGRESDB_SCHEMA` `/_FILE`	String	`public`	The Post schema.
`DB_POSTGRESDB_SSL_ENABLED` `/_FILE`	Boolean	`false`	Whether SSL. Aut enabled `DB_POSTG` `DB_POSTG` or `DB_POSTG` is define
`DB_POSTGRESDB_SSL_CA` `/_FILE`	String	-	The Post certifica
`DB_POSTGRESDB_SSL_CERT` `/_FILE`	String	-	The Post certifica
`DB_POSTGRESDB_SSL_KEY`	String	-	The Post

`/_FILE`			key.
`DB_POSTGRESDB_SSL_REJECT_UNAUTHORIZED_FILE`	Boolean	`true`	If n8n sh unautho connecti not (fals

SQLite

Variable	Type	Default	Description
`DB_SQLITE_POOL_SIZE`	Number	`0`	Controls whether to open the SQLite file in WAL mode or rollback journal mode. Uses rollback journal mode when set to zero. When greater than zero, uses WAL mode with the value determining the number of parallel SQL read connections to configure. WAL mode is much more performant and reliable than the rollback journal mode.
`DB_SQLITE_VACUUM_ON_STARTUP`	Boolean	`false`	Runs VACUUM operation on startup to rebuild the database. Reduces file size and optimizes indexes. This is a long running blocking operation and increases start-up time.

Deployment environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

This page lists the deployment configuration options for your self-hosted n8n instance, including setting up access URLs, enabling templates, customizing encryption, and configuring server details.

Variable	Type	Default
`HTTP_PROXY`	String	-
`HTTPS_PROXY`	String	-
`ALL_PROXY`	String	-
`NO_PROXY`	String	-

`N8N_EDITOR_BASE_URL`	String	-
`N8N_CONFIG_FILES` (deprecated)	String	-
`N8N_DISABLE_UI`	Boolean	`false`
`N8N_PREVIEW_MODE`	Boolean	`false`
`N8N_TEMPLATES_ENABLED`	Boolean	`false`
`N8N_TEMPLATES_HOST`	String	`https://api.n8n.io`

`N8N_ENCRYPTION_KEY`	String	Random key generated
`N8N_USER_FOLDER`	String	`user-folder`
`N8N_PATH`	String	`/`
`N8N_HOST`	String	`localhost`
`N8N_PORT`	Number	`5678`
`N8N_LISTEN_ADDRESS`	String	`::`
`N8N_PROTOCOL`	Enum string: `http`, `https`	`http`
`N8N_SSL_KEY`	String	-
`N8N_SSL_CERT`	String	-
`N8N_PERSONALIZATION_ENABLED`	Boolean	`true`
`N8N_VERSION_NOTIFICATIONS_ENABLED`	Boolean	`true`

N8N_VERSION_NOTIFICATIONS_ENDPOINT	String	https://api.n8n.io/ver
N8N_VERSION_NOTIFICATIONS_INFO_URL	String	https://docs.n8n.io/ge started/installation/u
N8N_DIAGNOSTICS_ENABLED	Boolean	true
N8N_DIAGNOSTICS_CONFIG_FRONTEND	String	1zPn9bgWPzlQc0p8Gj1uiK
N8N_DIAGNOSTICS_CONFIG_BACKEND	String	1zPn7YoGC3ZXE9zLeTKLuQ
N8N_PUSH_BACKEND	String	websocket
VUE_APP_URL_BASE_API	String	http://localhost:5678/
N8N_HIRING_BANNER_ENABLED	Boolean	true

`N8N_PUBLIC_API_SWAGGERUI_DISABLED`	Boolean	`false`
`N8N_PUBLIC_API_DISABLED`	Boolean	`false`
`N8N_PUBLIC_API_ENDPOINT`	String	`api`
`N8N_GRACEFUL_SHUTDOWN_TIMEOUT`	Number	`30`
`N8N_DEV_RELOAD`	Boolean	`false`
`N8N_REINSTALL_MISSING_PACKAGES`	Boolean	`false`
`N8N_TUNNEL_SUBDOMAIN`	String	-
`N8N_PROXY_HOPS`	Number	0

Endpoints environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

This page lists environment variables for customizing endpoints in n8n.

Variable	Type	Default	I
N8N_PAYLOAD_SIZE_MAX	Number	16	T n p i
N8N_FORMDATA_FILE_SIZE_MAX	Number	200	N s i v p N
N8N_METRICS	Boolean	false	W e / e
N8N_METRICS_PREFIX	String	n8n_	C p n n n
N8N_METRICS_INCLUDE_DEFAULT_METRICS	Boolean	true	W e d s n n
N8N_METRICS_INCLUDE_CACHE_METRICS	Boolean	false	W i n (c a c i t
N8N_METRICS_INCLUDE_MESSAGE_EVENT_BUS_METRICS	Boolean	false	W i n (e n t
N8N_METRICS_INCLUDE_WORKFLOW_ID_LABEL	Boolean	false	W i l v c n
			W i

N8N_METRICS_INCLUDE_NODE_TYPE_LABEL	Boolean	false	l n c n
N8N_METRICS_INCLUDE_CREDENTIAL_TYPE_LABEL	Boolean	false	W i l c t c n
N8N_METRICS_INCLUDE_API_ENDPOINTS	Boolean	false	W e n A e
N8N_METRICS_INCLUDE_API_PATH_LABEL	Boolean	false	W i l p i
N8N_METRICS_INCLUDE_API_METHOD_LABEL	Boolean	false	W i l H n (. i
N8N_METRICS_INCLUDE_API_STATUS_CODE_LABEL	Boolean	false	W i l H c 4 A i
N8N_METRICS_INCLUDE_QUEUE_METRICS	Boolean	false	W i n j s n
N8N_METRICS_QUEUE_METRICS_INTERVAL	Integer	20	H (t q n
N8N_ENDPOINT_REST	String	rest	T u H e
N8N_ENDPOINT_WEBHOOK	String	webhook	T u v e

`N8N_ENDPOINT_WEBHOOK_TEST`	String	`webhook-test`	T u t v e
`N8N_ENDPOINT_WEBHOOK_WAIT`	String	`webhook-waiting`	T u v v e
`WEBHOOK_URL`	String	-	U n p W U r h r p h n d
`N8N_DISABLE_PRODUCTION_MAIN_PROCESS`	Boolean	`false`	D p v f p T e H l p v v s p

Executions environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

This page lists environment variables to configure workflow execution settings.

Variable	Type	Defaul
`EXECUTIONS_MODE`	Enum string: `regular`, `queue`	`regular`

EXECUTIONS_TIMEOUT	Number	-1
EXECUTIONS_TIMEOUT_MAX	Number	3600
EXECUTIONS_DATA_SAVE_ON_ERROR	Enum string: all, none	all
EXECUTIONS_DATA_SAVE_ON_SUCCESS	Enum string: all, none	all
EXECUTIONS_DATA_SAVE_ON_PROGRESS	Boolean	false
EXECUTIONS_DATA_SAVE_MANUAL_EXECUTIONS	Boolean	true
EXECUTIONS_DATA_PRUNE	Boolean	true
EXECUTIONS_DATA_MAX_AGE	Number	336
EXECUTIONS_DATA_PRUNE_MAX_COUNT	Number	10000
EXECUTIONS_DATA_HARD_DELETE_BUFFER	Number	1
EXECUTIONS_DATA_PRUNE_HARD_DELETE_INTERVAL	Number	15
EXECUTIONS_DATA_PRUNE_SOFT_DELETE_INTERVAL	Number	60
N8N_CONCURRENCY_PRODUCTION_LIMIT	Number	-1

N8N_CONCURRENCY_PRODUCTION_LIMIT	Number	-1
N8N_WORKFLOW_AUTODEACTIVATION_ENABLED	Boolean	false
N8N_WORKFLOW_AUTODEACTIVATION_MAX_LAST_EXECUTIONS	Number	3

External data storage environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

Refer to External storage for more information on using external storage for binary data.

Variable	Type	Default	Descr
N8N_EXTERNAL_STORAGE_S3_HOST	String	-	Host o bucket compa extern storag examp s3.us- 1.amaz
N8N_EXTERNAL_STORAGE_S3_BUCKET_NAME	String	-	Name n8n bu S3-cor extern storag
N8N_EXTERNAL_STORAGE_S3_BUCKET_REGION	String	-	Regior n8n bu S3-cor extern storag examp east-1
N8N_EXTERNAL_STORAGE_S3_ACCESS_KEY	String	-	Access S3-cor extern storag
N8N_EXTERNAL_STORAGE_S3_ACCESS_SECRET	String	-	Access in S3- compa extern storag
			Use au creder

`N8N_EXTERNAL_STORAGE_S3_AUTH_AUTO_DETECT`	Boolean	-	detect auther S3 cal extern storag will ig access access and us defaul creder provid

External hooks environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

You can define external hooks that n8n executes whenever a specific operation runs. Refer to Backend hooks for examples of available hooks and Hook files for information on file formatting.

Variable	Type	Description
`EXTERNAL_HOOK_FILES`	String	Files containing backend external hooks. Provide multiple files as a colon-separated list (“:”).
`EXTERNAL_FRONTEND_HOOKS_URLS`	String	URLs to files containing frontend external hooks. Provide multiple URLs as a colon-separated list (“:”).

External secrets environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

You can use an external secrets store to manage credentials for n8n. Refer to External secrets for details.

Variable	Type	Default	Descripti
`N8N_EXTERNAL_SECRETS_UPDATE_INTERVAL`	Number	`300` (5 minutes)	How ofter (in second to check f secret updates.

Insights environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

Insights gives instance owners and admins visibility into how workflows perform over time. Refer to Insights for details.

Variable	Type	D
N8N_DISABLED_MODULES	String	-
N8N_INSIGHTS_COMPACTION_BATCH_SIZE	Number	5(
N8N_INSIGHTS_COMPACTION_DAILY_TO_WEEKLY_THRESHOLD_DAYS	Number	1{
N8N_INSIGHTS_COMPACTION_HOURLY_TO_DAILY_THRESHOLD_DAYS	Number	9(
N8N_INSIGHTS_COMPACTION_INTERVAL_MINUTES	Number	6(
N8N_INSIGHTS_FLUSH_BATCH_SIZE	Number	1(
N8N_INSIGHTS_FLUSH_INTERVAL_SECONDS	Number	3(

Logs environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

This page lists environment variables to set up logging for debugging. Refer to Logging in n8n for details.

n8n logs

Variable	Type	Default	Descri
`N8N_LOG_LEVEL`	Enum string: `info`, `warn`, `error`, `debug`	`info`	Log ou Refer t for det
`N8N_LOG_OUTPUT`	Enum string: `console`, `file`	`console`	Where logs. Pr multipl comma list.
`N8N_LOG_FORMAT`	Enum string: `text`, `json`	`text`	The log use. `te` human messag prints o object contain messag timesta metada useful f produc monito as debu
`N8N_LOG_CRON_ACTIVE_INTERVAL`	Number	`0`	Interva to log c active c Set to `0`
`N8N_LOG_FILE_COUNT_MAX`	Number	`100`	Max nu files to
`N8N_LOG_FILE_SIZE_MAX`	Number	`16`	Max siz log file
`N8N_LOG_FILE_LOCATION`	String	`<n8n-directory-path>/logs/n8n.log`	Log file Require N8N_L set to f
`DB_LOGGING_ENABLED`	Boolean	`false`	Whethe databas logging

`DB_LOGGING_OPTIONS`	Enum string: `query`, `error`, `schema`, `warn`, `info`, `log`	`error`	Databa output enable specify to Type logging
`DB_LOGGING_MAX_EXECUTION_TIME`	Number	`1000`	Maximu executi millisec before warnin disable running warnin
`CODE_ENABLE_STDOUT`	Boolean	`false`	Set to t Code n from `co` `print` t process only fo executi
`NO_COLOR`	any	`undefined`	Set to a output ANSI c more in see the website

Log streaming

Refer to Log streaming for more information on this feature.

Variable	Type	Default	Desc
`N8N_EVENTBUS_CHECKUNSENTINTERVAL`	Number	`0`	How millis to ch unse mess Can i cases mess twice to dis
`N8N_EVENTBUS_LOGWRITER_SYNCFILEACCESS`	Boolean	`false`	Whet file a happ synch withi threa or no
`N8N_EVENTBUS_LOGWRITER_KEEPLOGCOUNT`	Number	`3`	Num even to ke

`N8N_EVENTBUS_LOGWRITER_MAXFILESIZEINKB`	Number	`10240`	Maxi size (bytes event befor one s
`N8N_EVENTBUS_LOGWRITER_LOGBASENAME`	String	`n8nEventLog`	Base the e file.

License environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

To enable certain licensed features, you must first activate your license. You can do this either through the UI or by setting environment variables. For more information, see license key.

Variable	Type	Default
`N8N_HIDE_USAGE_PAGE`	boolean	`false`
`N8N_LICENSE_ACTIVATION_KEY`	String	`''`
`N8N_LICENSE_AUTO_RENEW_ENABLED`	Boolean	`true`

N8N_LICENSE_DETACH_FLOATING_ON_SHUTDOWN	Boolean	true
N8N_LICENSE_SERVER_URL	String	https://license.n
N8N_LICENSE_TENANT_ID	Number	1
https_proxy_license_server	String	https://user:pass

Nodes environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

This page lists the environment variables configuration options for managing nodes in n8n, including specifying which nodes to load or exclude, importing built-in or external modules in the Code node, and enabling community nodes.

Variable	Type	Default
N8N_COMMUNITY_PACKAGES_ENABLED	Boolean	true
N8N_COMMUNITY_PACKAGES_PREVENT_LOADING	Boolean	false
N8N_COMMUNITY_PACKAGES_REGISTRY	String	https://registry.n
N8N_CUSTOM_EXTENSIONS	String	-
N8N_PYTHON_ENABLED	Boolean	true
N8N_UNVERIFIED_PACKAGES_ENABLED	Boolean	true
N8N_VERIFIED_PACKAGES_ENABLED	Boolean	true
NODE_FUNCTION_ALLOW_BUILTIN	String	-
NODE_FUNCTION_ALLOW_EXTERNAL	String	-
NODES_ERROR_TRIGGER_TYPE	String	n8n-nodes-base.errorTrigger

NODES_EXCLUDE	Array of strings	-
NODES_INCLUDE	Array of strings	-

Queue mode environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

You can run n8n in different modes depending on your needs. Queue mode provides the best scalability. Refer to Queue mode for more information.

Variable	Type	Default	Descript
OFFLOAD_MANUAL_EXECUTIONS_TO_WORKERS	Boolean	false	Set to tru manual e the work main.
QUEUE_BULL_PREFIX	String	-	Prefix to keys.
QUEUE_BULL_REDIS_DB	Number	0	The Redi
QUEUE_BULL_REDIS_HOST	String	localhost	The Redi
QUEUE_BULL_REDIS_PORT	Number	6379	The Redi
QUEUE_BULL_REDIS_USERNAME	String	-	The Redi Redis ver Don’t def compatib
QUEUE_BULL_REDIS_PASSWORD	String	-	The Redi
QUEUE_BULL_REDIS_TIMEOUT_THRESHOLD	Number	10000	The Redi (in ms).
QUEUE_BULL_REDIS_CLUSTER_NODES	String	-	Expects a list of Re the forma Redis clie connect t queue mo = queue), will creat client ins client, an QUEUE_BUL QUEUE_BUL
QUEUE_BULL_REDIS_TLS	Boolean	false	Enable T connectio
QUEUE BULL REDIS DUALSTACK	Boolean	false	Enable du (IPv4 and

			connectic
			Depreca N8N_GRACE instead.
QUEUE_WORKER_TIMEOUT (**deprecated**)	Number	30	How long (seconds) execution worker p
QUEUE_HEALTH_CHECK_ACTIVE	Boolean	false	Whether checks (t (false).
QUEUE_HEALTH_CHECK_PORT	Number	5678	The port checks o a port co starting a using its this.
QUEUE_WORKER_LOCK_DURATION	Number	60000	How long period fo on a mes
QUEUE_WORKER_LOCK_RENEW_TIME	Number	10000	How freq should a lease tim
QUEUE_WORKER_STALLED_INTERVAL	Number	30000	How ofte check for for never
QUEUE_WORKER_MAX_STALLED_COUNT	Number	1	Maximum stalled jo processe

Multi-main setup

Refer to Configuring multi-main setup for details.

Variable	Type	Default	Descriptio
N8N_MULTI_MAIN_SETUP_ENABLED	Boolean	false	Whether to enable multi-main setup for queue mod (license required).
N8N_MULTI_MAIN_SETUP_KEY_TTL	Number	10	Time to live (in seconds for leader key in mult main setup.
N8N_MULTI_MAIN_SETUP_CHECK_INTERVAL	Number	3	Interval (in seconds) fo leader chec in multi-main setup.

Security environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

Variable	Type	Default	Des
`N8N_BLOCK_ENV_ACCESS_IN_NODE`	Boolean	`false`	Whe acce vari and or n
`N8N_BLOCK_FILE_ACCESS_TO_N8N_FILES`	Boolean	`true`	Set acce `.n8n` defi files
`N8N_ENFORCE_SETTINGS_FILE_PERMISSIONS`	Boolean	`false`	Set 0600 setti the acce
`N8N_RESTRICT_FILE_ACCESS_TO`	String		Limi thes mult sem (“;”
`N8N_SECURITY_AUDIT_DAYS_ABANDONED_WORKFLOW`	Number	90	Num cons aba exec
`N8N_CONTENT_SECURITY_POLICY`	String	`{}`	Set Poli helm dire exa ance `["ht` }
`N8N_SECURE_COOKIE`	Boolean	`true`	Ens only enh
`N8N_SAMESITE_COOKIE`	Enum string: `strict`, `lax`, `none`	`lax`	Con beh
`N8N_GIT_NODE_DISABLE_BARE_REPOS`	Boolean	`false`	Set Git with enh
`N8N_GIT_NODE_ENABLE_HOOKS`	Boolean	`false`	Set Git hoo

Source control environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

n8n uses Git-based source control to support environments. Refer to Source control and environments for more information on how to link a Git repository to an n8n instance and configure your source control.

Variable	Type	Default	Descripti
`N8N_SOURCECONTROL_DEFAULT_SSH_KEY_TYPE`	String	`ed25519`	Set to `rsa` make RSA the defaul SSH key type for Source control setup.

Task runner environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

Task runners execute code defined by the Code node.

n8n instance environment variables

Variable	Type	Default	Descriptio
`N8N_RUNNERS_ENABLED`	Boolean	`false`	Are task runners enabled.
`N8N_RUNNERS_MODE`	Enum string: `internal`, `external`	`internal`	How to laur and run the task runner `internal` means n8n launch a tas runner as c process. `external` means an external orchestrato will launch task runner
`N8N_RUNNERS_AUTH_TOKEN`	String	Random string	Shared secr used by a ta runner to authenticat n8n. Requir when using `external` mo

`N8N_RUNNERS_BROKER_PORT`	Number	`5679`	Port the tas broker liste on for task runner connections
`N8N_RUNNERS_BROKER_LISTEN_ADDRESS`	String	`127.0.0.1`	Address the task broker listens on.
`N8N_RUNNERS_MAX_PAYLOAD`	Number	`1 073 741 824`	Maximum payload size bytes for communica between a t broker and task runner
`N8N_RUNNERS_MAX_OLD_SPACE_SIZE`	String		The `--max-o space-size` option to us for a task runner (in MB). By default, Node.js will set this bas on available memory.
`N8N_RUNNERS_MAX_CONCURRENCY`	Number	`5`	The numbe concurrent tasks a task runner can execute at a time.
`N8N_RUNNERS_TASK_TIMEOUT`	Number	`300`	The maximu time, in seconds, a t can run bef the runner stops it and restarts. Th value must greater tha
`N8N_RUNNERS_HEARTBEAT_INTERVAL`	Number	`30`	The interva seconds, at which the runner mus send a heartbeat t the broker. the runner doesn't sen heartbeat i time, the ta stops and t runner restarts. Th value must greater tha
			Whether to

`N8N_RUNNERS_INSECURE_MODE`	Boolean	`false`	Whether to disable all security measures in the task runner, for compatibility with module that rely on insecure JS features. **Discourage for production use.**
`N8N_RUNNERS_TASK_REQUEST_TIMEOUT`	Number	`20`	How long (in seconds) a task request can wait for a runner to become available before timing out. This prevents workflows from hanging indefinitely when no runners are available. Must be greater than 0.

Task runner launcher environment variables

Variable	Type	Default
`N8N_RUNNERS_LAUNCHER_LOG_LEVEL`	Enum string: `debug`, `info`, `warn`, `error`	`info`
`N8N_RUNNERS_AUTH_TOKEN`	String	-
`N8N_RUNNERS_AUTO_SHUTDOWN_TIMEOUT`	Number	`15`
`N8N_RUNNERS_TASK_BROKER_URI`	String	`http://127.0.0.1:5`

`N8N_RUNNERS_LAUNCHER_HEALTH_CHECK_PORT`	Number	`5680`
`N8N_RUNNERS_MAX_PAYLOAD`	Number	`1 073 741 824`
`N8N_RUNNERS_MAX_CONCURRENCY`	Number	`5`

Task runner environment variables (all languages)

Variable	Type	Default
`N8N_RUNNERS_GRANT_TOKEN`	String	Random string
`N8N_RUNNERS_AUTO_SHUTDOWN_TIMEOUT`	Number	`15`
`N8N_RUNNERS_TASK_BROKER_URI`	String	`http://127.0.0.1:5`
`N8N_RUNNERS_LAUNCHER_HEALTH_CHECK_PORT`	Number	`5680`
`N8N_RUNNERS_MAX_PAYLOAD`	Number	`1 073 741 824`

N8N_RUNNERS_MAX_CONCURRENCY	Number	5	

Task runner environment variables (JavaScript)

Variable	Type	Default	De
NODE_FUNCTION_ALLOW_BUILTIN	String	-	Pe im bu in nc all dis im mo de
NODE_FUNCTION_ALLOW_EXTERNAL	String	-	Pe im ex (fr n8 in nc dis im mo de
N8N_RUNNERS_ALLOW_PROTOTYPE_MUTATION	Boolean	false	W pr mu ex lib tr mo re pr mu ex pu co se
GENERIC_TIMEZONE	*	America/New_York	Th tir co th
NODE_OPTIONS	String	-	Op No
			Th sp to ru

`N8N_RUNNERS_MAX_OLD_SPACE_SIZE`	String	By No th av m

Task runner environment variables (Python)

Variable	Type	Default
`N8N_RUNNERS_STDLIB_ALLOW`	String	-
`N8N_RUNNERS_EXTERNAL_ALLOW`	String	-

`N8N_RUNNERS_BUILTINS_DENY`	String	`eval,exec,compile,open,input,`
`N8N_BLOCK_RUNNER_ENV_ACCESS`	Boolean	`true`

Timezone and localization environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

Variable	Type	Default	Description
`GENERIC_TIMEZONE`	*	`America/New_York`	The n8n instance timezone. Important for schedule nodes (such as Cron).
			A locale identifier, compatible with the Accept-Language header. n8n doesn’t support regional identifiers, such as

`N8N_DEFAULT_LOCALE`	String	`en`	`de-AT`. When running in a locale other than the default, n8n displays UI strings in the selected locale, and falls back to `en` for any untranslated strings.

User management SMTP, and two-factor authentication environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

Refer to User management for more information on setting up user management and emails.

Variable	Type	Default	D
`N8N_EMAIL_MODE`	String	`smtp`	E
`N8N_SMTP_HOST`	String	-	y
`N8N_SMTP_PORT`	Number	-	y
`N8N_SMTP_USER`	String	-	y
`N8N_SMTP_PASS`	String	-	y
`N8N_SMTP_OAUTH_SERVICE_CLIENT`	String	-	I is
`N8N_SMTP_OAUTH_PRIVATE_KEY`	String	-	I is
`N8N_SMTP_SENDER`	String	-	S o E <
`N8N_SMTP_SSL`	Boolean	`true`	V n
`N8N_SMTP_STARTTLS`	Boolean	`true`	V (
`N8N_UM_EMAIL_TEMPLATES_INVITE`	String	-	F T in
`N8N_UM_EMAIL_TEMPLATES_PWRESET`	String	-	F T p
`N8N_UM_EMAIL_TEMPLATES_WORKFLOW_SHARED`	String	-	C f s t
`N8N_UM_EMAIL_TEMPLATES_CREDENTIALS_SHARED`	String	-	C f w t

N8N_UM_EMAIL_TEMPLATES_PROJECT_SHARED	String	-	C f s t
N8N_USER_MANAGEMENT_JWT_SECRET	String	-	S n
N8N_USER_MANAGEMENT_JWT_DURATION_HOURS	Number	168	S h
N8N_USER_MANAGEMENT_JWT_REFRESH_TIMEOUT_HOURS	Number	0	H e n N - f p N
N8N_MFA_ENABLED	Boolean	`true`	V a n 2
N8N_INVITE_LINKS_EMAIL_ONLY	Boolean	`false`	V i e o i p u

Workflows environment variables

–8<– “_snippets/self-hosting/file-based-configuration.md”

Variable	Type	Default
N8N_ONBOARDING_FLOW_DISABLED	Boolean	`false`
N8N_WORKFLOW_ACTIVATION_BATCH_SIZE	Number	`1`
N8N_WORKFLOW_CALLER_POLICY_DEFAULT_OPTION	String	`workflowsFromSa`
N8N_WORKFLOW_TAGS_DISABLED	Boolean	`false`

`WORKFLOWS_DEFAULT_NAME`	String	`My workflow`

Configuration

You can change n8n's settings using environment variables. For a full list of available configurations see Environment Variables.

Set environment variables by command line

npm

For npm, set your desired environment variables in terminal. The command depends on your command line.

Bash CLIs:

```
export <variable>=<value>
```

In cmd.exe:

```
set <variable>=<value>
```

In PowerShell:

```
$env:<variable>=<value>
```

Docker

In Docker you can use the `-e` flag from the command line:

```
docker run -it --rm \
 --name n8n \
 -p 5678:5678 \
 -e N8N_TEMPLATES_ENABLED="false" \
 docker.n8n.io/n8nio/n8n
```

Docker Compose file

In Docker, you can set your environment variables in the `n8n:` `environment:` element of your `docker-compose.yaml` file.

For example:

```
n8n:
    environment:
      - N8N_TEMPLATES_ENABLED=false
```

Keeping sensitive data in separate files

You can append `_FILE` to individual environment variables to provide their configuration in a separate file, enabling you to avoid passing sensitive details using environment variables. n8n loads the data from the file with the given name, making it possible to load data from Docker-Secrets and Kubernetes-Secrets.

Refer to Environment variables for details on each variable.

While most environment variables can use the `_FILE` suffix, it's more beneficial for sensitive data such as credentials and database configuration. Here are some examples:

```
        CREDENTIALS_OVERWRITE_DATA_FILE=/path/to/credentials_data
        DB_TYPE_FILE=/path/to/db_type
        DB_POSTGRESDB_DATABASE_FILE=/path/to/database_name
        DB_POSTGRESDB_HOST_FILE=/path/to/database_host
        DB_POSTGRESDB_PORT_FILE=/path/to/database_port
        DB_POSTGRESDB_USER_FILE=/path/to/database_user
        DB_POSTGRESDB_PASSWORD_FILE=/path/to/database_password
        DB_POSTGRESDB_SCHEMA_FILE=/path/to/database_schema
        DB_POSTGRESDB_SSL_CA_FILE=/path/to/ssl_ca
        DB_POSTGRESDB_SSL_CERT_FILE=/path/to/ssl_cert
        DB_POSTGRESDB_SSL_KEY_FILE=/path/to/ssl_key
        DB_POSTGRESDB_SSL_REJECT_UNAUTHORIZED_FILE=/path/to/ssl_reject_unaut
h
```

Configuration examples

This section contains examples for how to configure n8n to solve particular use cases.

Isolate n8n

By default, a self-hosted n8n instance sends data to n8n's servers. It notifies users about available updates, workflow templates, and diagnostics.

To prevent your n8n instance from connecting to n8n's servers, set these environment variables to false:

```
N8N_DIAGNOSTICS_ENABLED=false
N8N_VERSION_NOTIFICATIONS_ENABLED=false
N8N_TEMPLATES_ENABLED=false
```

Unset n8n's diagnostics configuration:

```
EXTERNAL_FRONTEND_HOOKS_URLS=
N8N_DIAGNOSTICS_CONFIG_FRONTEND=
N8N_DIAGNOSTICS_CONFIG_BACKEND=
```

Refer to Environment variables reference for more information on these variables.

Configure the Base URL for n8n's front end access

You can configure the Base URL that the front end uses to connect to the back end's REST API. This is relevant when you want to host n8n's front end and back end separately.

```
export VUE_APP_URL_BASE_API=https://n8n.example.com/
```

Refer to Environment variables reference for more information on this variable.

Configure n8n to use your own certificate authority or self-signed certificate

You can add your own certificate authority (CA) or self-signed certificate to n8n. This means you are able to trust a certain SSL certificate instead of trusting all invalid certificates, which is a potential security risk.

To use this feature you need to place your certificates in a folder and mount the folder to `/opt/custom-certificates` in the container. The external path that you map to `/opt/custom-certificates` must be writable by the container.

Docker

The examples below assume you have a folder called `pki` that contains your certificates in either the directory you run the command from or next to your docker compose file.

Docker CLI

When using the CLI you can use the `-v` flag from the command line:

```
docker run -it --rm \
 --name n8n \
 -p 5678:5678 \
 -v ./pki:/opt/custom-certificates \
 docker.n8n.io/n8nio/n8n
```

Docker Compose

```
name: n8n
services:
    n8n:
        volumes:
            - ./pki:/opt/custom-certificates
        container_name: n8n
        ports:
            - 5678:5678
```

```
image: docker.n8n.io/n8nio/n8n
```

You should also give the right permissions to the imported certs. You can do this once the container is running (assuming n8n as the container name):

```
docker exec --user 0 n8n chown -R 1000:1000 /opt/custom-certificates
```

Certificate requirements for Custom Trust Store

Supported certificate types:

- Root CA Certificates: these are certificates from Certificate Authorities that sign other certificates. Trust these to accept all certificates signed by that CA.
- Self-Signed Certificates: certificates that servers create and sign themselves. Trust these to accept connections to that specific server only.

You must use PEM format:

- Text-based format with BEGIN/END markers
- Supported file extensions: `.pem`, `.crt`, `.cer`
- Contains the public certificate (no private key needed)

For example:

```
-----BEGIN CERTIFICATE-----
MIIDXTCCAkWgAwIBAgIJAKoK/heBjcOuMA0GCSqGSIb3DQEBBQUAMEUxCzAJBgNV
[base64 encoded data]
-----END CERTIFICATE-----
```

The system doesn't accept:

- DER/binary format files
- PKCS#7 (.p7b) files
- PKCS#12 (.pfx, .p12) files
- Private key files
- Convert these formats to PEM before use.

Set a custom encryption key

n8n creates a random encryption key automatically on the first launch and saves it in the `~/.n8n` folder. n8n uses that key to encrypt the credentials before they get saved to the database. If the key isn't yet in the settings file, you can set it using an environment variable, so that n8n uses your custom key instead of generating a new one.

In queue mode, you must specify the encryption key environment variable for all workers.

```
export N8N_ENCRYPTION_KEY=<SOME RANDOM STRING>
```

Refer to Environment variables reference for more information on this variable.

Configure workflow timeout settings

A workflow times out and gets canceled after this time (in seconds). If the workflow runs in the main process, a soft timeout happens (takes effect after the current node finishes). If a workflow runs in its own process, n8n attempts a soft timeout first, then kills the process after waiting for a fifth of the given timeout duration.

`EXECUTIONS_TIMEOUT` default is `-1`. For example, if you want to set the timeout to one hour:

```
export EXECUTIONS_TIMEOUT=3600
```

You can also set maximum execution time (in seconds) for each workflow individually. For example, if you want to set maximum execution time to two hours:

```
export EXECUTIONS_TIMEOUT_MAX=7200
```

Refer to Environment variables reference for more information on these variables.

Specify location for your custom nodes

Every user can add custom nodes that get loaded by n8n on startup. The default location is in the subfolder `.n8n/custom` of the user who started n8n.

You can define more folders with an environment variable:

```
export N8N_CUSTOM_EXTENSIONS="/home/jim/n8n/custom-nodes;/data/n8n/nodes"
```

Refer to Environment variables reference for more information on this variable.

Enable modules in Code node

For security reasons, the Code node restricts importing modules. It's possible to lift that restriction for built-in and external modules by setting the following environment variables:

- `NODE_FUNCTION_ALLOW_BUILTIN`: For built-in modules
- `NODE_FUNCTION_ALLOW_EXTERNAL`: For external modules sourced from n8n/node_modules directory. External module support is disabled when an environment variable isn't set.

```
# Allows usage of all builtin modules
export NODE_FUNCTION_ALLOW_BUILTIN=*

# Allows usage of only crypto
export NODE_FUNCTION_ALLOW_BUILTIN=crypto
```

```
# Allows usage of only crypto and fs
export NODE_FUNCTION_ALLOW_BUILTIN=crypto,fs

# Allow usage of external npm modules.
export NODE_FUNCTION_ALLOW_EXTERNAL=moment,lodash
```

Refer to Environment variables reference for more information on these variables.

Set the self-hosted instance timezone

The default timezone is America/New_York. For instance, the Schedule node uses it to know at what time the workflow should start. To set a different default timezone, set `GENERIC_TIMEZONE` to the appropriate value. For example, if you want to set the timezone to Berlin (Germany):

```
export GENERIC_TIMEZONE=Europe/Berlin
```

You can find the name of your timezone here.

Refer to Environment variables reference for more information on this variable.

Specify user folder path

n8n saves user-specific data like the encryption key, SQLite database file, and the ID of the tunnel (if used) in the subfolder `.n8n` of the user who started n8n. It's possible to overwrite the user-folder using an environment variable.

```
export N8N_USER_FOLDER=/home/jim/n8n
```

Refer to Environment variables reference for more information on this variable.

Configure n8n webhooks with reverse proxy

n8n creates the webhook URL by combining `N8N_PROTOCOL`, `N8N_HOST` and `N8N_PORT`. If n8n runs behind a reverse proxy, that won't work. That's because n8n runs internally on port 5678 but the reverse proxy exposes it to the web on port 443.

When running n8n behind a reverse proxy, it's important to do the following:

- set the webhook URL manually with the `WEBHOOK_URL` environment variable so that n8n can display it in the editor UI and register the

correct webhook URLs with external services.
- Set the `N8N_PROXY_HOPS` environment variable to `1`.
- On the last proxy on the request path, set the following headers to pass on information about the initial request:
  - `X-Forwarded-For`
  - `X-Forwarded-Host`
  - `X-Forwarded-Proto`

```
export WEBHOOK_URL=https://n8n.example.com/
export N8N_PROXY_HOPS=1
```

Refer to Environment variables reference for more information on this variable.

Enable Prometheus metrics

To collect and expose metrics, n8n uses the prom-client library.

The `/metrics` endpoint is disabled by default, but it's possible to enable it using the `N8N_METRICS` environment variable.

```
export N8N_METRICS=true
```

Refer to the respective Environment Variables (`N8N_METRICS_INCLUDE_*`) for configuring which metrics and labels should get exposed.

Both `main` and `worker` instances are able to expose metrics.

Queue metrics

To enable queue metrics, set the `N8N_METRICS_INCLUDE_QUEUE_METRICS` env var to `true`. You can adjust the refresh rate with `N8N_METRICS_QUEUE_METRICS_INTERVAL`.

n8n gathers these metrics from Bull and exposes them on the main instances. On multi-main setups, when aggregating queries, you can identify the leader using the `instance_role_leader` gauge, set to `1` for the leader main and `0` otherwise.

```
# HELP n8n_scaling_mode_queue_jobs_active Current number of jobs
being processed across all workers in scaling mode.
# TYPE n8n_scaling_mode_queue_jobs_active gauge
n8n_scaling_mode_queue_jobs_active 0

# HELP n8n_scaling_mode_queue_jobs_completed Total number of jobs
completed across all workers in scaling mode since instance start.
# TYPE n8n_scaling_mode_queue_jobs_completed counter
n8n_scaling_mode_queue_jobs_completed 0

# HELP n8n_scaling_mode_queue_jobs_failed Total number of jobs
failed across all workers in scaling mode since instance start.
# TYPE n8n_scaling_mode_queue_jobs_failed counter
n8n_scaling_mode_queue_jobs_failed 0

# HELP n8n_scaling_mode_queue_jobs_waiting Current number of
enqueued jobs waiting for pickup in scaling mode.
# TYPE n8n_scaling_mode_queue_jobs_waiting gauge
```

```
n8n_scaling_mode_queue_jobs_waiting 0
```

Supported databases

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

Database type by n8n installation

The database type used varies depending on your n8n installation:

Self-hosted n8n

By default, self-hosted installations use **SQLite**. You can optionally configure PostgreSQL by setting the appropriate environment variables (see PostgresDB configuration).

n8n Cloud

n8n Cloud installations use different databases depending on your plan tier:

- **SQLite**: Trial, Starter, and Pro plans, as well as legacy Enterprise plans
- **PostgreSQL**: Enterprise Scaling plans only

Shared settings

The following environment variables get used by all databases:

- `DB_TABLE_PREFIX` (default: -) - Prefix for table names

PostgresDB

To use PostgresDB as the database, you can provide the following environment variables:

- `DB_TYPE=postgresdb`
- `DB_POSTGRESDB_DATABASE` (default: ‘n8n’)
- `DB_POSTGRESDB_HOST` (default: ‘localhost’)
- `DB_POSTGRESDB_PORT` (default: 5432)
- `DB_POSTGRESDB_USER` (default: ‘postgres’)
- `DB_POSTGRESDB_PASSWORD` (default: empty)
- `DB_POSTGRESDB_SCHEMA` (default: ‘public’)
- `DB_POSTGRESDB_SSL_CA` (default: undefined): Path to the server’s CA certificate used to validate the connection (opportunistic encryption isn’t supported)
- `DB_POSTGRESDB_SSL_CERT` (default: undefined): Path to the client’s TLS certificate
- `DB_POSTGRESDB_SSL_KEY` (default: undefined): Path to the client’s private key corresponding to the certificate
- `DB_POSTGRESDB_SSL_REJECT_UNAUTHORIZED` (default: true): If TLS connections that fail validation should be rejected

```
export DB_TYPE=postgresdb
export DB_POSTGRESDB_DATABASE=n8n
export DB_POSTGRESDB_HOST=postgresdb
export DB_POSTGRESDB_PORT=5432
export DB_POSTGRESDB_USER=n8n
export DB_POSTGRESDB_PASSWORD=n8n
export DB_POSTGRESDB_SCHEMA=n8n

# optional:
export DB_POSTGRESDB_SSL_CA=$(pwd)/ca.crt
export DB_POSTGRESDB_SSL_REJECT_UNAUTHORIZED=false

n8n start
```

Required permissions

n8n needs to create and modify the schemas of the tables it uses.

Recommended permissions:

```
CREATE DATABASE n8n-db;
CREATE USER n8n-user WITH PASSWORD 'random-password';
GRANT ALL PRIVILEGES ON DATABASE n8n-db TO n8n-user;
```

TLS

You can choose between these configurations:

- Not declaring (default): Connect with `SSL=off`
- Declaring only the CA and unauthorized flag: Connect with `SSL=on` and verify the server's signature
- Declaring `_{CERT,KEY}` and the above: Use the certificate and key for client TLS authentication

SQLite

This is the default database that gets used if nothing is defined.

The database file is located at: `~/.n8n/database.sqlite`

Task runners

Task runners are a generic mechanism to execute tasks in a secure and performant way. They're used to execute user-provided JavaScript and Python code in the Code node.

This document describes how task runners work and how you can configure them.

How it works

The task runner feature consists of these components: one or more task runners, a task broker, and a task requester.

[Image: Task runner overview]

Task runners connect to the task broker using a websocket connection. A task requester submits a task request to the broker where an available task runner can pick it up for execution.

The runner executes the task and submits the results to the task requester. The task broker coordinates communication between the runner and the requester.

The n8n instance (main and worker) acts as the broker. The Code node in this case is the task requester.

Task runner modes

You can use task runners in two different modes: internal and external.

Internal mode

In internal mode, the n8n instance launches the task runner as a child process. The n8n process monitors and manages the life cycle of the task runner. The task runner process shares the same `uid` and `gid` as n8n. This is **not** recommended for production.

External mode

In external mode, a launcher application launches task runners on demand and manages their lifecycle. Typically, this means that next to n8n you add a sidecar container running the `n8nio/runners` image containing the launcher, the JS task runner and the Python task runner. This sidecar container is independent from the n8n instance.

[Image: Task runner deployed as a side-car container]

When using Queue mode, each worker needs to have its own sidecar container for task runners.

In addition, if you haven't enabled offloading manual executions to workers (if you aren't setting `OFFLOAD_MANUAL_EXECUTIONS_TO_WORKERS=true` in your configuration), then your main instance will run manual executions and needs its own sidecar container for task runners as well. Please note that running n8n with offloading disabled isn't recommended for production.

Setting up external mode

In external mode, you run the `n8nio/runners` image as a sidecar container next to n8n. Below you will find a docker compose as a reference. Keep in mind that the `n8nio/runners` image version must match that of the `n8nio/n8n` image, and the n8n version must be >=1.111.0.

```
services:
  n8n:
    image: n8nio/n8n:1.111.0
    container_name: n8n-main
    environment:
      - N8N_RUNNERS_ENABLED=true
```

```
      - N8N_RUNNERS_MODE=external
      - N8N_RUNNERS_BROKER_LISTEN_ADDRESS=0.0.0.0
      - N8N_RUNNERS_AUTH_TOKEN=your-secret-here
      - N8N_NATIVE_PYTHON_RUNNER=true
    ports:
      - "5678:5678"
    volumes:
      - n8n_data:/home/node/.n8n
    # etc.

  task-runners:
    image: n8nio/runners:1.111.0
    container_name: n8n-runners
    environment:
      - N8N_RUNNERS_TASK_BROKER_URI=http://n8n-main:5679
      - N8N_RUNNERS_AUTH_TOKEN=your-secret-here
      # etc.
    depends_on:
      - n8n

volumes:
  n8n_data:
```

There are three layers of configuration: the n8n container, the runners container, and the launcher inside the runners container.

Configuring n8n container in external mode

These are the main environment variables that you can set on the n8n container running in external mode:

Environment variables	Description
`N8N_RUNNERS_ENABLED=true`	Enables task runners.
`N8N_RUNNERS_MODE=external`	Use task runners in external mode.
`N8N_RUNNERS_AUTH_TOKEN=<random secure shared secret>`	A shared secret task runners use to connect to the broker.
`N8N_RUNNERS_BROKER_LISTEN_ADDRESS=0.0.0.0`	By default, the task broker only listens to localhost. When using multiple containers (for example, with Docker Compose), it needs to be able to accept external connections.

For full list of environment variables see task runner environment variables.

Configuring runners container in external mode

These are the main environment variables that you can set on the runners container running in external mode:

Environment variables	Description
`N8N_RUNNERS_AUTH_TOKEN=<random secure`	The shared secret the

`shared secret>`	task runner uses to connect to the broker.
`N8N_RUNNERS_TASK_BROKER_URI=localhost:5679`	The address of the task broker server within the n8n instance.
`N8N_RUNNERS_AUTO_SHUTDOWN_TIMEOUT=15`	Number of seconds of inactivity to wait before shutting down the task runner process. The launcher will automatically start the runner again when there are new tasks to execute. Set to `0` to disable automatic shutdown.

For full list of environment variables see task runner environment variables.

Configuring launcher in runners container in external mode

The launcher reads environment variables from runners container environment, and performs the following actions:

- Passing environment variables from the launcher's own environment to all runners (`allowed-env`)
- Setting specific environment variables on specific runners (`env-overrides`)

Which environment variables to pass and to set are defined in the launcher config file included in the runners image. This config file is located in the container at `/etc/task-runners.json`. To learn more about the launcher config file, refer to the Config file documentation.

The default launcher configuration file is locked down, but you can edit this file, for example, to allowlist first- or third-party modules. To customize the launcher configuration file, mount to this path:

```
path/to/n8n-task-runners.json:/etc/n8n-task-runners.json
```

Adding extra dependencies

1. Extend the `n8nio/runners` image

You can extend the `n8nio/runners` image to add extra dependencies to the runners. You'll need `n8nio/runners:1.121.0` or later to do this.

```
        FROM n8nio/runners:1.121.0
        USER root
        RUN cd /opt/runners/task-runner-javascript && pnpm add moment uuid
        RUN cd /opt/runners/task-runner-python && uv pip install numpy
pandas
        COPY n8n-task-runners.json /etc/n8n-task-runners.json
        USER runner
```

You must also allowlist any first-party or third-party packages for use by the Code node. Do this by editing the configuration file `n8n-task-runners.json` to include the packages in your extended image.

```
{
  "task-runners": [
    {
      "runner-type": "javascript",
      "env-overrides": {
        "NODE_FUNCTION_ALLOW_BUILTIN": "crypto",         // <-- allowlist Node.js builtin modules here
        "NODE_FUNCTION_ALLOW_EXTERNAL": "moment,uuid",   // <-- allowlist third-party JS packages here
      }
    },
    {
      "runner-type": "python",
      "env-overrides": {
        "PYTHONPATH": "/opt/runners/task-runner-python",
        "N8N_RUNNERS_STDLIB_ALLOW": "json",              // <-- allowlist Python standard library packages here
        "N8N_RUNNERS_EXTERNAL_ALLOW": "numpy,pandas"     // <-- allowlist third-party Python packages here
      }
    }
  ]
}
```

- `NODE_FUNCTION_ALLOW_BUILTIN`: comma-separated list of allowed node builtin modules.
- `NODE_FUNCTION_ALLOW_EXTERNAL`: comma-separated list of allowed JS packages.
- `N8N_RUNNERS_STDLIB_ALLOW`: comma-separated list of allowed Python standard library packages.
- `N8N_RUNNERS_EXTERNAL_ALLOW`: comma-separated list of allowed Python packages.

2. Build your custom image

For example, from the n8n repository root:

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

3. Run the image

For example:

```
docker run --rm -it \
  -e N8N_RUNNERS_AUTH_TOKEN=test \
  -e N8N_RUNNERS_LAUNCHER_LOG_LEVEL=debug \
  -e N8N_RUNNERS_TASK_BROKER_URI=http://host.docker.internal:5679 \
  -p 5680:5680 \
  n8nio/runners:custom
```

Configure self-hosted n8n for user management

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

This document describes how to configure your n8n instance to support user management, and the steps to start inviting users.

Refer to the main User management guide for more information about usage, including:

- Managing users
- Account types
- Best practices

For LDAP setup information, refer to LDAP.

For SAML setup information, refer to SAML.

Setup

There are three stages to set up user management in n8n:

1. Configure your n8n instance to use your SMTP server.
2. Start n8n and follow the setup steps in the app.
3. Invite users.

Step one: SMTP

n8n recommends setting up an SMTP server, for user invites and password resets.

Get the following information from your SMTP provider:

- Server name
- SMTP username
- SMTP password
- SMTP sender name

To set up SMTP with n8n, configure the SMTP environment variables for your n8n instance. For information on how to set environment variables, refer to Configuration

Variable	Type	Descript
`N8N_EMAIL_MODE`	string	`smtp`
`N8N_SMTP_HOST`	string	*your_SMTP_ser*
`N8N_SMTP_PORT`	number	*your_SMTP_ser* Default is `465`.
`N8N_SMTP_USER`	string	*your_SMTP_use*
`N8N_SMTP_PASS`	string	*your_SMTP_pas*
`N8N_SMTP_OAUTH_SERVICE_CLIENT`	string	*your_OAuth_ser*
`N8N_SMTP_OAUTH_PRIVATE_KEY`	string	*your_OAuth_pri*
`N8N_SMTP_SENDER`	string	Sender email ac can optionally i sender name. E

		with name: *N8N* <*contact@n8n.co*
`N8N_SMTP_SSL`	boolean	Whether to use SMTP (true) or Defaults to `true`
`N8N_UM_EMAIL_TEMPLATES_INVITE`	string	Full path to you email template. overrides the de template for inv
`N8N_UM_EMAIL_TEMPLATES_PWRESET`	string	Full path to you email template. overrides the de template for pas reset emails.
`N8N_UM_EMAIL_TEMPLATES_WORKFLOW_SHARED`	String	Overrides the d HTML template notifying users credential was s Provide the full the template.
`N8N_UM_EMAIL_TEMPLATES_CREDENTIALS_SHARED`	String	Overrides the d HTML template notifying users credential was s Provide the full the template.
`N8N_UM_EMAIL_TEMPLATES_PROJECT_SHARED`	String	Overrides the d HTML template notifying users project was sha Provide the full the template.

If your n8n instance is already running, you need to restart it to enable the new SMTP settings.

Step two: In-app setup

–8<– “_snippets/user-management/in-app-setup.md”

Step three: Invite users

–8<– “_snippets/user-management/invite-users.md”

Logging in n8n

Logging is an important feature for debugging. n8n uses the winston logging library.

Setup

To set up logging in n8n, you need to set the following environment variables (you can also set the values in the configuration file)

Setting in the configuration file	Using environment variables	D
n8n.log.level	N8N_LOG_LEVEL	The log ou available lowest to error, war The defau You can le these opti
n8n.log.output	N8N_LOG_OUTPUT	Where to available and `file`. can be use comma (,) by default
n8n.log.file.location	N8N_LOG_FILE_LOCATION	The log fil only if log file. By de `<n8nFolder` is used.
n8n.log.file.fileSizeMax	N8N_LOG_FILE_SIZE_MAX	The maxi for each l n8n uses 1
n8n.log.file.fileCountMax	N8N_LOG_FILE_COUNT_MAX	The maxi files to ke value is 1 should be workers.

```
# Set the logging level to 'debug'
export N8N_LOG_LEVEL=debug

# Set log output to both console and a log file
export N8N_LOG_OUTPUT=console,file

# Set a save location for the log file
export N8N_LOG_FILE_LOCATION=/home/jim/n8n/logs/n8n.log

# Set a 50 MB maximum size for each log file
export N8N_LOG_FILE_SIZE_MAX=50

# Set 60 as the maximum number of log files to be kept
export N8N_LOG_FILE_COUNT_MAX=60
```

Log levels

n8n uses standard log levels to report:

- `silent`: outputs nothing at all
- `error`: outputs only errors and nothing else
- `warn`: outputs errors and warning messages
- `info`: contains useful information about progress
- `debug`: the most verbose output. n8n outputs a lot of information to help you debug issues.

Development

During development, adding log messages is a good practice. It assists in debugging errors. To configure logging for development, follow the guide below.

Implementation details

n8n uses the `LoggerProxy` class, located in the `workflow` package. Calling the `LoggerProxy.init()` by passing in an instance of `Logger`, initializes the class before the usage.

The initialization process happens only once. The `start.ts` file already does this process for you. If you are creating a new command from scratch, you need to initialize the `LoggerProxy` class.

Once the `Logger` implementation gets created in the `cli` package, it can be obtained by calling the `getInstance` convenience method from the exported module.

Check the start.ts file to learn more about how this process works.

Adding logs

Once the `LoggerProxy` class gets initialized in the project, you can import it to any other file and add logs.

Convenience methods are provided for all logging levels, so new logs can be added whenever needed using the format `Logger.<logLevel>('<message>', ...meta)`, where `meta` represents any additional properties desired beyond `message`.

In the example above, we use the standard log levels described above. The `message` argument is a string, and `meta` is a data object.

```
        // You have to import the LoggerProxy. We rename it to Logger to
make it easier

        import {
            LoggerProxy as Logger
        } from 'n8n-workflow';

        // Info-level logging of a trigger function, with workflow name and
workflow ID as additional metadata properties

        Logger.info(`Polling trigger initiated for workflow
"${workflow.name}"`, {workflowName: workflow.name, workflowId:
workflow.id});
```

When creating new loggers, some useful standards to keep in mind are:

- Craft log messages to be as human-readable as possible. For example, always wrap names in quotes.
- Duplicating information in the log message and metadata, like workflow name in the above example, can be useful as messages are easier to search and metadata enables easier filtering.
- Include multiple IDs (for example, `executionId`, `workflowId`, and `sessionId`) throughout all logs.
- Use node types instead of node names (or both) as this is more

consistent, and so easier to search.

Front-end logs

As of now, front-end logs aren't available. Using `Logger` or `LoggerProxy` would yield errors in the `editor-ui` package. This functionality will get implemented in the future versions.

Monitoring

There are three API endpoints you can call to check the status of your instance: `/healthz`, `healthz/readiness`, and `/metrics`.

healthz and healthz/readiness

The `/healthz` endpoint returns a standard HTTP status code. 200 indicates the instance is reachable. It doesn't indicate DB status. It's available for both self-hosted and Cloud users.

Access the endpoint:

```
<your-instance-url>/healthz
```

The `/healthz/readiness` endpoint is similar to the `/healthz` endpoint, but it returns a HTTP status code of 200 if the DB is connected and migrated and therefore the instance is ready to accept traffic.

Access the endpoint:

```
<your-instance-url>/healthz/readiness
```

metrics

The `/metrics` endpoint provides more detailed information about the current status of the instance.

Access the endpoint:

```
<your-instance-url>/metrics
```

Enable metrics and healthz for self-hosted n8n

The `/metrics` and `/healthz` endpoints are disabled by default. To enable them, configure your n8n instance:

```
# metrics
N8N_METRICS=true
# healthz
QUEUE_HEALTH_CHECK_ACTIVE=true
```

Refer to Configuration methods for more information on how to configure your instance using environment variables.

Security audit

You can run a security audit on your n8n instance, to detect common security issues.

Run an audit

You can run an audit using the CLI, the public API, or the n8n node.

CLI

Run `n8n audit`.

API

Make a `POST` call to the `/audit` endpoint. You must authenticate as the instance owner.

n8n node

Add the n8n node to your workflow. Select **Resource** > **Audit** and **Operation** > **Generate**.

Report contents

The audit generates five risk reports:

Credentials

This report shows:

- Credentials not used in a workflow.
- Credentials not used in an active workflow.
- Credentials not use in a recently active workflow.

Database

This report shows:

- Expressions used in **Execute Query** fields in SQL nodes.
- Expressions used in **Query Parameters** fields in SQL nodes.
- Unused **Query Parameters** fields in SQL nodes.

File system

This report lists nodes that interact with the file system.

Nodes

This report shows:

- Official risky nodes. These are n8n built in nodes. You can use them to fetch and run any code on the host system, which exposes the instance to exploits. You can view the list in n8n code | Audit constants, under `OFFICIAL_RISKY_NODE_TYPES`.
- Community nodes.
- Custom nodes.

Instance

This report shows:

- Unprotected webhooks in the instance.
- Missing security settings
- If your instance is outdated.

Scaling n8n

When running n8n at scale, with a large number of users, workflows, or executions, you need to change your n8n configuration to ensure good performance.

n8n can run in different modes depending on your needs. The `queue` mode provides the best scalability. Refer to Queue mode for configuration details.

You can configure data saving and pruning to improve database performance. Refer to Execution data for details.

Performance and benchmarking

n8n can handle up to 220 workflow executions per second on a single instance, with the ability to scale up further by adding more instances.

This document outlines n8n's performance benchmarking. It describes the factors that affect performance, and includes two example benchmarks.

Performance factors

The performance of n8n depends on factors including:

- The workflow type
- The resources available to n8n
- How you configure n8n's scaling options

Run your own benchmarking

To get an accurate estimate for your use case, run n8n's benchmarking framework. The repository contains more information about the benchmarking.

Example: Single instance performance

This test measures how response time increases as requests per second increase. It looks at the response time when calling the Webhook Trigger node.

Setup:

- Hardware: ECS c5a.large instance (4GB RAM)
- n8n setup: Single n8n instance (running in main mode, with Postgres database)
- Workflow: Webhook Trigger node, Edit Fields node

> [Image: Graph showing n8n response times by requests per second]
> This graph shows the percentage of requests to the Webhook Trigger node getting a response within 100 seconds, and how that varies with load. Under higher loads n8n usually still processes the data, but takes over 100s to respond.

Example: Multi-instance performance

This test measures how response time increases as requests per second increase. It looks at the response time when calling the Webhook Trigger node.

Setup:

- Hardware: seven ECS c5a.4xlarge instances (8GB RAM each)
- n8n setup: two webhook instances, four worker instances, one database instance (MySQL), one main instance running n8n and Redis
- Workflow: Webhook Trigger node, Edit Fields node
- Multi-instance setups use Queue mode

> [Image: Graph showing n8n response times by requests per second]
> This graph shows the percentage of requests to the Webhook Trigger node getting a response within 100 seconds, and how that varies with load. Under higher loads n8n usually still processes the data, but takes over 100s to respond.

Queue mode

You can run n8n in different modes depending on your needs. The queue mode provides the best scalability.

How it works

When running in queue mode, you have multiple n8n instances set up, with one main instance receiving workflow information (such as triggers) and the worker instances performing the executions.

Each worker is its own Node.js instance, running in `main` mode, but able to handle multiple simultaneous workflow executions due to their high IOPS (input-output operations per second).

By using worker instances and running in queue mode, you can scale n8n up (by adding workers) and down (by removing workers) as needed to handle the workload.

This is the process flow:

1. The main n8n instance handles timers and webhook calls, generating (but not running) a workflow execution.
2. It passes the execution ID to a message broker, Redis, which maintains the queue of pending executions and allows the next available worker to pick them up.
3. A worker in the pool picks up message from Redis.
4. The worker uses the execution ID to get workflow information from the database.
5. After completing the workflow execution, the worker:
   - Writes the results to the database.
   - Posts to Redis, saying that the execution has finished.
6. Redis notifies the main instance.

[Image: “Diagram showing the flow of data between the main n8n instance, Redis, the n8n workers, and the n8n database”]

Configuring workers

Workers are n8n instances that do the actual work. They receive information from the main n8n process about the workflows that have to get executed, execute the workflows, and update the status after each execution is complete.

Set encryption key

n8n automatically generates an encryption key upon first startup. You can also provide your own custom key using environment variable if desired.

The encryption key of the main n8n instance must be shared with all worker and webhooks processor nodes to ensure these worker nodes are able to access credentials stored in the database.

Set the encryption key for each worker node in a configuration file or by setting the corresponding environment variable:

```
export N8N_ENCRYPTION_KEY=<main_instance_encryption_key>
```

Set executions mode

Set the environment variable `EXECUTIONS_MODE` to `queue` on the main instance and any workers using the following command.

```
export EXECUTIONS_MODE=queue
```

Alternatively, you can set `executions.mode` to `queue` in the configuration file.

Start Redis

To run Redis in a Docker container, follow the instructions below:

Run the following command to start a Redis instance:

```
docker run --name some-redis -p 6379:6379  -d redis
```

By default, Redis runs on `localhost` on port `6379` with no password. Based on your Redis configuration, set the following configurations for the main n8n process. These will allow n8n to interact with Redis.

Using configuration file	Using environment variables	D
`queue.bull.redis.host:localhost`	`QUEUE_BULL_REDIS_HOST=localhost`	By R or lo
`queue.bull.redis.port:6379`	`QUEUE_BULL_REDIS_PORT=6379`	Th po If ru di po co th

You can also set the following optional configurations:

Using configuration file	Using environment va
`queue.bull.redis.username:USERNAME`	`QUEUE_BULL_REDIS_USERNAME`
`queue.bull.redis.password:PASSWORD`	`QUEUE_BULL_REDIS_PASSWORD`
`queue.bull.redis.db:0`	`QUEUE_BULL_REDIS_DB`
`queue.bull.redis.timeoutThreshold:10000ms`	`QUEUE_BULL_REDIS_TIMEOUT_`

```
queue.bull.gracefulShutdownTimeout:30      N8N_GRACEFUL_SHUTDOWN_TIM
```

Now you can start your n8n instance and it will connect to your Redis instance.

Start workers

You will need to start worker processes to allow n8n to execute workflows. If you want to host workers on a separate machine, install n8n on the machine and make sure that it's connected to your Redis instance and the n8n database.

Start worker processes by running the following command from the root directory:

```
./packages/cli/bin/n8n worker
```

If you're using Docker, use the following command:

```
docker run --name n8n-queue -p 5679:5678 docker.n8n.io/n8nio/n8n
worker
```

You can set up multiple worker processes. Make sure that all the worker processes have access to Redis and the n8n database.

Worker server

Each worker process runs a server that exposes optional endpoints:

- `/healthz`: returns whether the worker is up, if you enable the `QUEUE_HEALTH_CHECK_ACTIVE` environment variable
- `/healthz/readiness`: returns whether worker's DB and Redis connections are ready, if you enable the `QUEUE_HEALTH_CHECK_ACTIVE` environment variable
- credentials overwrite endpoint
- `/metrics`

View running workers

You can view running workers and their performance metrics in n8n by selecting **Settings > Workers**.

Running n8n with queues

When running n8n with queues, all the production workflow executions get processed by worker processes. This means that even the webhook calls get delegated to the worker processes, which might add some overhead and extra latency.

Redis acts as the message broker, and the database persists data, so access to both is required. Running a distributed system with this setup over SQLite isn't supported.

Webhook processors

Webhook processors are another layer of scaling in n8n. Configuring the webhook processor is optional, and allows you to scale the incoming webhook requests.

This method allows n8n to process a huge number of parallel requests. All you have to do is add more webhook processes and workers accordingly. The webhook process will listen to requests on the same port (default: `5678`). Run these processes in containers or separate machines, and have a load balancing system to route requests accordingly.

n8n doesn't recommend adding the main process to the load balancer pool. If you add the main process to the pool, it will receive requests and possibly a heavy load. This will result in degraded performance for editing, viewing, and interacting with the n8n UI.

You can start the webhook processor by executing the following command from the root directory:

```
./packages/cli/bin/n8n webhook
```

If you're using Docker, use the following command:

```
docker run --name n8n-queue -p 5679:5678 -e "EXECUTIONS_MODE=queue"
docker.n8n.io/n8nio/n8n webhook
```

Configure webhook URL

To configure your webhook URL, execute the following command on the machine running the main n8n instance:

```
export WEBHOOK_URL=https://your-webhook-url.com
```

You can also set this value in the configuration file.

Configure load balancer

When using multiple webhook processes you will need a load balancer to route requests. If you are using the same domain name for your n8n instance and the webhooks, you can set up your load balancer to route requests as follows:

- Redirect any request that matches `/webhook/*` to the webhook servers pool
- All other paths (the n8n internal API, the static files for the editor, etc.) should get routed to the main process

Note: The default URL for manual workflow executions is `/webhook-test/*`. Make sure that these URLs route to your main process.

You can change this path in the configuration file `endpoints.webhook` or using the `N8N_ENDPOINT_WEBHOOK` environment variable. If you change these, update your load balancer accordingly.

Disable webhook processing in the main process (optional)

You have webhook processors to execute the workflows. You can disable the webhook processing in the main process. This will make sure to execute all webhook executions in the webhook processors. In the configuration file set `endpoints.disableProductionWebhooksOnMainProcess` to `true` so that n8n doesn't process webhook requests on the main process.

Alternatively, you can use the following command:

```
export N8N_DISABLE_PRODUCTION_MAIN_PROCESS=true
```

When disabling the webhook process in the main process, run the main process and don't add it to the load balancer's webhook pool.

Configure worker concurrency

You can define the number of jobs a worker can run in parallel by using the `concurrency` flag. It defaults to `10`. To change it:

```
n8n worker --concurrency=5
```

Concurrency and scaling recommendations

n8n recommends setting concurrency to 5 or higher for your worker instances. Setting low concurrency values with a large numbers of workers can exhaust your database's connection pool, leading to processing delays and failures.

Multi-main setup

In queue mode you can run more than one `main` process for high availability.

In a single-mode setup, the `main` process does two sets of tasks:

- **regular tasks**, such as running the API, serving the UI, and listening for webhooks, and
- **at-most-once tasks**, such as running non-HTTP triggers (timers, pollers, and persistent connections like RabbitMQ and IMAP), and pruning executions and binary data.

In a multi-main setup, there are two kinds of `main` processes:

- **followers**, which run **regular tasks**, and
- the **leader**, which runs **both regular and at-most-once tasks**.

Leader designation

In a multi-main setup, all main instances handle the leadership process transparently to users. In case the current leader becomes unavailable, for example because it crashed or its event loop became too busy, other followers can take over. If the previous leader becomes responsive again, it becomes a follower.

Configuring multi-main setup

To deploy n8n in multi-main setup, ensure:

- All `main` processes are running in queue mode and are connected to Postgres and Redis.
- All `main` and `worker` processes are running the same version of n8n.
- All `main` processes have set the environment variable `N8N_MULTI_MAIN_SETUP_ENABLED` to `true`.
- All `main` processes are running behind a load balancer with session persistence (sticky sessions) enabled.

If needed, you can adjust the leader key options:

Using configuration file	Using environment variables	D
`multiMainSetup.ttl:10`	`N8N_MULTI_MAIN_SETUP_KEY_TTL=10`	Ti (i fo ke m
`multiMainSetup.interval:3`	`N8N_MULTI_MAIN_SETUP_CHECK_INTERVAL=3`	In se le in m

Self-hosted concurrency control

In regular mode, n8n doesn't limit how many production executions may run at the same time. This can lead to a scenario where too many concurrent executions thrash the event loop, causing performance degradation and unresponsiveness.

To prevent this, you can set a concurrency limit for production executions in regular mode. Use this to control how many production executions run concurrently, and queue up any concurrent production executions over the limit. These executions remain in the queue until concurrency capacity frees up, and are then processed in FIFO order.

Concurrency control is disabled by default. To enable it:

```
export N8N_CONCURRENCY_PRODUCTION_LIMIT=20
```

Keep in mind:

- Concurrency control applies only to production executions: those started from a webhook or trigger node. It doesn't apply to any other kinds, such as manual executions, sub-workflow executions, error executions, or started from CLI.

- You can’t retry queued executions. Cancelling or deleting a queued execution also removes it from the queue.
- On instance startup, n8n resumes queued executions up to the concurrency limit and re-enqueues the rest.
- To monitor concurrency control, watch logs for executions being added to the queue and released. In a future version, n8n will show concurrency control in the UI.

When you enable concurrency control, you can view the number of active executions and the configured limit at the top of a project’s or workflow’s executions tab.

Comparison to queue mode

In queue mode, you can control how many jobs a worker may run concurrently using the `--concurrency` flag.

Concurrency control in queue mode is a separate mechanism from concurrency control in regular mode, but the environment variable `N8N_CONCURRENCY_PRODUCTION_LIMIT` controls both of them. In queue mode, n8n takes the limit from this variable if set to a value other than `-1`, falling back to the `--concurrency` flag or its default.

Execution data

Depending on your executions settings and volume, your n8n database can grow in size and run out of storage.

To avoid this, n8n recommends that you don’t save unnecessary data, and enable pruning of old executions data.

To do this, configure the corresponding environment variables.

Reduce saved data

You can select which executions data n8n saves. For example, you can save only executions that result in an `Error`.

```
# npm
# Save executions ending in errors
export EXECUTIONS_DATA_SAVE_ON_ERROR=all

# Don't save successful executions
export EXECUTIONS_DATA_SAVE_ON_SUCCESS=none

# Don't save node progress for each execution
export EXECUTIONS_DATA_SAVE_ON_PROGRESS=false

# Don't save manually launched executions
export EXECUTIONS_DATA_SAVE_MANUAL_EXECUTIONS=false

# Docker
docker run -it --rm \
 --name n8n \
```

```
 -p 5678:5678 \
 -e EXECUTIONS_DATA_SAVE_ON_ERROR=all \
 -e EXECUTIONS_DATA_SAVE_ON_SUCCESS=none \
 -e EXECUTIONS_DATA_SAVE_ON_PROGRESS=true \
 -e EXECUTIONS_DATA_SAVE_MANUAL_EXECUTIONS=false \
 docker.n8n.io/n8nio/n8n

# Docker Compose
n8n:
    environment:
      - EXECUTIONS_DATA_SAVE_ON_ERROR=all
      - EXECUTIONS_DATA_SAVE_ON_SUCCESS=none
      - EXECUTIONS_DATA_SAVE_ON_PROGRESS=true
      - EXECUTIONS_DATA_SAVE_MANUAL_EXECUTIONS=false
```

Enable executions pruning

Executions pruning deletes finished executions along with their execution data and binary data on a regular schedule. n8n enables pruning by default. For performance reasons, pruning first marks targets for deletion, and then later permanently removes them.

n8n prunes executions when **either** of the following condition occur:

- **Age**: The execution finished more than `EXECUTIONS_DATA_MAX_AGE` hours ago (default: 336 hours -> 14 days).
- **Count**: The total number of executions exceeds `EXECUTIONS_DATA_PRUNE_MAX_COUNT` (default: 10,000). When this occurs, n8n deletes executions from oldest to newest.

Keep in mind:

- Executions with the `new`, `running`, or `waiting` status aren't eligible for pruning.
- Annotated executions (for example, executions with tags or ratings) are never pruned.
- Pruning honors a safety buffer period of `EXECUTIONS_DATA_HARD_DELETE_BUFFER` hours (default: 1h), to ensure recent data remains available while the user is building or debugging a workflow.

```
# Enable executions pruning
export EXECUTIONS_DATA_PRUNE=true

# How old (hours) a finished execution must be to qualify for soft-deletion
export EXECUTIONS_DATA_MAX_AGE=168

# Max number of finished executions to keep. May not strictly prune back down to the exact max count. Set to `0` for unlimited.
export EXECUTIONS_DATA_PRUNE_MAX_COUNT=50000

# Docker
docker run -it --rm \
 --name n8n \
 -p 5678:5678 \
 -e EXECUTIONS_DATA_PRUNE=true \
 -e EXECUTIONS_DATA_MAX_AGE=168 \
 docker.n8n.io/n8nio/n8n
```

```
# Docker Compose
n8n:
    environment:
      - EXECUTIONS_DATA_PRUNE=true
      - EXECUTIONS_DATA_MAX_AGE=168
      - EXECUTIONS_DATA_PRUNE_MAX_COUNT=50000
```

–8<– “_snippets/self-hosting/scaling/binary-data-pruning.md”

Binary data

Binary data is any file-type data, such as image files or documents generated or processed during the execution of a workflow.

Enable filesystem mode

When handling binary data, n8n keeps the data in memory by default. This can cause crashes when working with large files.

To avoid this, change the `N8N_DEFAULT_BINARY_DATA_MODE` environment variable to `filesystem`. This causes n8n to save data to disk, instead of using memory.

If you’re using queue mode, switch this to `database`. n8n doesn’t support `filesystem` mode with queue mode.

Binary data pruning

n8n executes binary data pruning as part of execution data pruning. Refer to Execution data | Enable executions pruning for details.

If you configure multiple binary data modes, binary data pruning operates on the active binary data mode. For example, if your instance stored data in S3, and you later switched to filesystem mode, n8n only prunes binary data in the filesystem. Refer to External storage for details.

External storage

n8n can store binary data produced by workflow executions externally. This feature is useful to avoid relying on the filesystem for storing large amounts of binary data.

n8n will introduce external storage for other data types in the future.

Storing n8n’s binary data in S3

n8n supports AWS S3 as an external store for binary data produced by workflow executions. You can use other S3-compatible services like Cloudflare R2 and Backblaze B2, but n8n doesn’t officially support these.

Setup

Create and configure a bucket following the AWS documentation. You can use the following policy, replacing `<bucket-name>` with the name of the bucket you created:

```
        {
         "Version": "2012-10-17",
         "Statement": [
          {
           "Sid": "VisualEditor0",
           "Effect": "Allow",
           "Action": ["s3:*"],
           "Resource": ["arn:aws:s3:::<bucket-name>", "arn:aws:s3:::<bucket-
name>/*"]
          }
         ]
        }
```

Set a bucket-level lifecycle configuration so that S3 automatically deletes old binary data. n8n delegates pruning of binary data to S3, so setting a lifecycle configuration is required unless you want to preserve binary data indefinitely.

Once you finish creating the bucket, you will have a host, bucket name and region, and an access key ID and secret access key. You need to set them in n8n's environment:

```
        export N8N_EXTERNAL_STORAGE_S3_HOST=... # example: s3.us-east-
1.amazonaws.com
        export N8N_EXTERNAL_STORAGE_S3_BUCKET_NAME=...
        export N8N_EXTERNAL_STORAGE_S3_BUCKET_REGION=...
        export N8N_EXTERNAL_STORAGE_S3_ACCESS_KEY=...
        export N8N_EXTERNAL_STORAGE_S3_ACCESS_SECRET=...
```

Tell n8n to store binary data in S3:

```
        export N8N_AVAILABLE_BINARY_DATA_MODES=filesystem,s3
        export N8N_DEFAULT_BINARY_DATA_MODE=s3
```

Restart the server to load the new configuration.

Usage

After you enable S3, n8n writes and reads any new binary data to and from the S3 bucket. n8n writes binary data to your S3 bucket in this format:

```
workflows/{workflowId}/executions/{executionId}/binary_data/{binaryF
```

n8n continues to read older binary data stored in the filesystem from the filesystem, if `filesystem` remains listed as an option in `N8N_AVAILABLE_BINARY_DATA_MODES`.

If you store binary data in S3 and later switch to filesystem mode, the instance continues to read any data stored in S3, as long as `s3` remains listed in `N8N_AVAILABLE_BINARY_DATA_MODES` and your S3 credentials remain valid.

-8<- "_snippets/self-hosting/scaling/binary-data-pruning.md"

Memory-related errors

n8n doesn't restrict the amount of data each node can fetch and process. While this gives you freedom, it can lead to errors when workflow executions require more memory than available. This page explains how to identify and avoid these errors.

Identifying out of memory situations

n8n provides error messages that warn you in some out of memory situations. For example, messages such as **Execution stopped at this node (n8n may have run out of memory while executing it)**.

Error messages including **Problem running workflow**, **Connection Lost**, or **503 Service Temporarily Unavailable** suggest that an n8n instance has become unavailable.

When self-hosting n8n, you may also see error messages such as **Allocation failed - JavaScript heap out of memory** in your server logs.

On n8n Cloud, or when using n8n's Docker image, n8n restarts automatically when encountering such an issue. However, when running n8n with npm you might need to restart it manually.

Typical causes

Such problems occur when a workflow execution requires more memory than available to an n8n instance. Factors increasing the memory usage for a workflow execution include:

- Amount of JSON data.
- Size of binary data.
- Number of nodes in a workflow.
- Some nodes are memory-heavy: the Code node and the older Function node can increase memory consumption significantly.
- Manual or automatic workflow executions: manual executions increase memory consumption as n8n makes a copy of the data for the frontend.
- Additional workflows running at the same time.

Avoiding out of memory situations

When encountering an out of memory situation, there are two options: either increase the amount of memory available to n8n or reduce the memory consumption.

Increase available memory

When self-hosting n8n, increasing the amount of memory available to n8n means provisioning your n8n instance with more memory. This may incur additional costs with your hosting provider.

On n8n cloud you need to upgrade to a larger plan.

Reduce memory consumption

This approach is more complex and means re-building the workflows causing the issue. This section provides some guidelines on how to reduce memory consumption. Not all suggestions are applicable to all workflows.

–8<– “_snippets/self-hosting/scaling/reduce-memory-consumption.md”

Increase old memory

This applies to self-hosting n8n. When encountering **JavaScript heap out of memory** errors, it’s often useful to allocate additional memory to the old memory section of the V8 JavaScript engine. To do this, set the appropriate V8 option `--max-old-space-size=SIZE` either through the CLI or through the `NODE_OPTIONS` environment variable.

Securing n8n

Securing your n8n instance can take several forms.

At a high level, you can:

- Conduct a security audit to identify security risks.
- Set up SSL to enforce secure connections.
- Set up Single Sign-On for user account management.
- Use two-factor authentication (2FA) for your users.

More granularly, consider blocking or opting out of features or data collection you don’t want:

- Disable the public API if you aren’t using it.
- Opt out of data collection of the anonymous data n8n collects automatically.
- Block certain nodes from being available to your users.
- Restrict account registration to email-verified users.

Set up SSL

There are two methods to support TLS/SSL in n8n.

Use a reverse proxy (recommended)

Use a reverse proxy like Traefik or a Network Load Balancer (NLB) in front of the n8n instance. This should also take care of certificate renewals.

Refer to Security | Data encryption for more information.

Pass certificates into n8n directly

You can also choose to pass certificates into n8n directly. To do so, set the `N8N_SSL_CERT` and `N8N_SSL_KEY` environment variables to point to your generated certificate and key file.

You'll need to make sure the certificate stays renewed and up to date.

Refer to Deployment environment variables for more information on these variables and Configuration for more information on setting environment variables.

Set up Single Sign-On (SSO)

n8n supports the SAML and OIDC authentication protocols for single sign-on (SSO). See OIDC vs SAML for more general information on the two protocols, the differences between them, and their respective benefits.

- Set up SAML: a general guide to setting up SAML in n8n, and links to resources for common identity providers (IdPs).
- Set up OIDC: a general guide to setting up OpenID Connect (OIDC) SSO in n8n.

Security audit

You can run a security audit on your n8n instance, to detect common security issues.

Run an audit

You can run an audit using the CLI, the public API, or the n8n node.

CLI

Run `n8n audit`.

API

Make a `POST` call to the `/audit` endpoint. You must authenticate as the instance owner.

n8n node

Add the n8n node to your workflow. Select **Resource** > **Audit** and **Operation** > **Generate**.

Report contents

The audit generates five risk reports:

Credentials

This report shows:

- Credentials not used in a workflow.
- Credentials not used in an active workflow.
- Credentials not use in a recently active workflow.

Database

This report shows:

- Expressions used in **Execute Query** fields in SQL nodes.
- Expressions used in **Query Parameters** fields in SQL nodes.
- Unused **Query Parameters** fields in SQL nodes.

File system

This report lists nodes that interact with the file system.

Nodes

This report shows:

- Official risky nodes. These are n8n built in nodes. You can use them to fetch and run any code on the host system, which exposes the instance to exploits. You can view the list in n8n code | Audit constants, under `OFFICIAL_RISKY_NODE_TYPES`.
- Community nodes.
- Custom nodes.

Instance

This report shows:

- Unprotected webhooks in the instance.
- Missing security settings
- If your instance is outdated.

Disable the public REST API

The n8n public REST API allows you to programmatically perform many of the same tasks as you can in the n8n GUI.

If you don't plan on using this API, n8n recommends disabling it to improve the security of your n8n installation.

To disable the public REST API, set the `N8N_PUBLIC_API_DISABLED` environment variable to `true`, for example:

```
export N8N_PUBLIC_API_DISABLED=true
```

Disable the API playground

To disable the API playground, set the `N8N_PUBLIC_API_SWAGGERUI_DISABLED` environment variable to `true`, for example:

```
export N8N_PUBLIC_API_SWAGGERUI_DISABLED=true
```

Related resources

Refer to Deployment environment variables for more information on these environment variables.

Refer to Configuration for more information on setting environment variables.

Data collection

n8n collects some anonymous data from self-hosted n8n installations. Use the instructions below to opt out of data telemetry collection.

Collected data

Refer to Privacy | Data collection in self-hosted n8n for details on the data n8n collects.

How collection works

Your n8n instance sends most data to n8n as the events that generate it occur. Workflow execution counts and an instance pulse are sent periodically (every 6 hours). These data types mostly fall into n8n telemetry collection.

Opting out of data collection

n8n enables telemetry collection by default. To disable it, configure the following environment variables.

Opt out of telemetry events

To opt out of telemetry events, set the `N8N_DIAGNOSTICS_ENABLED` environment variable to false, for example:

```
export N8N_DIAGNOSTICS_ENABLED=false
```

Opt out of checking for new versions of n8n

To opt out of checking for new versions of n8n, set the `N8N_VERSION_NOTIFICATIONS_ENABLED` environment variable to false, for example:

```
export N8N_VERSION_NOTIFICATIONS_ENABLED=false
```

Disable all connection to n8n servers

If you want to fully prevent all communication with n8n's servers, refer to Isolate n8n.

Related resources

Refer to Deployment environment variables for more information on these environment variables.

Refer to Configuration for more information on setting environment variables.

Block access to nodes

For security reasons, you may want to block your users from accessing or working with specific n8n nodes. This is helpful if your users might be untrustworthy.

Use the `NODES_EXCLUDE` environment variable to prevent your users from accessing specific nodes.

Exclude nodes

Update your `NODES_EXCLUDE` environment variable to include an array of strings containing any nodes you want to block your users from using.

For example, setting the variable this way:

```
NODES_EXCLUDE: "[\"n8n-nodes-base.executeCommand\", \"n8n-nodes-base.readWriteFile\"]"
```

Blocks the Execute Command and Read/Write Files from Disk nodes.

Your n8n users won't be able to search for or use these nodes.

Suggested nodes to block

The nodes that can pose security risks vary based on your use case and user profile. Here are some nodes you might want to start with:

- Execute Command
- Read/Write Files from Disk

Related resources

Refer to Nodes environment variables for more information on this environment variable.

Refer to Configuration for more information on setting environment variables.

Hardening task runners

Task runners are responsible for executing code from the Code node. While Code node executions are secure, you can follow these recommendations to further harden your task runners.

Run task runners as sidecars in external mode

To increase the isolation between the core n8n process and code in the Code node, run task runners in external mode. External task runners launch as separate containers, providing a fully isolated environment to execute the JavaScript defined in the Code node.

Restrict account registration to email-verified users

You can require all new accounts to be verified by email. This prevents malicious admins from registering accounts without email verification.

Prerequisites

- SMTP must be set up and n8n must be able to send emails.

How to restrict account registration

Set the environment variable `N8N_INVITE_LINKS_EMAIL_ONLY` to `true`. This locks down your instance so that only users with verified email addresses can register.

For more details on configuring SMTP, see Set up SMTP.

Self-hosted AI Starter Kit

The Self-hosted AI Starter Kit is an open, docker compose template that bootstraps a fully featured Local AI and Low Code development environment.

Curated by n8n, it combines the self-hosted n8n platform with a list of compatible AI products and components to get you started building self-hosted AI workflows.

What’s included

✅ **Self-hosted n8n**: Low-code platform with over 400 integrations and advanced AI components.

✅ **Ollama**: Cross-platform LLM platform to install and run the latest local LLMs.

✅ **Qdrant**: Open-source, high performance vector store with a comprehensive API.

✅ **PostgreSQL**: The workhorse of the Data Engineering world, handles large amounts of data safely.

What you can build

⭐ AI Agents{ data-preview} that can schedule appointments

⭐ Summaries of company PDFs without leaking data

⭐ Smarter Slackbots for company communications and IT-ops

⭐ Private, low-cost analyses of financial documents

Get the kit

Head to the GitHub repository to clone the repo and get started!

Architecture

Understanding n8n's underlying architecture is helpful if you need to:

- Embed n8n
- Customize n8n's default databases

This section is a work in progress. If you have questions, please try the forum and let n8n know which architecture documents would be useful for you.

Database structure

This page describes the purpose of each table in the n8n database.

Database and query technology

By default, n8n uses SQLite as the database. If you are using another database the structure will be similar, but the data-types may be different depending on the database.

n8n uses TypeORM for queries and migrations.

To inspect the n8n database, you can use DBeaver, which is an open-source universal database tool.

Tables

These are the tables n8n creates during setup.

auth_identity

Stores details of external authentication providers when using SAML.

auth_provider_sync_history

Stores the history of a SAML connection.

credentials_entity

Stores the credentials used to authenticate with integrations.

event_destinations

Contains the destination configurations for Log streaming.

execution_data

Contains the workflow at time of running, and the execution data.

execution_entity

Stores all saved workflow executions. Workflow settings can affect which executions n8n saves.

execution_metadata

Stores Custom executions data.

installed_nodes

Lists the community nodes installed in your n8n instance.

installed_packages

Details of npm community nodes packages installed in your n8n instance. installed_nodes lists each individual node. `installed_packages` lists npm packages, which may contain more than one node.

migrations

A log of all database migrations. Read more about Migrations in TypeORM's documentation.

project

Lists the projects in your instance.

project_relation

Describes the relationship between a user and a project, including the user's role type.

role

Not currently used. For use in future work on custom roles.

settings

Records custom instance settings. These are settings that you can't control using environment variables. They include:

- Whether the instance owner is set up
- Whether the user chose to skip owner and user management setup
- Whether certain types of authentication, including SAML and LDAP, are on
- License key

shared_credentials

Maps credentials to users.

shared_workflow

Maps workflows to users.

tag_entity

All workflow tags created in the n8n instance. This table lists the tags. workflows_tags records which workflows have which tags.

user

Contains user data.

variables

Store variables.

webhook_entity

Records the active webhooks in your n8n instance's workflows. This isn't just webhooks uses in the Webhook node. It includes all active webhooks used by any trigger node.

workflow_entity

Your n8n instance's saved workflows.

workflow_history

Store previous versions of workflows.

workflow_statistics

Counts workflow IDs and their status.

workflows_tags

Maps tags to workflows. tag_entity contains tag details.

Entity Relationship Diagram (ERD)

[Image: “n8n ERD”]

CLI commands for n8n

n8n includes a CLI (command line interface), allowing you to perform actions using the CLI rather than the n8n editor. These include starting workflows, and exporting and importing workflows and credentials.

Running CLI commands

You can use CLI commands with self-hosted n8n. Depending on how you choose to install n8n, there are differences in how to run the commands:

- npm: the `n8n` command is directly available. The documentation uses this in the examples below.
- Docker: the `n8n` command is available within your Docker container:

```
docker exec -u node -it <n8n-container-name> <n8n-cli-command>
```

Start a workflow

You can start workflows directly using the CLI.

Execute a saved workflow by its ID:

```
n8n execute --id <ID>
```

Change the active status of a workflow

You can change the active status of a workflow using the CLI.

Set the active status of a workflow by its ID to false:

```
n8n update:workflow --id=<ID> --active=false
```

Set the active status of a workflow by its ID to true:

```
n8n update:workflow --id=<ID> --active=true
```

Set the active status to false for all the workflows:

```
n8n update:workflow --all --active=false
```

Set the active status to true for all the workflows:

```
n8n update:workflow --all --active=true
```

Export entities

You can export your database entities from n8n using the CLI. This tooling allows you to export all entity types from one database type, such as SQLite, and import them into another database type, such as Postgres.

Command flags:

Flag	Description
–help	Help prompt.
–outputDir	Output directory path
–includeExecutionHistoryDataTables	Include execution history data tables, these are excluded by default as they can be very large

```
n8n export:entities --outputDir=./outputs --includeExecutionHistoryDataTables=true
```

Export workflows and credentials

You can export your workflows and credentials from n8n using the CLI.

Command flags:

Flag	Description
–help	Help prompt.
–all	Exports all workflows/credentials.
–backup	Sets –all –pretty –separate for backups. You can optionally set –output.
–id	The ID of the workflow to export.
–output	Outputs file name or directory if using separate files.
–pretty	Formats the output in an easier to read fashion.
–separate	Exports one file per workflow (useful for versioning). Must set a directory using –output.
–decrypted	Exports the credentials in a plain text format.

Workflows

Export all your workflows to the standard output (terminal):

```
n8n export:workflow --all
```

Export a workflow by its ID and specify the output file name:

```
n8n export:workflow --id=<ID> --output=file.json
```

Export all workflows to a specific directory in a single file:

```
n8n export:workflow --all --output=backups/latest/file.json
```

Export all the workflows to a specific directory using the `--backup` flag (details above):

```
n8n export:workflow --backup --output=backups/latest/
```

Credentials

Export all your credentials to the standard output (terminal):

```
n8n export:credentials --all
```

Export credentials by their ID and specify the output file name:

```
n8n export:credentials --id=<ID> --output=file.json
```

Export all credentials to a specific directory in a single file:

```
n8n export:credentials --all --output=backups/latest/file.json
```

Export all the credentials to a specific directory using the `--backup` flag (details above):

```
n8n export:credentials --backup --output=backups/latest/
```

Export all the credentials in plain text format. You can use this to migrate from one installation to another that has a different secret key in the configuration file.

```
n8n export:credentials --all --decrypted --
output=backups/decrypted.json
```

Import entities

You can import entities from a previous `export:entities` command using this command, it allows importing of entities into a database type that differs from the exported database type. Current supported database types include: SQLite, Postgres.

The database is expected to be empty prior to import, this can be forced with the `--truncateTables` parameter.

Command flags:

Flag	Description
–help	Help prompt.
–inputDir	Input directory that holds output files for import

–truncateTables	Truncate tables before import

```
n8n import:entities --inputDir ./outputs --truncateTables true
```

Import workflows and credentials

You can import your workflows and credentials from n8n using the CLI.

Available flags:

Flag	Description
–help	Help prompt.
–input	Input file name or directory if you use –separate.
–projectId	Import the workflow or credential to the specified project. Can't be used with `--userId`.
–separate	Imports `*.json` files from directory provided by –input.
–userId	Import the workflow or credential to the specified user. Can't be used with `--projectId`.

Workflows

Import workflows from a specific file:

```
n8n import:workflow --input=file.json
```

Import all the workflow files as JSON from the specified directory:

```
n8n import:workflow --separate --input=backups/latest/
```

Credentials

Import credentials from a specific file:

```
n8n import:credentials --input=file.json
```

Import all the credentials files as JSON from the specified directory:

```
n8n import:credentials --separate --input=backups/latest/
```

License

Clear

Clear your existing license from n8n's database and reset n8n to default features:

```
n8n license:clear
```

If your license includes floating entitlements, running this command will also attempt to release them back to the pool, making them available for other instances.

Info

Display information about the existing license:

```
n8n license:info
```

User management

You can reset user management using the n8n CLI. This returns user management to its pre-setup state. It removes all user accounts.

Use this if you forget your password, and don't have SMTP set up to do password resets by email.

```
n8n user-management:reset
```

Disable MFA for a user

If a user loses their recovery codes you can disable MFA for a user with this command. The user will then be able to log back in to set up MFA again.

```
n8n mfa:disable --email=johndoe@example.com
```

Disable LDAP

You can reset the LDAP settings using the command below.

```
n8n ldap:reset
```

Uninstall community nodes and credentials

You can manage community nodes using the n8n CLI. For now, you can only uninstall community nodes and credentials, which is useful if a community node causes instability.

Command flags:

Flag	Description
–help	Show CLI help.
–credential	The credential type. Get this value by visiting the node's `<NODE>.credential.ts` file and getting the value of `name`.
–package	Package name of the community node.
–uninstall	Uninstalls the node.
–userId	The ID of the user who owns the credential. On self-hosted, query the database. On cloud, query the API with your API key.

Nodes

Uninstall a community node by package name:

```
n8n community-node --uninstall --package <COMMUNITY_NODE_NAME>
```

For example, to uninstall the Evolution API community node, type:

```
n8n community-node --uninstall --package n8n-nodes-evolution-api
```

Credentials

Uninstall a community node credential:

```
n8n community-node --uninstall --credential <CREDENTIAL_TYPE> --
userId <ID>
```

For example, to uninstall the Evolution API community node credential, visit the repository and navigate to the `credentials.ts` file to find the `name`:

```
n8n community-node --uninstall --credential evolutionApi --userId
1234
```

Security audit

You can run a security audit on your n8n instance, to detect common security issues.

```
n8n audit
```

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.

  :octicons-arrow-right-24: Expressions

- **Code node**

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

  :octicons-arrow-right-24: Code node

Other technical resources

These are features that are relevant to technical users.

Technical nodes

n8n provides core nodes, which simplify adding key functionality such as API requests, webhooks, scheduling, and file handling.

- **Write a backend**

  The HTTP Request, Webhook, and Code nodes help you make API calls, respond to webhooks, and write any JavaScript in your workflow.

  Use this do things like Create an API endpoint.

  :octicons-arrow-right-24: Core nodes

- **Represent complex logic**

  You can build complex flows, using nodes like If, Switch, and Merge nodes.

  :octicons-arrow-right-24: Flow logic

Other developer resources

- **The n8n API**

  n8n provides an API, where you can programmatically perform many of the same tasks as you can in the GUI. There's an n8n API node to access the API in your workflows.

  :octicons-arrow-right-24: API

- **Self-host**

  You can self-host n8n. This keeps your data on your own infrastructure.

  :octicons-arrow-right-24: Hosting

- **Build your own nodes**

  You can build custom nodes, install them on your n8n instance, and publish them to npm.

  :octicons-arrow-right-24: Creating nodes

Expressions

Expressions are a powerful feature implemented in all n8n nodes. They allow node parameters to be set dynamically based on data from:

- Previous node executions
- The workflow
- Your n8n environment

You can also execute JavaScript within an expression, making this a convenient and easy way to manipulate data into useful parameter values without writing extensive extra code.

n8n created and uses a templating language called Tournament, and extends it with custom methods and variables and data transformation functions. These features make it easier to perform common tasks like getting data from other nodes or accessing workflow metadata.

n8n additionally supports two libraries:

- Luxon, for working with dates and time.
- JMESPath, for querying JSON.

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** Image: Open expressions editor icon 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:

```
[
  {
    "headers": {
      "host": "n8n.instance.address",
      ...
    },
    "params": {},
    "query": {},
    "body": {
      "name": "Jim",
      "age": 30,
      "city": "New York"
    }
  }
]
```

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}}
```

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

You can do things like variable assignments or multiple statements in an expression, but you need to wrap your code using the syntax for an IIFE (Immediately Invoked Function Expression).

The following code use the Luxon date and time library to find the time between two dates in months. We surround the code in both the handlebar brackets for an expression and the IIFE syntax.

```
{{(()=>{
  let end = DateTime.fromISO('2017-03-13');
  let start = DateTime.fromISO('2017-02-13');
  let diffInMonths = end.diff(start, 'months');
  return diffInMonths.toObject();
})()}}
```

Common issues

For common errors or issues with expressions and suggested resolution steps, refer to Common Issues.

Using the Code node

–8<– “_snippets/integrations/builtin/core-nodes/code-node.md”

AI coding with GPT

Not available on self-hosted.

Python isn’t supported. ///

Use AI in the Code node

–8<– “_snippets/code/ai-how-to.md”

Usage limits

During the trial phase there are no usage limits. If n8n makes the feature permanent, there may be usage limits as part of your pricing tier.

Feature limits

The ChatGPT implementation in n8n has the following limitations:

- The AI writes code that manipulates data from the n8n workflow. You can’t ask it to pull in data from other sources.
- The AI doesn’t know your data, just the schema, so you need to tell it things like how to find the data you want to extract, or how to

check for null.
- Nodes before the Code node must execute and deliver data to the Code node before you run your AI query.
- Doesn't work with large incoming data schemas.
- May have issues if there are a lot of nodes before the code node.

Writing good prompts

Writing good prompts increases the chance of getting useful code back.

Some general tips:

- Provide examples: if possible, give a sample expected output. This helps the AI to better understand the transformation or logic you're aiming for.
- Describe the processing steps: if there are specific processing steps or logic that should apply to the data, list them in sequence. For example: "First, filter out all users under 18. Then, sort the remaining users by their last name."
- Avoid ambiguities: while the AI understands various instructions, being clear and direct ensures you get the most accurate code. Instead of saying "Get the older users," you might say "Filter users who are 60 years and above."
- Be clear about what you expect as the output. Do you want the data transformed, filtered, aggregated, or sorted? Provide as much detail as possible.

And some n8n-specific guidance:

- Think about the input data: make sure ChatGPT knows which pieces of the data you want to access, and what the incoming data represents. You may need to tell ChatGPT about the availability of n8n's built-in methods and variables.
- Declare interactions between nodes: if your logic involves data from multiple nodes, specify how they should interact. "Merge the output of 'Node A' with 'Node B' based on the 'userID' property". if you prefer data to come from certain nodes or to ignore others, be clear: "Only consider data from the 'Purchases' node and ignore the 'Refunds' node."
- Ensure the output is compatible with n8n. Refer to Data structure for more information on the data structure n8n requires.

Example prompts

These examples show a range of possible prompts and tasks.

Example 1: Find a piece of data inside a second dataset

To try the example yourself, download the example workflow and import it into n8n.

In the third Code node, enter this prompt:

> The slack data contains only one item. The input data represents all Notion users. Sometimes the person property that holds the email can be null. I want to find the notionId of the Slack user and return it.

Take a look at the code the AI generates.

This is the JavaScript you need:

```
const slackUser = $("Mock Slack").all()[0];
const notionUsers = $input.all();
const slackUserEmail = slackUser.json.email;

const notionUser = notionUsers.find(
  (user) => user.json.person && user.json.person.email ===
slackUserEmail
);

return notionUser ? [{ json: { notionId: notionUser.json.id } }] :
[];
```

Example 2: Data transformation

To try the example yourself, download the example workflow and import it into n8n.

In the **Join items** Code node, enter this prompt:

> Return a single line of text that has all usernames listed with a comma. Each username should be enquoted with a double quotation mark.

Take a look at the code the AI generates.

This is the JavaScript you need:

```
const items = $input.all();
const usernames = items.map((item) => `"${item.json.username}"`);
const result = usernames.join(", ");
return [{ json: { usernames: result } }];
```

Example 3: Summarize data and create a Slack message

To try the example yourself, download the example workflow and import it into n8n.

In the **Summarize** Code node, enter this prompt:

> Create a markdown text for Slack that counts how many ideas, features and bugs have been submitted. The type of submission is saved in the property_type field. A feature has the property “Feature”, a bug has the property “Bug” and an idea has the property “Bug”. Also, list the five top submissions by vote in that message. Use “<url|text>” as markdown for links.

Take a look at the code the AI generates.

This is the JavaScript you need:

```
const submissions = $input.all();

// Count the number of ideas, features, and bugs
let ideaCount = 0;
let featureCount = 0;
let bugCount = 0;
```

```
        submissions.forEach((submission) => {
          switch (submission.json.property_type[0]) {
            case "Idea":
              ideaCount++;
              break;
            case "Feature":
              featureCount++;
              break;
            case "Bug":
              bugCount++;
              break;
          }
        });

        // Sort submissions by votes and take the top 5
        const topSubmissions = submissions
          .sort((a, b) => b.json.property_votes - a.json.property_votes)
          .slice(0, 5);

        let topSubmissionText = "";
        topSubmissions.forEach((submission) => {
          topSubmissionText +=
`<${submission.json.url}|${submission.json.name}> with
${submission.json.property_votes} votes\n`;
        });

        // Construct the Slack message
        const slackMessage = `*Summary of Submissions*\n
        Ideas: ${ideaCount}\n
        Features: ${featureCount}\n
        Bugs: ${bugCount}\n
        Top 5 Submissions:\n
        ${topSubmissionText}`;

        return [{ json: { slackMessage } }];
```

Reference incoming node data explicitly

If your incoming data contains nested fields, using dot notation to reference them can help the AI understand what data you want.

[Image: “Screenshot of an n8n code node, highlighting how to reference data with dot notation in an AI query”]

To try the example yourself, download the example workflow and import it into n8n.

In the second Code node, enter this prompt:

> The data in “Mock data” represents a list of people. For each person, return a new item containing personal_info.first_name and work_info.job_title.

This is the JavaScript you need:

```
        const items = $input.all();
        const newItems = items.map((item) => {
          const firstName = item.json.personal_info.first_name;
          const jobTitle = item.json.work_info.job_title;
          return {
            json: {
```

```
        firstName,
        jobTitle,
      },
    };
});
return newItems;
```

Related resources

Pluralsight offer a short guide on How to use ChatGPT to write code, which includes example prompts.

Fixing the code

The AI-generated code may work without any changes, but you may have to edit it. You need to be aware of n8n's Data structure. You may also find n8n's built-in methods and variables useful.

Built-in methods and variables

n8n provides built-in methods and variables for working with data and accessing n8n data. This section provides a reference of available methods and variables for use in expressions, with a short description.

The Cookbook contains examples for some common tasks, including some Code node only functions.

Current node input

Methods for working with the input of the current node. Some methods and variables aren't available in the Code node.

=== "JavaScript" | Method | Description | Available in Code node? | | —— | ———– | :————————-: | | `$binary` | Shorthand for `$input.item.binary`. Incoming binary data from a node | :x: | | `$input.item` | The input item of the current node that's being processed. Refer to Item linking for more information on paired items and item linking. | :white_check_mark: | | `$input.all()` | All input items in current node. | :white_check_mark: | | `$input.first()` | First input item in current node. | :white_check_mark: | | `$input.last()` | Last input item in current node. | :white_check_mark: | | `$input.params` | Object containing the query settings of the previous node. This includes data such as the operation it ran, result limits, and so on. | :white_check_mark: | | `$json` | Shorthand for `$input.item.json`. Incoming JSON data from a node. Refer to Data structure for information on item structure. | :white_check_mark: (when running once for each item) | | `$input.context.noItemsLeft` | Boolean. Only available when working with the Loop Over Items node. Provides information about what's happening in the node. Use this to determine whether the node is still processing items. | :white_check_mark: | === "Python" | Method | Description | | —— | ———– | | `_input.item` | The input item of the current node that's being processed. Refer to Item linking for more information on paired items

and item linking. | | `_input.all()` | All input items in current node. | | `_input.first()` | First input item in current node. | | `_input.last()` | Last input item in current node. | | `_input.params` | Object containing the query settings of the previous node. This includes data such as the operation it ran, result limits, and so on. | | `_json` | Shorthand for `_input.item.json`. Incoming JSON data from a node. Refer to Data structure for information on item structure. Available when you set **Mode** to **Run Once for Each Item**. | | `_input.context.noItemsLeft` | Boolean. Only available when working with the Loop Over Items node. Provides information about what's happening in the node. Use this to determine whether the node is still processing items. |

Output of other nodes

Methods for working with the output of other nodes. Some methods and variables aren't available in the Code node.

=== "JavaScript" | Method | Description | Available in Code node? | | —— | ———– | :————————-: | | `$("<node-name>").all(branchIndex?, runIndex?)` | Returns all items from a given node. If `branchIndex` isn't given it will default to the output that connects `node-name` with the node where you use the expression or code. | :white_check_mark: | | `$("<node-name>").first(branchIndex?, runIndex?)` | The first item output by the given node. If `branchIndex` isn't given it will default to the output that connects `node-name` with the node where you use the expression or code. | :white_check_mark: | | `$("<node-name>").last(branchIndex?, runIndex?)` | The last item output by the given node. If `branchIndex` isn't given it will default to the output that connects `node-name` with the node where you use the expression or code. | :white_check_mark: | | `$("<node-name>").item` | The linked item. This is the item in the specified node used to produce the current item. Refer to Item linking for more information on item linking. | :white_check_mark: | | `$("<node-name>").params` | Object containing the query settings of the given node. This includes data such as the operation it ran, result limits, and so on. | :white_check_mark: | | `$("<node-name>").context` | Boolean. Only available when working with the Loop Over Items node. Provides information about what's happening in the node. Use this to determine whether the node is still processing items. | :white_check_mark: | | `$("<node-name>").itemMatching(currentNodeInputIndex)` | Use instead of `$("<node-name>").item` in the Code node if you need to trace back from an input item. | :white_check_mark: | === "Python" | Method | Description | Available in Code node? | | —— | ———– | :————————-: | | `_("<node-name>").all(branchIndex?, runIndex?)` | Returns all items from a given node. If `branchIndex` isn't given it will default to the output that connects`node-name` with the node where you use the expression or code. | :white_check_mark: | | `_("<node-name>").first(branchIndex?, runIndex?)` | The first item output by the given node. If `branchIndex` isn't given it will default to the output that connects`node-name` with the node where you use the expression or code. | :white_check_mark: | | `_("<node-name>").last(branchIndex?, runIndex?)` | The last item output by the given node. If `branchIndex` isn't given it will default to the output that connects`node-name` with the node where you use the expression or code. | :white_check_mark: | | `_("<node-name>").item` | The linked item. This is the item in the specified node used to produce the current item. Refer to Item linking for more information on item linking. | :white_check_mark: | | `_("<node-name>").params` | Object containing the query settings of the given node. This includes data

such as the operation it ran, result limits, and so on. | :white_check_mark: | | `_("<node-name>").context` | Boolean. Only available when working with the Loop Over Items node. Provides information about what's happening in the node. Use this to determine whether the node is still processing items. | :white_check_mark: | | `_("<node-name>").itemMatching(currentNodeInputIndex)` | Use instead of `_("<node-name>").item` in the Code node if you need to trace back from an input item. Refer to Retrieve linked items from earlier in the workflow for an example. | :white_check_mark: |

Built-in date and time methods

Methods for working with date and time.

=== "JavaScript" | Method | Description | Available in Code node? | | —— | ———– | :————————-: | | `$now` | A Luxon object containing the current timestamp. Equivalent to `DateTime.now()`. | :white_check_mark: | | `$today` | A Luxon object containing the current timestamp, rounded down to the day. Equivalent to `DateTime.now().set({ hour: 0, minute: 0, second: 0, millisecond: 0 })`. | :white_check_mark: | === "Python" | Method | Description | | —— | ———– | | `_now` | A Luxon object containing the current timestamp. Equivalent to `DateTime.now()`. | | `_today` | A Luxon object containing the current timestamp, rounded down to the day. Equivalent to `DateTime.now().set({ hour: 0, minute: 0, second: 0, millisecond: 0 })`. |

n8n provides built-in convenience functions to support data transformation in expressions for dates. Refer to Data transformation functions | Dates for more information.

JMESPath method

This is an n8n-provided method for working with the JMESPath library.

=== "JavaScript" | Method | Description | Available in Code node? | | —— | ———– | :————————-: | | `$jmespath()` | Perform a search on a JSON object using JMESPath. | :white_check_mark: | === "Python" | Method | Description | | —— | ———– | | `_jmespath()` | Perform a search on a JSON object using JMESPath. |

HTTP node variables

Variables for working with HTTP node requests and responses when using pagination.

Refer to HTTP Request for guidance on using the HTTP node, including configuring pagination.

Refer to HTTP Request node cookbook | Pagination for example pagination configurations.

–8<– “_snippets/integrations/builtin/core-nodes/http/pagination-variables.md”

LangChain Code node methods

n8n provides these methods to make it easier to perform common tasks in the LangChain Code node.

–8<– “_snippets/integrations/builtin/cluster-nodes/langchain-root-nodes/langchaincode/builtin-methods.md”

---

n8n metadata

Methods for working with n8n metadata.

This includes:

- Access to n8n environment variables for self-hosted n8n.
- Metadata about workflows, executions, and nodes.
- Information about instance Variables and External secrets.

=== “JavaScript” | Method | Description | Available in Code node? | | —— | ———– | :————————-: | | `$env` | Contains n8n instance configuration environment variables. | :white_check_mark: | | `$execution.customData` | Set and get custom execution data. Refer to Custom executions data for more information. | :white_check_mark: | | `$execution.id` | The unique ID of the current workflow execution. | :white_check_mark: | | `$execution.mode` | Whether the execution was triggered automatically, or by manually running the workflow. Possible values are `test` and `production`. | :white_check_mark: | | `$execution.resumeUrl` | The webhook URL to call to resume a workflow waiting at a Wait node. | :white_check_mark: | | `$getWorkflowStaticData(type)` | View an example. Static data doesn’t persist when testing workflows. The workflow must be active and called by a trigger or webhook to save static data. This gives access to the static workflow data. | :white_check_mark: | | `$("<node-name>").isExecuted` | Check whether a node has already executed. | :white_check_mark: | | `$itemIndex` | The index of an item in a list of items. | :x: | | `$nodeVersion` | Get the version of the current node. | :white_check_mark: | | `$prevNode.name` | The name of the node that the current input came from. When using the Merge node, note that `$prevNode` always uses the first input connector. | :white_check_mark: | | `$prevNode.outputIndex` | The index of the output connector that the current input came from. Use this when the previous node had multiple outputs (such as an If or Switch node). When using the Merge node, note that `$prevNode` always uses the first input connector. | :white_check_mark: | | `$prevNode.runIndex` | The run of the previous node that generated the current input. When using the Merge node, note that `$prevNode` always uses the first input connector. | :white_check_mark: | | `$runIndex` | How many times n8n has executed the current node. Zero-based (the first run is 0, the second is 1, and so on). | :white_check_mark: | | `$secrets` | Contains information about your External secrets setup. | :x: | | `$vars` | Contains the Variables available in the active environment. | :white_check_mark: | | `$version` | The node version. | :x: | | `$workflow.active` | Whether the workflow is

active (true) or not (false). | :white_check_mark: | | `$workflow.id` | The workflow ID. | :white_check_mark: | | `$workflow.name` | The workflow name. | :white_check_mark: | === “Python (native)” | Method | Description | | —— | ———– | | `_items` | Contains incoming items in “Run once for all items” mode. | | `_item` | Contains the item being iterated on in “Run once for each item” mode. | === “Python (Pyodide, deprecated)” | Method | Description | | —— | ———– | | `_env` | Contains n8n instance configuration environment variables. | | `_execution.customData` | Set and get custom execution data. Refer to Custom executions data for more information. | | `_execution.id` | The unique ID of the current workflow execution. | | `_execution.mode` | Whether the execution was triggered automatically, or by manually running the workflow. Possible values are `test` and `production`. | | `_execution.resumeUrl` | The webhook URL to call to resume a workflow waiting at a Wait node. | | `_getWorkflowStaticData(type)` | View an example. Static data doesn’t persist when testing workflows. The workflow must be active and called by a trigger or webhook to save static data. This gives access to the static workflow data. | | `_("<node-name>").isExecuted` | Check whether a node has already executed. | | `_nodeVersion` | Get the version of the current node. | :white_check_mark: | | `_prevNode.name` | The name of the node that the current input came from. When using the Merge node, note that `_prevNode` always uses the first input connector. | | `_prevNode.outputIndex` | The index of the output connector that the current input came from. Use this when the previous node had multiple outputs (such as an If or Switch node). When using the Merge node, note that `_prevNode` always uses the first input connector. | | `_prevNode.runIndex` | The run of the previous node that generated the current input. When using the Merge node, note that `_prevNode` always uses the first input connector. | | `_runIndex` | How many times n8n has executed the current node. Zero-based (the first run is 0, the second is 1, and so on). | | `_secrets` | Contains information about your External secrets setup. | | `_vars` | Contains the Variables available in the active environment. | | `_workflow.active` | Whether the workflow is active (true) or not (false). | | `_workflow.id` | The workflow ID. | | `_workflow.name` | The workflow name. |

Convenience methods

n8n provides these methods to make it easier to perform common tasks in expressions.

=== “JavaScript” | Method | Description | Available in Code node? | | —— | ———– | :———————: | | `$evaluateExpression(expression: string, itemIndex?: number)` | Evaluates a string as an expression. If you don’t provide `itemIndex`, n8n uses the data from item 0 in the Code node. | :white_check_mark: | | `$ifEmpty(value, defaultValue)` | The `$ifEmpty()` function takes two parameters, tests the first to check if it’s empty, then returns either the parameter (if not empty) or the second parameter (if the first is empty). The first parameter is empty if it’s:

- `undefined`
- `null`
- An empty string `''`
- An array where `value.length` returns `false`
- An object where `Object.keys(value).length` returns `false`

| :white_check_mark: | | `$if()` | The `$if()` function takes three

parameters: a condition, the value to return if true, and the value to return if false. | :x: | | `$max()` | Returns the highest of the provided numbers. | :x: | | `$min()` | Returns the lowest of the provided numbers. | :x: | === “Python” | Method | Description | | —— | ——–- | | `_evaluateExpression(expression: string, itemIndex?: number)` | Evaluates a string as an expression. If you don’t provide `itemIndex`, n8n uses the data from item 0 in the Code node. | | `_ifEmpty(value, defaultValue)` | The `_ifEmpty()` function takes two parameters, tests the first to check if it’s empty, then returns either the parameter (if not empty) or the second parameter (if the first is empty). The first parameter is empty if it’s:

- `undefined`
- `null`
- An empty string `''`
- An array where `value.length` returns `false`
- An object where `Object.keys(value).length` returns `false`

| :white_check_mark: |

Data transformation functions

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

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() }}
```

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

```
{{ "example@example.com".isEmail() }}

// Returns true
```

Arrays

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

Booleans

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

Dates

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

Numbers

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

Objects

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

Strings

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

Custom variables

Custom variables are read-only variables that you can use to store and reuse values in n8n workflows.

Create variables

You can access the **Variables** tab from either the overview page or a specific project.

To create a new variable:

1. On the **Variables** tab, select **Add Variable**.
2. Enter a **Key** and **Value**. The maximum key length is 50 characters, and the maximum value length is 1000 characters. n8n limits the characters you can use in the key and value to lowercase and uppercase letters, numbers, and underscores (`A-Z`, `a-z`, `0-9`, `_`).
3. Select the **Scope** (only available when creating from the overview page):

    - **Global**: The variable is available across all projects in the n8n instance.
    - **Project**: The variable is available only within a specific project (you can select which project).
    - When creating from a project page, the scope is automatically set to that project.
4. Select **Save**. The variable is now available for use in workflows according to its scope.

Edit and delete variables

To edit or delete a variable:

1. On the **Variables** tab, hover over the variable you want to change.
2. Select **Edit** or **Delete**.

Use variables in workflows

You can access variables in the Code node and in expressions:

```
// Access a variable
$vars.<variable-name>
```

All variables are strings.

During workflow execution, n8n replaces the variables with the variable value. If the variable has no value, n8n treats its value as `undefined`. Workflows don't automatically fail in this case.

Variables are read-only. You must use the UI to change the values. If you need to set and access custom data within your workflow, use Workflow static data.

Date and time with Luxon

Luxon is a JavaScript library that makes it easier to work with date and time. For full details of how to use Luxon, refer to Luxon's documentation.

n8n passes dates between nodes as strings, so you need to parse them. Luxon makes this easier.

Date and time behavior in n8n

Be aware of the following:

- In a workflow, n8n converts dates and times to strings between nodes. Keep this in mind when doing arithmetic on dates and times from other nodes.
- With vanilla JavaScript, you can convert a string to a date with `new Date('2019-06-23')`. In Luxon, you must use a function explicitly stating the format, such as `DateTime.fromISO('2019-06-23')` or `DateTime.fromFormat("23-06-2019", "dd-MM-yyyy")`.

Setting the timezone in n8n

Luxon uses the n8n timezone. This value is either:

- Default: `America/New York`
- A custom timezone for your n8n instance, set using the `GENERIC_TIMEZONE` environment variable.
- A custom timezone for an individual workflow, configured in workflow settings.

Common tasks

This section provides examples for some common operations. More examples, and detailed guidance, are available in Luxon's own documentation.

Get the current datetime or date

Use the `$now` and `$today` Luxon objects to get the current time or day:

- `now`: a Luxon object containing the current timestamp. Equivalent to `DateTime.now()`.
- `today`: a Luxon object containing the current timestamp, rounded down to the day. Equivalent to `DateTime.now().set({ hour: 0, minute: 0, second: 0, millisecond: 0 })`.

Note that these variables can return different time formats when cast as a string:

=== "Expressions (JavaScript)" `javascript {{$now}} // n8n displays the ISO formatted timestamp // For example 2022-03-09T14:02:37.065+00:00 {{"Today's date is " + $now}} // n8n displays "Today's date is <unix timestamp>" // For example "Today's date is 1646834498755"`

=== "Code node (JavaScript)" `javascript $now // n8n displays <ISO formatted timestamp> // For example 2022-03-09T14:00:25.058+00:00 let rightNow = "Today's date is " + $now // n8n displays "Today's date is <unix timestamp>" // For example "Today's date is 1646834498755"` === "Code node (Python)" `python _now # n8n displays <ISO formatted timestamp> # For example 2022-03-09T14:00:25.058+00:00 rightNow = "Today's date is " + str(_now) # n8n displays "Today's date is <unix timestamp>" # For example "Today's date is 1646834498755"`

n8n provides built-in convenience functions to support data transformation in expressions for dates. Refer to Data transformation functions | Dates for more information.

Convert JavaScript dates to Luxon

To convert a native JavaScript date to a Luxon date:

- In expressions, use the `.toDateTime()` method. For example, `{{ (new Date()).ToDateTime() }}`.
- In the Code node, use `DateTime.fromJSDate()`. For example, `let luxondate = DateTime.fromJSDate(new Date())`.

Convert date string to Luxon

You can convert date strings and other date formats to a Luxon DateTime object. You can convert from standard formats and from arbitrary strings.

If you have a date in a supported standard technical format:

Most dates use `fromISO()`. This creates a Luxon DateTime from an ISO 8601 string. For example:

=== “Expressions (JavaScript)”

```
```js
{{DateTime.fromISO('2019-06-23T00:00:00.00')}}
```
```

=== “Code node (JavaScript)”

```
```js
let luxonDateTime = DateTime.fromISO('2019-06-23T00:00:00.00')
```
```

Luxon’s API documentation has more information on fromISO.

Luxon provides functions to handle conversions for a range of formats. Refer to Luxon’s guide to Parsing technical formats for details.

If you have a date as a string that doesn’t use a standard format:

Use Luxon’s Ad-hoc parsing. To do this, use the `fromFormat()` function, providing the string and a set of tokens that describe the format.

For example, you have n8n’s founding date, 23rd June 2019, formatted as `23-06-2019`. You want to turn this into a Luxon object:

=== “Expressions (JavaScript)”

```
```js
{{DateTime.fromFormat("23-06-2019", "dd-MM-yyyy")}}
```
```

=== “Code node (JavaScript)”

```
```js
let newFormat = DateTime.fromFormat("23-06-2019", "dd-MM-yyyy")
```
```

When using ad-hoc parsing, note Luxon’s warning about Limitations. If you see unexpected results, try their Debugging guide.

Get n days from today

Get a number of days before or after today.

=== “Expressions (JavaScript)”

```
For example, you want to set a field to always show the date seven
days before the current date.
```

```
In the expressions editor, enter:


``` js
{{$today.minus({days: 7})}}
```

On the 23rd June 2019, this returns `[Object: "2019-06-
16T00:00:00.000+00:00"]`.

This example uses n8n's custom variable `$today` for convenience.
It's the equivalent of `DateTime.now().set({ hour: 0, minute: 0,
second: 0, millisecond: 0 }).minus({days: 7})`.

=== "Code node (JavaScript)"

For example, you want a variable containing the date seven days
before the current date.

In the code editor, enter:

``` js
let sevenDaysAgo = $today.minus({days: 7})
```

On the 23rd June 2019, this returns `[Object: "2019-06-
16T00:00:00.000+00:00"]`.

This example uses n8n's custom variable `$today` for convenience.
It's the equivalent of `DateTime.now().set({ hour: 0, minute: 0,
second: 0, millisecond: 0 }).minus({days: 7})`.
```

For more detailed information and examples, refer to:

- Luxon's guide to math
- Their API documentation on DateTime plus and DateTime minus

Create human-readable dates

In Get n days from today, the example gets the date seven days before the current date, and returns it as `[Object: "yyyy-mm-dd-T00:00:00.000+00:00"]` (for expressions) or `yyyy-mm-dd-T00:00:00.000+00:00` (in the Code node). To make this more readable, you can use Luxon's formatting functions.

For example, you want the field containing the date to be formatted as DD/MM/YYYY, so that on the 23rd June 2019, it returns `23/06/2019`.

This expression gets the date seven days before today, and converts it to the DD/MM/YYYY format.

=== "Expressions (JavaScript)"

```
```js
{{$today.minus({days: 7}).toLocaleString()}}
```
```

=== "Code node (JavaScript)"

```
```js
let readableSevenDaysAgo = $today.minus({days: 7}).toLocaleString()
```
```

```

You can alter the format. For example:

=== “Expressions (JavaScript)”

```js
{{$today.minus({days: 7}).toLocaleString({month: 'long', day:
'numeric', year: 'numeric'})}}
```

```
On 23rd June 2019, this returns "16 June 2019".
```

=== “Code node (JavaScript)”

```js
let readableSevenDaysAgo = $today.minus({days:
7}).toLocaleString({month: 'long', day: 'numeric', year: 'numeric'})
```

```
On 23rd June 2019, this returns "16 June 2019".
```

Refer to Luxon’s guide on toLocaleString (strings for humans) for more information.

Get the time between two dates

To get the time between two dates, use Luxon’s diffs feature. This subtracts one date from another and returns a duration.

For example, get the number of months between two dates:

=== “Expressions (JavaScript)”

```js
{{DateTime.fromISO('2019-06-23').diff(DateTime.fromISO('2019-05-
23'), 'months').toObject()}}
```

```
This returns `[Object: {"months":1}]`.
```

=== “Code node (JavaScript)”

```js
let monthsBetweenDates = DateTime.fromISO('2019-06-
23').diff(DateTime.fromISO('2019-05-23'), 'months').toObject()
```

```
This returns `{"months":1}`.
```

Refer to Luxon’s Diffs for more information.

A longer example: How many days to Christmas?

This example brings together several Luxon features, uses JMESPath, and does some basic string manipulation.

The scenario: you want a countdown to 25th December. Every day, it should tell you the number of days remaining to Christmas. You don’t want to update it for next year - it needs to seamlessly work for every year.
```

=== “Expressions (JavaScript)”

```js
{{"There are " + $today.diff(DateTime.fromISO($today.year + '-12-25'), 'days').toObject().days.toString().substring(1) + " days to Christmas!"}}
```

This outputs `"There are <number of days> days to Christmas!"`. For example, on 9th March, it outputs "There are 291 days to Christmas!".

A detailed explanation of what the expression does:

* `{{`: indicates the start of the expression.
* `"There are "`: a string.
* `+`: used to join two strings.
* `$today.diff()`: This is similar to the example in [Get the time between two dates](#get-the-time-between-two-dates), but it uses n8n's custom `$today` variable.
* `DateTime.fromISO($today.year + '-12-25'), 'days'`: this part gets the current year using `$today.year`, turns it into an ISO string along with the month and date, and then takes the whole ISO string and converts it to a Luxon DateTime data structure. It also tells Luxon that you want the duration in days.
* `toObject()` turns the result of diff() into a more usable object. At this point, the expression returns `[Object: {"days":-<number-of-days>}]`. For example, on 9th March, `[Object: {"days":-291}]`.
* `.days` uses JMESPath syntax to retrieve just the number of days from the object. For more information on using JMESPath with n8n, refer to our [JMESpath](/code/cookbook/jmespath.md) documentation. This gives you the number of days to Christmas, as a negative number.
* `.toString().substring(1)` turns the number into a string and removes the `-`.
* `+ " days to Christmas!"`: another string, with a `+` to join it to the previous string.
* `}}`: indicates the end of the expression.

=== “Code node (JavaScript)”

```js
let daysToChristmas = "There are " + $today.diff(DateTime.fromISO($today.year + '-12-25'), 'days').toObject().days.toString().substring(1) + " days to Christmas!";
```

This outputs `"There are <number of days> days to Christmas!"`. For example, on 9th March, it outputs "There are 291 days to Christmas!".

A detailed explanation of what the code does:

* `"There are "`: a string.
* `+`: used to join two strings.
* `$today.diff()`: This is similar to the example in [Get the time between two dates](#get-the-time-between-two-dates), but it uses n8n's custom `$today` variable.
* `DateTime.fromISO($today.year + '-12-25'), 'days'`: this part gets the current year using `$today.year`, turns it into an ISO string

```
along with the month and date, and then takes the whole ISO string
and converts it to a Luxon DateTime data structure. It also tells
Luxon that you want the duration in days.
* `toObject()` turns the result of diff() into a more usable object.
At this point, the expression returns `[Object: {"days":-<number-of-
days>}]`. For example, on 9th March, `[Object: {"days":-291}]`.
* `.days` uses JMESPath syntax to retrieve just the number of days
from the object. For more information on using JMESPath with n8n,
refer to our [JMESpath](/code/cookbook/jmespath.md) documentation.
This gives you the number of days to Christmas, as a negative
number.
* `.toString().substring(1)` turns the number into a string and
removes the `-`.
* `+ " days to Christmas!"`: another string, with a `+` to join it
to the previous string.
```

Query JSON with JMESPath

JMESPath is a query language for JSON that you can use to extract and transform elements from a JSON document. For full details of how to use JMESPath, refer to the JMESPath documentation.

The `jmespath()` method

n8n provides a custom method, `jmespath()`. Use this method to perform a search on a JSON object using the JMESPath query language.

The basic syntax is:

=== "JavaScript" `js $jmespath(object, searchString)` === "Python" `python _jmespath(object, searchString)`

To help understand what the method does, here is the equivalent longer JavaScript:

```
var jmespath = require('jmespath');
jmespath.search(object, searchString);
```

`object` is a JSON object, such as the output of a previous node. `searchString` is an expression written in the JMESPath query language. The JMESPath Specification provides a list of supported expressions, while their Tutorial and Examples provide interactive examples.

Common tasks

This section provides examples for some common operations. More examples, and detailed guidance, are available in JMESPath's own documentation.

When trying out these examples, you need to set the Code node **Mode** to **Run Once for Each Item**.

Apply a JMESPath expression to a collection of elements with projections

From the JMESPath projections documentation:

> Projections are one of the key features of JMESPath. Use it to apply an expression to a collection of elements. JMESPath supports five kinds of projections:
>
> - List Projections
> - Slice Projections
> - Object Projections
> - Flatten Projections
> - Filter Projections

The following example shows basic usage of list, slice, and object projections. Refer to the JMESPath projections documentation for detailed explanations of each projection type, and more examples.

Given this JSON from a webhook node:


```
[
  {
    "headers": {
      "host": "n8n.instance.address",
      ...
    },
    "params": {},
    "query": {},
    "body": {
      "people": [
        {
          "first": "James",
          "last": "Green"
        },
        {
          "first": "Jacob",
          "last": "Jones"
        },
        {
          "first": "Jayden",
          "last": "Smith"
        }
      ],
      "dogs": {
        "Fido": {
          "color": "brown",
          "age": 7
        },
        "Spot": {
          "color": "black and white",
          "age": 5
        }
      }
    }
  }
]
```


Retrieve a list of all the people's first names:

=== "Expressions (JavaScript)"

```
```js
{{$jmespath($json.body.people, "[*].first" )}}
// Returns ["James", "Jacob", "Jayden"]
```
```

```

=== “Code node (JavaScript)”

```js
let firstNames = $jmespath($json.body.people, "[*].first" )
return {firstNames};
/* Returns:
[
    {
        "firstNames": [
            "James",
            "Jacob",
            "Jayden"
        ]
    }
]
*/
```

=== “Code node (Python)” python firstNames = _jmespath(_json.body.people, "[*].first") return {"firstNames":firstNames} """ Returns: [{ "firstNames": ["James", "Jacob", "Jayden"] }] """

Get a slice of the first names:

=== “Expressions (JavaScript)”

```js
{{$jmespath($json.body.people, "[:2].first")}}
// Returns ["James", "Jacob"]
```

=== “Code node (JavaScript)” js let firstTwoNames = $jmespath($json.body.people, "[:2].first"); return {firstTwoNames}; /* Returns: [{ "firstNames": ["James", "Jacob", "Jayden"] }] */ === “Code node (Python)” python firstTwoNames = _jmespath(_json.body.people, "[:2].first") return {"firstTwoNames":firstTwoNames} """ Returns: [{ "firstTwoNames": ["James", "Jacob"] }] """

Get a list of the dogs’ ages using object projections:

=== “Expressions (JavaScript)”

```js
{{$jmespath($json.body.dogs, "*.age")}}
// Returns [7,5]
```

=== “Code node (JavaScript)” js let dogsAges = $jmespath($json.body.dogs, "*.age"); return {dogsAges}; /* Returns: [{ "dogsAges": [7, 5] }] */ === “Code node (Python)” python dogsAges = _jmespath(_json.body.dogs, "*.age") return {"dogsAges":
```

```
dogsAges}     """     Returns:     [         {
"dogsAges": [                 7,                 5            ]
}     ]     """
```

Select multiple elements and create a new list or object

Use Multiselect to select elements from a JSON object and combine them into a new list or object.

Given this JSON from a webhook node:

```
        [
          {
            "headers": {
              "host": "n8n.instance.address",
              ...
            },
            "params": {},
            "query": {},
            "body": {
              "people": [
                {
                  "first": "James",
                  "last": "Green"
                },
                {
                  "first": "Jacob",
                  "last": "Jones"
                },
                {
                  "first": "Jayden",
                  "last": "Smith"
                }
              ],
              "dogs": {
                "Fido": {
                  "color": "brown",
                  "age": 7
                },
                "Spot": {
                  "color": "black and white",
                  "age": 5
                }
              }
            }
          }
        ]
```

Use multiselect list to get the first and last names and create new lists containing both names:

=== “Expressions (JavaScript)”

```
```js
{{$jmespath($json.body.people, "[].[first, last]")}}
// Returns
```
```

=== “Code node (JavaScript)”

```js
let newList = $jmespath($json.body.people, "[].[first, last]");
return {newList};
/* Returns:
[
    {
        "newList": [
            [
                "James",
                "Green"
            ],
            [
                "Jacob",
                "Jones"
            ],
            [
                "Jayden",
                "Smith"
            ]
        ]
    }
]
*/
```

=== “Code node (Python)” `python newList = _jmespath(_json.body.people, "[].[first, last]") return {"newList":newList} """ Returns: [ { "newList": [ [ "James", "Green" ], [ "Jacob", "Jones" ], [ "Jayden", "Smith" ] ] } ] """`

An alternative to arrow functions in expressions

For example, generate some input data by returning the below code from the Code node:

```
return[
  {
    "json": {
      "num_categories": "0",
      "num_products": "45",
      "category_id": 5529735,
      "parent_id": 1407340,
      "pos_enabled": 1,
      "pos_favorite": 0,
      "name": "HP",
      "description": "",
      "image": ""
    }
  },
  {
    "json": {
      "num_categories": "0",
      "num_products": "86",
      "category_id": 5529740,
      "parent_id": 1407340,
      "pos_enabled": 1,
```

```
            "pos_favorite": 0,
            "name": "Lenovo",
            "description": "",
            "image": ""
          }
        }
      ]
```

You could do a search like “find the item with the name Lenovo and tell me their category ID.”

```
      {{ $jmespath($("Code").all(), "[?
json.name=='Lenovo'].json.category_id") }}
```

Examples using n8n’s built-in methods and variables

n8n provides built-in methods and variables for working with data and accessing n8n data. This section provides usage examples.

Related resources

- Built-in methods and variables reference
- Expressions
- Code node

---

execution

execution.id

Contains the unique ID of the current workflow execution.

=== “JavaScript” `js let executionId = $execution.id;` === “Python” `python executionId = _execution.id`

execution.resumeUrl

The webhook URL to call to resume a waiting workflow.

See the Wait > On webhook call documentation to learn more.

`execution.resumeUrl` is available in workflows containing a Wait node, along with a node that waits for a webhook response.

execution.customData

This is only available in the Code node.

=== “JavaScript” ```js // Set a single piece of custom execution data $execution.customData.set(“key”, “value”);

```
// Set the custom execution data object
$execution.customData.setAll({"key1": "value1", "key2": "value2"})

// Access the current state of the object during the execution
var customData = $execution.customData.getAll()

// Access a specific value set during this execution
var customData = $execution.customData.get("key")
```

=== “Python” ```python # Set a single piece of custom execution data _execution.customData.set(“key”, “value”);

```
# Set the custom execution data object
_execution.customData.setAll({"key1": "value1", "key2": "value2"})

# Access the current state of the object during the execution
customData = _execution.customData.getAll()

# Access a specific value set during this execution
customData = _execution.customData.get("key")
```

Refer to Custom executions data for more information.

---

getWorkflowStaticData(type)

This gives access to the static workflow data.

You can save data directly in the workflow. This data should be small.

As an example: you can save a timestamp of the last item processed from an RSS feed or database. It will always return an object. Properties can then read, delete or set on that object. When the workflow execution succeeds, n8n checks automatically if the data has changed and saves it, if necessary.

There are two types of static data, global and node. Global static data is the same in the whole workflow. Every node in the workflow can access it. The node static data is unique to the node. Only the node that set it can retrieve it again.

Example with global data:

=== “JavaScript” ```javascript // Get the global workflow static data const workflowStaticData = $getWorkflowStaticData(‘global’);

```
// Access its data
const lastExecution = workflowStaticData.lastExecution;

// Update its data
workflowStaticData.lastExecution = new Date().getTime();

// Delete data
delete workflowStaticData.lastExecution;
```

```

=== “Python” ```python # Get the global workflow static data workflowStaticData = _getWorkflowStaticData(‘global’)

```
# Access its data
lastExecution = workflowStaticData.lastExecution

# Update its data
workflowStaticData.lastExecution = new Date().getTime()

Delete data
delete workflowStaticData.lastExecution
```

Example with node data:

=== “JavaScript” ```js // Get the static data of the node const nodeStaticData = $getWorkflowStaticData(‘node’);

```
// Access its data
const lastExecution = nodeStaticData.lastExecution;

// Update its data
nodeStaticData.lastExecution = new Date().getTime();

// Delete data
delete nodeStaticData.lastExecution;
```

=== “Python” ```python # Get the static data of the node nodeStaticData = _getWorkflowStaticData(‘node’)

```
# Access its data
lastExecution = nodeStaticData.lastExecution

# Update its data
nodeStaticData.lastExecution = new Date().getTime()

# Delete data
delete nodeStaticData.lastExecution
```

Templates and examples

Retrieve linked items from earlier in the workflow

Every item in a node’s input data links back to the items used in previous nodes to generate it. This is useful if you need to retrieve linked items from further back than the immediate previous node.

To access the linked items from earlier in the workflow, use `("<node-name>").itemMatching(currentNodeinputIndex)`.

For example, consider a workflow that does the following:
```

1. The Customer Datastore node generates example data: `json [{ "id": "23423532", "name": "Jay Gatsby", "email": "gatsby@west-egg.com", "notes": "Keeps asking about a green light??", "country": "US", "created": "1925-04-10" }, { "id": "23423533", "name": "José Arcadio Buendía", "email": "jab@macondo.co", "notes": "Lots of people named after him. Very confusing", "country": "CO", "created": "1967-05-05" }, ...]`
2. The Edit Fields node simplifies this data: `json [ { "name": "Jay Gatsby" }, { "name": "José Arcadio Buendía" }, ... ]`
3. The Code node restore the email address to the correct person: `json [ { "name": "Jay Gatsby", "restoreEmail": "gatsby@west-egg.com" }, { "name": "José Arcadio Buendía", "restoreEmail": "jab@macondo.co" }, ... ]`

The Code node does this using the following code:

=== “JavaScript” `js for(let i=0; i<$input.all().length; i++) { $input.all()[i].json.restoreEmail = $('Customer Datastore (n8n training)').itemMatching(i).json.email; } return $input.all(); === “Python” ```python for i,item in enumerate(_input.all()): *input.all()[i].json.restoreEmail* = (‘Customer Datastore (n8n training)’).itemMatching(i).json.email

return _input.all();
```

You can view and download the example workflow from n8n website | itemMatchin usage example.

---

## ("<node-name>").all(branchIndex?: number, runIndex?: number)

This gives access to all the items of the current or parent nodes. If you don’t supply any parameters, it returns all the items of the current node.

## Getting items

=== “JavaScript” ```js // Returns all the items of the given node and current run let allItems = $(“”).all();

```
// Returns all items the node "IF" outputs (index: 0 which is Output "true" of its most recent run)
let allItems = $("IF").all();

// Returns all items the node "IF" outputs (index: 0 which is Output "true" of the same run as current node)
let allItems = $("IF").all(0, $runIndex);

// Returns all items the node "IF" outputs (index: 1 which is Output "false" of run 0 which is the first run)
let allItems = $("IF").all(1, 0);
```
```

```

=== “Python” ```python # Returns all the items of the given node and current run allItems = _(“”).all();

```
Returns all items the node "IF" outputs (index: 0 which is Output
"true" of its most recent run)
allItems = _("IF").all();

Returns all items the node "IF" outputs (index: 0 which is Output
"true" of the same run as current node)
allItems = _("IF").all(0, _runIndex);

Returns all items the node "IF" outputs (index: 1 which is Output
"false" of run 0 which is the first run)
allItems = _("IF").all(1, 0);
```

## Accessing item data

Get all items output by a previous node, and log out the data they contain:

=== “JavaScript” `typescript previousNodeData = $("<node-name>").all(); for(let i=0; i<previousNodeData.length; i++) { console.log(previousNodeData[i].json); } === “Python” python previousNodeData = _("<node-name>").all(); for item in previousNodeData: # item is of type <class 'pyodide.ffi.JsProxy'> # You need to convert it to a Dict itemDict = item.json.to_py() print(itemDict)`

---

## `vars`

`vars` contains all Variables for the active environment. It’s read-only: you can access variables using `vars`, but must set them using the UI.

=== “JavaScript” `js // Access a variable $vars.<variable-name>` === “Python” `python # Access a variable _vars.<variable-name>`

---

# Expressions cookbook

This section contains examples and recipes for tasks you can do with expressions.

## Related resources

- Built-in methods and variables reference
- Expressions

---
```

Check incoming data

At times, you may want to check the incoming data. If the incoming data doesn't match a condition, you may want to return a different value. For example, you want to check if a variable from the previous node is empty and return a string if it's empty. Use the following code snippet to return `not found` if the variable is empty.

```
{{$json["variable_name"]? $json["variable_name"] :"not found"}}
```

The above expression uses the ternary operator. You can learn more about the ternary operator here.

As an alternative, you can use the nullish coalescing operator (??) or the logical or operator (||):

```
{{ $x ?? "default value" }}
{{ $x || "default value" }}
```

In either of the above two cases, the value of `$x` will be used if it's set to a non-null, non-false value. The string `default value` is the fallback value.

Expressions common issues

Here are some common errors and issues related to expressions and steps to resolve or troubleshoot them.

The 'JSON Output' in item 0 contains invalid JSON

This error occurs when you use JSON mode but don't provide a valid JSON object. Depending on the problem with the JSON object, the error sometimes display as `The 'JSON Output' in item 0 does not contain a valid JSON object.`

To resolve this, make sure that the code you provide is valid JSON:

- Check the JSON with a JSON validator.
- Check that your JSON object doesn't reference undefined input data. This may occur if the incoming data doesn't always include the same fields.

Can't get data for expression

This error occurs when n8n can't retrieve the data referenced by an expression. Often, this happens when the preceding node hasn't been run yet.

Another variation of this may appear as `Referenced node is unexecuted`. In that case, the full text of this error will tell you the exact node that isn't executing in this format:

> An expression references the node '<node-name>', but it hasn't been executed yet. Either change the expression, or re-wire your workflow to make sure that node executes first.

To begin troubleshooting, test the workflow up to the named node.

For nodes that use JavaScript or other custom code, you can check if a previous node has executed before trying to use its value by checking the following:

```
$("<node-name>").isExecuted
```

As an example, this JSON references the parameters of the input data. This error will display if you test this step without connecting it to another node:

```
{
  "my_field_1": {{ $input.params }}
}
```

Invalid syntax

This error occurs when you use an expression that has a syntax error.

For example, the expression in this JSON includes a trailing period, which results in an invalid syntax error:

```
{
  "my_field_1": "value",
  "my_field_2": {{ $('If').item.json. }}
}
```

To resolve this error, check your expression syntax to make sure they follow the expected format.

Code node cookbook

This section contains examples and recipes for tasks you can do with the Code node.

Related resources

- Built-in methods and variables reference
- Code node

Get number of items returned by the previous node

To get the number of items returned by the previous node:

=== "JavaScript"

```js
```

```
if (Object.keys(items[0].json).length === 0) {
return [
    {
        json: {
            results: 0,
        }
    }
]
}
return [
    {
        json: {
            results: items.length,
        }
    }
];
```

The output will be similar to the following.

```json
[
    {
        "results": 8
    }
]
```

=== “Python” `python   if len(items[0].json) == 0:   return [{   "json": {   "results": 0,   }   }   ]   else: return [{   "json": {   "results": items.length,   }   }   ]`
The output will be similar to the following.

```json
[
    {
        "results": 8
    }
]
```

Get the binary data buffer

The binary data buffer contains all the binary file data processed by a workflow. You need to access it if you want to perform operations on the binary data, such as:

- Manipulating the data: for example, adding column headers to a CSV file.
- Using the data in calculations: for example, calculating a hash value based on it.
- Complex HTTP requests: for example, combining file upload with sending other data formats.

You can access the buffer using n8n’s `getBinaryDataBuffer()` function:

```
        /*
        * itemIndex: number. The index of the item in the input data.
        * binaryPropertyName: string. The name of the binary property.
        * The default in the Read/Write File From Disk node is 'data'.
        */
        let binaryDataBufferItem = await
this.helpers.getBinaryDataBuffer(itemIndex, binaryPropertyName);
```

For example:

```
        let binaryDataBufferItem = await this.helpers.getBinaryDataBuffer(0,
'data');
        // Returns the data in the binary buffer for the first input item
```

You should always use the `getBinaryDataBuffer()` function, and avoid using older methods of directly accessing the buffer, such as targeting it with expressions like `items[0].binary.data.data`.

Output to the browser console with `console.log()` or `print()` in the Code node

You can use `console.log()` or `print()` in the Code node to help when writing and debugging your code.

For help opening your browser console, refer to this guide by Balsamiq.

console.log (JavaScript)

For technical information on `console.log()`, refer to the MDN developer docs.

For example, copy the following code into a Code node, then open your console and run the node:

```
        let a = "apple";
        console.log(a);
```

print (Python)

For technical information on `print()`, refer to the Real Python's guide.

For example, set your Code node **Language** to **Python**, copy the following code into the node, then open your console and run the node:

```
        a = "apple"
        print(a)
```

Handling an output of `[object Object]`

If the console displays `[object Object]` when you print, check the data type, then convert it as needed.

To check the data type:

```
print(type(myData))
```

JsProxy

If `type()` outputs `<class 'pyodide.ffi.JsProxy'>`, you need to convert the JsProxy to a native Python object using `to_py()`. This occurs when working with data in the n8n node data structure, such as node inputs and outputs. For example, if you want to print the data from a previous node in the workflow:

```
previousNodeData = _("<node-name>").all();
for item in previousNodeData:
    # item is of type <class 'pyodide.ffi.JsProxy'>
    # You need to convert it to a Dict
    itemDict = item.json.to_py()
    print(itemDict)
```

Refer to the Pyodide documentation on JsProxy for more information on this class.

Examples using n8n's HTTP Request node

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

Refer to HTTP Request for information on node settings.

Related resources

- HTTP Request
- Built-in methods and variables reference
- Expressions

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.

–8<– "_snippets/integrations/builtin/core-nodes/http/pagination-api-differences.md"

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

Get the next page by number using `$pageCount`

If the API you're using supports targeting a specific 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 and is usually described in its documentation. For example, some APIs use 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.

Navigate pagination through body parameters

If the API you're using allows you to paginate through the body parameters:

1. Set the HTTP Request Method to **POST**
2. Set **Pagination Mode** to **Update a Parameter in Each Request**.
3. Select **Body** in the **Type** parameter.
4. Enter the **Name** of the body parameter. This depends on the API you're using. `page` is a common key name.
5. Hover over **Value** and toggle **Expression** on.
6. Enter `{{ $pageCount + 1 }}`

Set the page size in the query

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

1. Select **Send Query Parameters** in main node parameters (this is

the parameters you see when you first open the node, not the settings within options).
2. Enter the **Name** of the query parameter. This depends on your API. For example, a lot of APIs use a query parameter named `limit` to set page size. So **Name** would be `limit`.
3. In **Value**, enter your page size.

Advanced AI

Build AI functionality using n8n: from creating your own chat bot, to using AI to process documents and data from other sources.

- **Get started**

  Work through the short tutorial to learn the basics of building AI workflows in n8n.

  :octicons-arrow-right-24: Tutorial

- **Use a Starter Kit**

  Try n8n's Self-hosted AI Starter Kit to quickly start building AI workflows.

  :octicons-arrow-right-24: Self-hosted AI Starter Kit

- **Explore examples and concepts**

  Browse examples and workflow templates to help you build. Includes explanations of important AI concepts.

  :octicons-arrow-right-24: Examples

- **How n8n uses LangChain**

  Learn more about how n8n builds on LangChain.

  :octicons-arrow-right-24: LangChain in n8n

- **Browse AI templates**

  Explore a wide range of AI workflow templates on the n8n website.

  :octicons-arrow-right-24: AI workflows on n8n.io

Related resources

Related documentation and tools.

Node types

This feature uses Cluster nodes: groups of root and sub nodes that work together.

–8<– "_snippets/integrations/builtin/cluster-nodes/cluster-nodes-summary.md"

Workflow templates

You can browse workflow templates in-app or on the n8n website Workflows page.

Refer to Templates for information on accessing templates in-app.

Chat trigger

Use the n8n Chat Trigger to trigger a workflow based on chat interactions.

Chatbot widget

n8n provides a chatbot widget that you can use as a frontend for AI-powered chat workflows. Refer to the @n8n/chat npm page for usage information.

AI Workflow Builder

AI Workflow Builder enables you to create, refine, and debug workflows using natural language descriptions of your goals.

It handles the entire workflow construction process, including node selection, placement, and configuration, thereby reducing the time required to build functional workflows.

For details of pricing and availability of AI Workflow Builder, see n8n Plans and Pricing.

Working with the builder

1. **Describe your workflow:** Either select an example prompt or describe your requirements in natural language.
2. **Monitor the build:** The builder provides real-time feedback through several phases.
3. **Review and refine the generated workflow:** Review required credentials and other parameters. Refine the workflow using prompts.

   [Image: ai-workflow-builder.png]

Commands you can run in the builder

- `/clear`: Clears the context for the LLM and lets you start from scratch

Understanding credits

How credits work

Each time you send a message to the builder asking it to create or modify a workflow, that counts as one **interaction**, which is worth one **credit**.

✅ **Counts as an interaction**

- Sending a message to create a new workflow
- Asking the builder to modify an existing workflow
- Clicking the **Execute and refine** button in the builder window after a workflow is built

❌ **Does NOT count as an interaction**

- Messages that fail or produce generation errors
- Requests you manually stop by clicking the stop button

Getting more credits

If you've used your monthly limit, you can upgrade to a higher plan.

For details on plans and pricing, see n8n Plans and Pricing.

AI model and data handling

The following data are sent to the LLM:

- Text prompts that you provide to create, refine, or debug the workflow
- Node definitions, parameters, and connections and the current workflow definition.
- Any mock execution data that is loaded when using the builder

The following data are not sent:

- Details of any credentials you use
- Past executions of the workflow

Accessing n8n MCP server

Connect supported MCP clients to your n8n workflows through n8n's built-in MCP server.

The server allows clients such as Lovable or Claude Desktop to connect securely to an n8n instance. Once connected, these clients can:

- Search within workflows marked as available in MCP
- Retrieve metadata and trigger information for workflows
- Trigger and run exposed workflows

Difference between instance-level MCP access and MCP Server Trigger node

Instance-level MCP access lets you create one connection per n8n instance, use centralized authentication, and choose which workflows to enable for access. Enabled workflows are easy to find and run without extra setup for each workflow.

In comparison, you configure an MCP Server Trigger node inside a single workflow. This node exposes tools only from that workflow, a useful approach when you want to craft a specific MCP server behavior within one workflow.

Key considerations when using instance-level MCP access

- It isn't a way to build or edit workflows from an AI client; authoring remains in n8n.
- It doesn't provide blanket exposure to all workflows in your instance. You must enable MCP at the instance level and then enable each workflow individually.
- It's not scoped to each MCP client. Any connected client sees all workflows you've enabled for MCP access.

Enabling MCP access

For Cloud and self-hosted instances

1. Navigate to **Settings > MCP Access**
2. Toggle **Enable MCP access** (requires instance owner or admin permissions).

   [Image: enable-mcp-access.png]

Once enabled, you'll see:

- Connection instructions
- List of workflows exposed to MCP clients

To disable: Toggle the switch off.

For self-hosted: Complete disablement

To remove the feature entirely, set the environment variable:

```
N8N_DISABLED_MODULES=mcp
```

This action removes MCP endpoints and hides all related UI elements.

Setting up MCP authentication

The **How to connect** section on the **MCP Access** page provides two authentication methods for MCP clients:

- **OAuth2**
- **Access Token**

Using OAuth2

Copy your instance server URL from the **OAuth** tab and use it to configure your MCP client. After connecting, the client will redirect you to n8n to authorize access.

Revoking client access

To revoke access for connected MCP clients:

1. Navigate to **Settings > MCP Access**.
2. Switch to the **OAuth** tab in the **How to connect** section. You should see a table of connected clients in the **Connected OAuth clients** section.
3. Use the action menu in each client's row to revoke access for specific clients.

Using Access Token

Use your instance server URL and your personal MCP Access Token from the **Access Token** tab on the settings page.

When you first visit the **MCP Access page**, n8n automatically generates a personal MCP Access Token tied to your user account.

Rotating your token

If you lose your token or need to rotate it:

1. Navigate to **Settings > MCP Access**.
2. Make sure you are on the **Access Token** tab in the **How to connect** section.
3. Generate a new token.

   n8n revokes the previous token when you generate a new one.
4. Update all connected MCP clients with the new value.

Exposing workflows to MCP clients

Workflow eligibility

In order for a workflow to be available to MCP clients, it must meet the following criteria:

1. Be active
2. Contain one of the following trigger nodes:
   - Webhook
   - Schedule
   - Chat
   - Form

By default, no workflows are visible to MCP clients. You must explicitly enable access.

Enabling access

Option 1: From the workflow editor

1. Open the workflow.
2. Go to **Settings**.
3. Toggle **Available in MCP**.

Option 2: From the workflows list

1. Go to **Workflows**.
2. Open the menu on a workflow card.
3. Select **Enable MCP access**.

Managing access

The **MCP Access settings page** shows all workflows available to MCP clients. From this list you can:

- Open a workflow directly
- Revoke access using the action menu (or use **Disable MCP access** from the workflow card menu)

Workflow descriptions

To help MCP clients identify workflows, you can add free-text descriptions as follows:

1. Open the workflow.
2. Click the pencil icon next to the workflow name.
3. Enter your description in the **Description** field.

   [Image: mcp-access-workflow-descriptions.png]

Executing workflows through MCP clients

MCP clients can execute eligible workflows on your request. When a client triggers a workflow, it runs as usual in n8n, and you can monitor its execution in the **Executions** list. Once the execution is complete, the MCP client will retrieve the results.

Providing input data

If a workflow requires input data, the MCP client must provide that data when triggering the workflow. The workflow's trigger node determines the schema of the input data:

1. **Webhook trigger**: The MCP client will look for hints in workflow contents and its description. It's up to the workflow author to provide enough information for the client to generate valid input data.
2. **Schedule trigger**: No input data is required.
3. **Chat trigger**: Chat input format is determined by the chat node configuration.
4. **Form trigger**: Form fields are determined by the form node configuration.

Workflow timeouts

n8n enforces a 5-minute timeout for workflow executions triggered by MCP clients. If a workflow doesn't finish in time, n8n stops the execution and sends an error to the MCP client, ignoring any timeout you set in the workflow settings for MCP-triggered executions.

Limitations

- If there are multiple supported triggers in a workflow, MCP clients may only be able to use one (first one) of them to trigger the workflow.
- Executing workflows with multi-step forms or any kind of human-in-the-loop interactions isn't supported.
- Binary input data isn't supported. MCP clients can only provide text-based inputs for your workflows.

Examples

Connecting Lovable to n8n MCP server

1. Configure MCP Server in Lovable (OAuth).
   - Navigate to your workspace **Settings > Integrations**.
   - In the **MCP Servers** section, find **n8n** and click **Connect**.
   - Enter your n8n server URL (shown on the **MCP Access** page).
   - Save the connection. If successful, n8n redirects you to authorize Lovable.
2. Verify connectivity.
   - Once connected, Lovable can query for workflows with MCP access enabled.
   - **Example:** Asking Lovable to build a workflow UI that lists users and allows deleting them.

Connecting Claude Desktop to n8n MCP server

Using OAuth2

1. Navigate to **Settings > Connectors** in Claude Desktop.
2. Click on **Add custom connector**.
3. Enter the following details:
   - **Name:** n8n MCP
   - **Remote MCP Server URL**: Your n8n base URL (shown on the **MCP Access** page)
4. Save the connector.
5. When prompted, authorize Claude Desktop to access your n8n instance.

Using Access Token

Add the following entry to your `claude_desktop_config.json` file:

```
{
  "mcpServers": {
    "n8n-mcp": {
      "command": "npx",
      "args": [
```

```
            "-y",
            "supergateway",
            "--streamableHttp",
            "https://<your-n8n-domain>/mcp-server/http",
            "--header",
            "authorization:Bearer <YOUR_N8N_MCP_TOKEN>"
          ]
        }
      }
    }
```

Here, replace:

- `<your-n8n-domain>`: Your n8n base URL (shown on the **MCP Access** page)
- `<YOUR_N8N_MCP_TOKEN>`: Your generated token

Connecting Claude Code to n8n MCP server

Use the following CLI command:

```
claude mcp add --transport http n8n-mcp https://<your-n8n-domain>/mcp-server/http \
  --header "Authorization: Bearer <YOUR_N8N_MCP_TOKEN>"
```

Alternatively, add the following entry to your `claude.json` file:

```
{
    "mcpServers": {
        "n8n-local": {
            "type": "http",
            "url": "https://<your-n8n-domain>/mcp-server/http",
            "headers": {
                "Authorization": "Bearer <YOUR_N8N_MCP_TOKEN>"
            }
        }
    }
}
```

Here, replace:

- `<your-n8n-domain>`: Your n8n base URL (shown on the **MCP Access** page)
- `<YOUR_N8N_MCP_TOKEN>`: Your generated token

Connecting Codex CLI to n8n MCP server

Add the following entry to your `~/.codex/config.toml` file:

```
[mcp_servers.n8n_mcp]
command = "npx"
args = [
    "-y",
    "supergateway",
    "--streamableHttp",
    "https://<your-n8n-domain>/mcp-server/http",
    "--header",
    "authorization:Bearer <YOUR_N8N_MCP_TOKEN>"
]
```

Here, replace:

- `<your-n8n-domain>`: Your n8n base URL (shown on the **MCP Access** page)
- `<YOUR_N8N_MCP_TOKEN>`: Your generated token

Troubleshooting

If you encounter issues connecting MCP clients to your n8n instance, consider the following:

- Ensure that your n8n instance is publicly accessible if you are using cloud-based MCP clients.
- Verify that the MCP access is enabled in n8n settings.
- Check that the workflows you want to access are marked as available in MCP.
- Confirm that the authentication method (OAuth2 or Access Token) is correctly configured in your MCP client.
- Review n8n server logs for any error messages related to MCP connections.
- If you are using desktop MCP clients, make sure you have the latest Node.js version installed.

Build an AI chat agent with n8n

Welcome to the introductory tutorial for building AI workflows with n8n. Whether you have used n8n before, or this is your first time, we will show you how the building blocks of AI workflows fit together and construct a working AI-powered chat agent which you can easily customize for your own purposes.

[Image: “Screenshot of the completed workflow”]

Many people find it easier to take in new information in video format. This tutorial is based on one of n8n’s popular videos, linked below. Watch the video or read the steps here, or both!

What you will need

- **n8n**: For this tutorial we recommend using the n8n cloud service - there is a free trial for new users! For a self hosted service, refer to the installation pages.
- **Credentials for a chat model**: This tutorial uses OpenAI, but you can easily use DeepSeek, Google Gemini, Groq, Azure, and others (see the sub-nodes documentation for more).

What you will learn

- AI concepts in n8n
- How to use the AI Agent node
- Working with Chat input
- Connecting with AI models
- Customising input
- Observing the conversation
- Adding persistence

AI concepts in n8n

If you're already familiar with AI, feel free to skip this section. This is a basic introduction to AI concepts and how they can be used in n8n workflows.

An AI agent builds on Large Language Models (LLMs), which generate text based on input by predicting the next word. While LLMs only process input to produce output, AI agents add goal-oriented functionality. They can use tools, process their outputs, and make decisions to complete tasks and solve problems.

In n8n, the AI agent is represented as a node with some extra connections.

Feature	LLM	AI Agent
Core Capability	Text generation	Goal-oriented task completion
Decision-Making	None	Yes
Uses Tools/APIs	No	Yes
Workflow Complexity	Single-step	Multi-step
Scope	Generates language	Performs complex, real-world tasks
Example	LLM generating a paragraph	An agent scheduling an appointment

By incorporating the AI agent as a node, n8n can combine AI-driven steps with traditional programming for efficient, real-world workflows. For instance, simpler tasks, like validating an email address, do not require AI, whereas a complex tasks, like processing the *content* of an email or dealing with multimodal inputs (e.g., images, audio), are excellent uses of an AI agent.

1. Create a new workflow

–8<– "_snippets/try-it-out/new-workflow.md"

2. Add a trigger node

Every workflow needs somewhere to start. In n8n these are called 'trigger nodes'. For this workflow, we want to start with a chat node.

1. Select **Add first step** or press ++tab++ to open the node menu.
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 (Select **Back to canvas**) to return to the canvas.

??? explanation “More about the Chat Trigger node…” The trigger node generates output when there is an event causing it to trigger. In this case we want to be able to type in text to cause the workflow to run. In production, this trigger can be hooked up to a public chat interface as provided by n8n or embedded into another website. To start this simple workflow we will just use the built-in local chat interface to communicate, so no further setup is required.

3. Add an AI Agent Node

The AI Agent node is the core of adding AI to your workflows.

1. Select the **Add node** Image: Add node icon connector on the trigger node to bring up the node search.
2. Start typing “AI” and choose the **AI agent** node to add it.
3. The editing view of the **AI agent** will now be displayed.
4. There are some fields which can be changed. As we’re using the **Chat Trigger** node, the default setting for the source and specification of the prompt don’t need to be changed.

4. Configure the node

AI agents require a chat model to be attached to process the incoming prompts.

1. Add a chat model by clicking the plus Image: Add node icon button underneath the **Chat Model** connection on the **AI Agent** node (it’s the first connection along the bottom of the node).
2. The search dialog will appear, filtered on ‘Language Models’. These are the models with built-in support in n8n. For this tutorial we will use **OpenAI Chat Model**.
3. Selecting the **OpenAI Chat model** from the list will attach it to the **AI Agent** node and open the node editor. One of the parameters which can be changed is the ‘Model’. Note that for the basic OpenAI accounts, only the ‘gpt-4o-mini’ model is allowed.

??? explanation “Which chat model?” As mentioned earlier, the LLM is the component which generates the text according to a prompt it is given. LLMs have to be created and trained, usually an intensive process. Different LLMS may have different capabilities or specialties, depending on the data they were trained with.

5. Add credentials (if needed)

In order for n8n to communicate with the chat model, it will need some credentials (login data giving it access to an account on a different online service). If you already have credentials set up for OpenAI, these should appear by default in the credentials selector. Otherwise you can use the Credentials selector to help you add a new credential.

[Image: image showing the credentials dialog for OpenAI]

1. To add a new credential, click on the text which says ‘Select credential’. An option to add a new credential will appear [Image: Screenshot showing create a new credential button]

2. This credential just needs an API key. When adding credentials of any type, check the text to the right-hand side. In this case it has a handy link to take you straight to your OpenAI account to retrieve the API key.

3. The API key is just one long string. That’s all you need for this particular credential. Copy it from the OpenAI website and paste it into the **API key** section.

??? explanation “Keeping your credentials safe” 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. The type of information required varies depending on the app/service concerned. You should be careful about sharing or revealing the credentials outside of n8n.

6. Test the node

Now that the node is connected to the **Chat Trigger** and a chat model, we can test this part of the workflow.

1. Click on the ‘Chat’ button near the bottom of the canvas. This opens up a local chat window on the left and the AI agent logs on the right.

2. Type in a message and press ++enter++. You will now see the response from the chat model appear below your message.

3. The log window displays the inputs to and outputs from the AI Agent. [Image: image showing a chat session in progress]

??? explanation “Accessing the logs…” You can access the logs for the AI node even when you aren’t using the chat interface. Open up the **AI Agent** node and click on the **Logs** tab in the right hand panel. [Image: screenshot showing the Logs tab in the AIAgent]

7. Changing the prompt

The logs in the previous step reveal some extra data - the system prompt. This is the default message that the **AI Agent** primes the chat model with. From the log you can see this is set to “You are a helpful assistant”. We can however change this prompt to alter the behavior of the chat model.

1. Open the **AI Agent** node. In the bottom of the panel is a section labeled ‘Options’ and a selector labeled ‘Add Option’. Use this to select ‘System message’

2. The system message is now displayed. This is the same priming prompt we noticed before in the logs. Change the prompt to something else to prime the chat model in a different way. You could try something like “You are a brilliant poet who always replies in rhyming couplets” for example.

3. Close the node and return to the chat window. Repeat your message and notice how the output has changed. [Image: image showing changed text for chat, now it rhymes; if you can believe that]

8. Adding persistence

The chat model is now giving us useful output, but there is something wrong with it which will become apparent when you try to have a conversation.

1. Use the chat and tell the chat model your name, for example “Hi there, my name is Nick”.

2. Wait for the response, then type the message “What’s my name?”. The AI will not be able to tell you, however apologetic it may seem. The reason for this is we are not saving the context. The AI Agent has no memory. [Image: image showing a conversation illustrating the above]

3. In order to remember what has happened in the conversation, the AI Agent needs to preserve context. We can do this by adding memory to the **AI Agent** node. On the canvas click on the Image: Add node icon on the bottom of the **AI Agent** node labeled “Memory”.

4. From the panel which appears, select “Simple Memory”. This will use the memory from the instance running n8n, and is usually sufficient for simple usage. The default value of 5 interactions should be sufficient here, but remember where this option is if you may want to change it later.

5. Repeat the exercise of having a conversation above, and see that the AI Agent now remembers your name.

9. Saving the workflow

Before we leave the workflow editor, remember to save the workflow or all your changes will be lost.

1. Click on the “Save” button in the top right of the editor window. Your workflow will now be saved and you can return to it later to chat again or add new features.

Congratulations!

You have taken your first steps in building useful and effective workflows with AI. In this tutorial we have investigated the basic building blocks of an AI workflow, added an **AI Agent** and a chat model, and adjusted the prompt to get the kind of output we wanted. We also added memory so the chat could retain context between messages.

Next steps

Now you have seen how to create a basic AI workflow, there are plenty of resources to build on that knowledge and plenty of examples to give you ideas of where to go next:

- Learn more about AI concepts and view examples in Examples and concepts.
- Browse AI Workflow templates.
- Find out how to enhance the AI agent with tools.

RAG in n8n

What is RAG

Retrieval-Augmented Generation (RAG) is a technique that improves AI responses by combining language models with external data sources. Instead of relying solely on the model's internal training data, RAG systems retrieve relevant documents to ground responses in up-to-date, domain-specific, or proprietary knowledge. RAG workflows typically rely on vector stores to manage and search this external data efficiently.

What is a vector store?

A vector store is a special database designed to store and search high-dimensional vectors: numerical representations of text, images, or other data. When you upload a document, the vector store splits it into chunks and converts each chunk into a vector using an embedding model.

You can query these vectors using similarity searches, which construct results based on *semantic meaning*, rather than keyword matches. This makes vector stores a powerful foundation for RAG and other AI systems that need to retrieve and reason over large sets of knowledge.

How to use RAG in n8n

Inserting data into your vector store

Before your agent can access custom knowledge, you need to upload that data to a vector store:

1. Add the nodes needed to fetch your source data.
2. Insert a **Vector Store** node (e.g. the Simple Vector Store) and choose the **Insert Documents** operation.
3. Select an **embedding model**, which converts your text into vector embeddings. Consult the FAQ for more information on choosing the right embedding model.
4. Add a Default Data Loader node, which splits your content into chunks. You can use the default settings or define your own chunking strategy:
    - **Character Text Splitter:** splits by character length.
    - **Recursive Character Text Splitter:** recursively splits by

Markdown, HTML, code blocks or simple characters (recommended for most use cases).
    - **Token Text Splitter:** splits by token count.
5. (Optional) Add **metadata** to each chunk to enrich the context and allow better filtering later.

Querying your data

You can query the data in two main ways: using an agent or directly through a node.

Using agents

1. Add an agent to your workflow.
2. Add the vector store as a **tool** and give it a **description** to help the agent understand when to use it:
    - Set the **limit** to define how many chunks to return.
    - Enable **Include Metadata** to provide extra context for each chunk.
3. Add the same **embedding model** you used when inserting the data.

Using the node directly

1. Add your vector store node to the canvas and choose the **Get Many** operation.
2. Enter a query or prompt:
    - Set a **limit** for how many chunks to return.
    - Enable **Include Metadata** if needed.

FAQs

How do I choose the right embedding model?

The right embedding model differs from case to case.

In general, smaller models (for example, `text-embedding-ada-002`) are faster and cheaper and thus ideal for short, general-purpose documents or lightweight RAG workflows. Larger models (for example, `text-embedding-3-large`) offer better semantic understanding. These are best for long documents, complex topics, or when accuracy is critical.

What is the best text splitting for my use case?

This again depends a lot on your data:

- Small chunks (for example, 200 to 500 tokens) are good for fine-grained retrieval.
- Large chunks may carry more context but can become diluted or noisy.

Using the right overlap size is important for the AI to understand the context of the chunk. That’s also why using the Markdown or Code Block splitting can often help to make chunks better.

Another good approach is to add more context to it (for example, about the document where the chunk came from). If you want you can read more about this, you can check out this great article from Anthropic.

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.

Trigger nodes

Chat Trigger

Cluster nodes

–8<– "_snippets/integrations/builtin/cluster-nodes/cluster-nodes-summary.md"

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
- Sentiment Analysis
- Text Classifier

Learn more about chaining in LangChain.

Agents

> An agent{ data-preview} 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.

- Simple Vector Store
- PGVector 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
- Hugging Face Inference Model
- Mistral Cloud Chat Model
- Ollama Chat Model
- Ollama Model
- OpenAI Chat 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
- Postgres Chat Memory
- Simple 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
- Think Tool
- Vector Store Tool
- 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

LangChain learning resources

You don't need to know details about LangChain to use n8n, but it can be helpful to learn a few concepts. This pages lists some learning resources that people at n8n have found helpful.

The LangChain documentation includes introductions to key concepts and possible use cases. Choose the LangChain | Python or LangChain | JavaScript documentation for quickstarts, code examples, and API documentation. LangChain also provide code templates (Python only), offering ideas for potential use cases and common patterns.

What Product People Need To Know About LangChain provides a list of terminology and concepts, explained with helpful metaphors. Aimed at a wide audience.

If you prefer video, this YouTube series by Greg Kamradt works through the LangChain documentation, providing code examples as it goes.

n8n offers space to discuss LangChain on the Discord. Join to share your projects and discuss ideas with the community.

Use LangSmith with n8n

LangSmith is a developer platform created by the LangChain team. You can connect your n8n instance to LangSmith to record and monitor runs in n8n, just as you can in a LangChain application.

Connect your n8n instance to LangSmith

1. Log in to LangSmith and get your API key.
2. Set the LangSmith environment variables:

Variable	Value
`LANGCHAIN_ENDPOINT`	`"https://api.smith.langchain.com"`
`LANGCHAIN_TRACING_V2`	`true`
`LANGCHAIN_API_KEY`	Set this to your API key
`LANGCHAIN_PROJECT`	Optional project name (defaults to `"default"`)
`LANGCHAIN_CALLBACKS_BACKGROUND`	`true` (asynchronous trace upload)

   Set the variables so that they're available globally in the environment where you host your n8n instance. You can do this in the same way as the rest of your general configuration.

3. Restart n8n.

For information on using LangSmith, refer to LangSmith's documentation.

Overview

What are evaluations?

Evaluation is a crucial technique for checking that your AI workflow is reliable. It can be the difference between a flaky proof of concept and a solid production workflow. It's important both in the building phase and after deploying to production.

The foundation of evaluation is running a test dataset through your workflow. This dataset contains multiple test cases. Each test case contains a sample input for your workflow, and often includes the expected output(s) too.

Evaluation allows you to:

- **Test your workflow over a range of inputs** so you know how it performs on edge cases

- **Make changes with confidence** without inadvertently making things worse elsewhere
- **Compare performance** across different models or prompts

The following video explains what evaluations are, why they're useful, and how they work:

Why is evaluation needed?

AI models are fundamentally different than code. Code is deterministic and you can reason about it. This is difficult to do with LLMs, since they're black boxes. Instead, you must *measure* LLM output by running data through them and observing the output.

You can only build confidence that your model performs reliably after you have run it over multiple inputs that accurately reflect all the edge cases that it will have to deal with in production.

Two types of evaluation

Light evaluation (pre-deployment)

Building a clean, comprehensive dataset is hard. In the initial building phase, it often makes sense to generate just a handful of examples. These can be enough to iterate the workflow to a releasable state (or a proof of concept). You can visually compare the results to get a sense of the workflow's quality, without setting up formal metrics.

Metric-based evaluation (post-deployment)

Once you deploy your workflow, it's easier to build a bigger, more representative dataset from production executions. When you discover a bug, you can add the input that caused it to the dataset. When fixing the bug, it's important to run the whole dataset over the workflow again as a regression test to check that the fix hasn't inadvertently made something else worse.

Since there are too many test cases to check individually, evaluations measure the quality of the outputs using a metric, a numeric value representing a particular characteristic. This also allows you to track quality changes between runs.

Comparison of evaluation types

	Light evaluation (pre-deployment)	Metric-based evaluation (post-deployment)
Performance improvements with each iteration	Large	Small
Dataset size	Small	Large
Dataset sources	Hand-generated AI-generated Other	Production executions AI-generated Other

Actual outputs	Required	Required
Expected outputs	Optional	Required (usually)
Evaluation metric	Optional	Required

Learn more

- Light evaluations: Perfect for evaluating your AI workflows against hand-selected test cases during development.
- Metric-based evaluations: Advanced evaluations to maintain performance and correctness in production by using scoring and metrics with large datasets.
- Tips and common issues: Learn how to set up specific evaluation use cases and work around common issues.

Light evaluations

What are light evaluations?

When building your workflow, you often want to test it with a handful of examples to get a sense of how it performs and make improvements. At this stage of workflow development, looking over workflow outputs for each example is often enough. The benefits of setting up more formal scoring or metrics don't yet justify the effort.

Light evaluation allows you to run the examples in a test dataset through your workflow one-by-one, writing the outputs back to your dataset. You can then examine those outputs next to each other, and visually compare them to the expected outputs (if you have them).

How it works

Light evaluations take place in the 'Editor' tab of your workflow, although you'll find instructions on how to set it up in the 'Evaluations' tab.

Steps:

1. Create a dataset
2. Wire the dataset up to the workflow
3. Write workflow outputs back to dataset
4. Run evaluation

The following explanation will use a sample workflow that assigns a category and priority to incoming support tickets.

[Image: Example AI workflow]

1. Create a dataset

Create a data table or Google Sheet with a handful of examples for your workflow. Your dataset should contain columns for:

- The workflow input

- (Optional) The expected or correct workflow output
- The actual output

Leave the actual output column or columns blank, since you'll be filling them during the evaluation.

> [Image: Sample dataset for a support ticket classification workflow]
> A sample dataset for the support ticket classification workflow.

2. Wire the dataset up to your workflow

Insert an evaluation trigger to pull in your dataset

Each time the evaluation trigger runs, it will output a single item representing one row of your dataset.

Clicking the 'Evaluate all' button to the left of the evaluation trigger will run your workflow multiple times in sequence, once for each row in your dataset. This is a special behavior of the evaluation trigger.

While wiring the trigger up, you often only want to run it once. You can do this by either:

- Setting the trigger's 'Max rows to process' to 1
- Clicking on the 'Execute node' button on the trigger (rather than the 'Evaluate all' button)

Wire the trigger up to your workflow

You can now connect the evaluation trigger to the rest of your workflow and reference the data that it outputs. At a minimum, you need to use the dataset's input column(s) later in the workflow.

If you have multiple triggers in your workflow you will need to merge their branches together.

> [Image: Connecting the evaluation trigger]
> The support ticket classification workflow with the evaluation trigger added in and wired up.

3. Write workflow outputs back to dataset

To populate the output column(s) of your dataset when the evaluation runs:

- Insert the 'Set outputs' action of the evaluation node
- Wire it up to your workflow at a point after it has produced the outputs you're evaluating
- In the node's parameters, map the workflow outputs into the correct dataset column

> [Image: Connecting the set outputs node]
> The support ticket classification workflow with the 'set outputs' node added in and wired up.

4. Run evaluation

Click on the **Execute workflow** button to the left of the evaluation trigger. The workflow will execute multiple times, once for each row of the dataset:

[Image: Execute workflow button]

Review the outputs of each execution in the data table or Google Sheet, and examine the execution details using the workflow's 'executions' tab if you need to.

Once your dataset grows past a handful of examples, consider metric-based evaluation to get a numerical view of performance. See also tips and common issues.

Metric-based evaluations

What are metric-based evaluations?

Once your workflow is ready for deployment, you often want to test it on more examples than when you were building it.

For example, when production executions start to turn up edge cases, you want to add them to your test dataset so that you can make sure they're covered.

For large datasets like the ones built from production data, it can be hard to get a sense of performance just by eyeballing the results. Instead, you must measure performance. Metric-based evaluations can assign one or more scores to each test run, which you can compare to previous runs. Individual scores get rolled up to measure performance on the whole dataset.

This feature allows you to run evaluations that calculate metrics, track how those metrics change between runs and drill down into the reasons for those changes.

Metrics can be deterministic functions (such as the distance between two strings) or you can calculate them using AI. Metrics often involve checking how far away the output is from a *reference output* (also called ground truth). To do so, the dataset must contain that reference output. Some evaluations don't need this reference output though (for example, checking text for sentiment or toxicity).

How it works

1. Set up light evaluation
2. Add metrics to workflow
3. Run evaluation and view results

1. Set up light evaluation

Follow the setup instructions to create a dataset and wire it up to your workflow, writing outputs back to the dataset.

The following steps use the same support ticket classification workflow from the light evaluation docs:

[Image: Light evaluation workflow]

2. Add metrics to workflow

Metrics are dimensions used to score the output of your workflow. They often compare the actual workflow output with a reference output. It's common to use AI to calculate metrics, although it's sometimes possible to just use code. In n8n, metrics are always numbers.

You need to add the logic to calculate the metrics for your workflow, at a point after it has produced the outputs. You can add any reference outputs your metric uses as a column in your dataset. This makes sure they it will be available in the workflow, since they will be output by the evaluation trigger.

Use the **Set Metrics** operation to calculate:

- **Correctness (AI-based)**: Whether the answer's meaning is consistent with a supplied reference answer. Uses a scale of 1 to 5, with 5 being the best.
- **Helpfulness (AI-based)**: Whether the response answers the given query. Uses a scale of 1 to 5, with 5 being the best.
- **String Similarity**: How close the answer is to the reference answer, measured character-by-character (edit distance). Returns a score between 0 and 1.
- **Categorization**: Whether the answer is an exact match with the reference answer. Returns 1 when matching and 0 otherwise.
- **Tools Used**: Whether the execution used tools or not. Returns a score between 0 and 1.

You can also add custom metrics. Just calculate the metrics within the workflow and then map them into an Evaluation node. Use the **Set Metrics** operation and choose **Custom Metrics** as the Metric. You can then set the names and values for the metrics you want to return.

For example:

- RAG document relevance: when working with a vector database, whether the documents retrieved are relevant to the question.

Calculating metrics can add latency and cost, so you may only want to do it when running an evaluation and avoid it when making a production execution. You can do this by putting the metric logic after a 'check if evaluating' operation.

[Image: Check if evaluating node]

3. Run evaluation and view results

Switch to the **Evaluations** tab on your workflow and click the **Run evaluation** button. An evaluation will start. Once the evaluation has finished, it will display a summary score for each metric.

You can see the results for each test case by clicking on the test run row. Clicking on an individual test case will open the execution that produced it (in a new tab).

Tips and common issues

Combining multiple triggers

If you have another trigger in the workflow already, you have two potential starting points: that trigger and the evaluation trigger. To make sure your workflow works as expected no matter which trigger executes, you will need to merge these branches together.

> [Image: Merging trigger branches]
> Logic to merge two trigger branches together so that they have the same data format and can be referenced from a single node.

To do so:

1. **Get the data format of the other trigger**:
   - Execute the other trigger.
   - Open it and navigate to the JSON view of its output pane.
   - Click the **copy** button on the right.
2. **Re-shape the evaluation trigger data to match**:
   - Insert an Edit Fields (Set) node after the evaluation trigger and connect them together.
   - Change its mode to **JSON**.
   - Paste your data into the 'JSON' field, removing the `[` and `]` on the first and last lines.
   - Switch the field type to **Expression**.
   - Map in the data from the trigger by dragging it from the input pane.
   - For strings, make sure to replace the entire value (including the quotes) and add `.toJsonString()` to the end of the expression.
3. **Merge the branches using a 'No-op' node**: Insert a No-op node and wire both the other trigger and the Set node up to it. The 'No-op' node just outputs whatever input it receives.
4. **Reference the 'No-op' node outputs in the rest of the workflow**: Since both paths will flow through this node with the same format, you can be sure that your input data will always be there.

Avoiding evaluation breaking the chat

n8n's internal chat reads the output data of the last executed node in the workflow. After adding an evaluation node with the 'set outputs' operation, this data may not be in the expected format, or even contain the chat response.

[Image: Add second output branch]

The solution is to add an extra branch coming out of your agent. Lower branches execute later in n8n, which means any node you attach to this branch will execute last. You can use a no-op node here since it only needs to pass the agent output through.

Accessing tool data when calculating metrics

Sometimes you need to know what happened in executed sub-nodes of an agent, for example to check whether it executed a tool. You can't reference these nodes directly with expressions, but you can enable the **Return intermediate steps** option in the agent. This will add an extra output field called `intermediateSteps` which you can use in later nodes:

[Image: Enable return intermediate steps]

Multiple evaluations in the same workflow

You can only have one evaluation set up per workflow. In other words, you can only have one evaluation trigger per workflow.

Even so, you can still test different parts of your workflow with different evaluations by putting those parts in sub-workflows and evaluating each sub-workflow.

Dealing with inconsistent results

Metrics can often have noise: they may be different across evaluation runs of the exact same workflow. This is because the workflow itself may return different results, or any LLM-based metrics might have natural variation in them.

You can compensate for this by duplicating the rows of your dataset, so that each row appears more than once in the dataset. Since this means that each input will effectively be running multiple times, it will smooth out any variations.

Advanced AI examples and concepts

This section provides explanations of important AI concepts, and workflow templates that highlight those concepts, with explanations and configuration guides. The examples cover common use cases and highlight different features of advanced AI in n8n.

- **Agents and chains**

  Learn about agents and chains in AI, including exploring key differences using the example workflow.

  :octicons-arrow-right-24: What's a chain in AI?
  :octicons-arrow-right-24: What's an agent in AI?
  :octicons-arrow-right-24: Demonstration of key differences between agents and chains

- **Call n8n Workflow Tool**

  Learn about tools in AI, then explore examples that use n8n workflows as custom tools to give your AI workflow access to more data.

:octicons-arrow-right-24: What's a tool in AI?
:octicons-arrow-right-24: Chat with Google Sheets
:octicons-arrow-right-24: Call an API to fetch data
:octicons-arrow-right-24: Set up a human fallback
:octicons-arrow-right-24: Let AI specify tool parameters with `$fromAI()`

- **Vector databases**

  Learn about vector databases in AI, along with related concepts including embeddings and retrievers.

  :octicons-arrow-right-24: What's a vector database?
  :octicons-arrow-right-24: Populate a Pinecone vector database from a website

- **Memory**

  Learn about memory in AI.

  :octicons-arrow-right-24: What's memory in AI?

- **AI workflow templates**

  You can browse AI templates, included community contributions, on the n8n website.

  :octicons-arrow-right-24: Browse all AI templates

What's a chain in AI?

Chains bring together different components of AI to create a cohesive system. They set up a sequence of calls between the components. These components can include models and memory (though note that in n8n chains can't use memory).

Chains in n8n

n8n provides three chain nodes:

- Basic LLM Chain: use to interact with an LLM, without any additional components.
- Question and Answer Chain: can connect to a vector store using a retriever, or to an n8n workflow using the Workflow Retriever node. Use this if you want to create a workflow that supports asking questions about specific documents.
- Summarization Chain: takes an input and returns a summary.

There's an important difference between chains in n8n and in other tools such as LangChain: none of the chain nodes support memory. This means they can't remember previous user queries. If you use LangChain to code an AI application, you can give your application memory. In n8n, if you need your workflow to support memory, use an agent. This is essential if you want users to be able to have a natural ongoing conversation with your app.

What’s an agent in AI?

One way to think of an agent is as a chain that knows how to make decisions. Where a chain follows a predetermined sequence of calls to different AI components, an agent uses a language model to determine which actions to take.

Agents are the part of AI that act as decision-makers. They can interact with other agents and tools. When you send a query to an agent, it tries to choose the best tools to use to answer. Agents adapt to your specific queries, as well as the prompts that configure their behavior.

Agents in n8n

n8n provides one Agent node, which can act as different types of agent depending on the settings you choose. Refer to the Agent node documentation for details on the available agent types.

When you execute a workflow containing an agent, the agent runs multiple times. For example, it may do an initial setup, followed by a run to call a tool, then another run to evaluate the tool response and respond to the user.

Demonstration of key differences between agents and chains

In this workflow you can choose whether your chat query goes to an agent or chain. It shows some of the ways that agents are more powerful than chains.

Key features

This workflow uses:

- Chat Trigger: start your workflow and respond to user chat interactions. The node provides a customizable chat interface.
- Switch node: directs your query to either the agent or chain, depending on which you specify in your query. If you say “agent” it sends it to the agent. If you say “chain” it sends it to the chain.
- Agent: the Agent node interacts with other components of the workflow and makes decisions about what tools to use.
- Basic LLM Chain: the Basic LLM Chain node supports chatting with a connected LLM, but doesn’t support memory or tools.

Using the example

–8<– “_snippets/examples-color-key.md”

What’s memory in AI?

Memory is a key part of AI chat services. The memory keeps a history of previous messages, allowing for an ongoing conversation with the AI, rather than every interaction starting fresh.

AI memory in n8n

To add memory to your AI workflow you can use either:

- Simple Memory: stores a customizable length of chat history for the current session. This is the easiest to get started with.
- One of the memory services that n8n provides nodes for. These include:
  - Motorhead
  - Redis Chat Memory
  - Postgres Chat Memory
  - Xata
  - Zep

If you need to do advanced AI memory management in your workflows, use the Chat Memory Manager node.

–8<– “_snippets/integrations/builtin/cluster-nodes/langchain-sub-nodes/chat-memory-manager-purpose.md”

What’s 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)

AI 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 three especially powerful tools:

- Call n8n Workflow Tool: use this to load any n8n workflow as a tool.
- Custom Code Tool: write code that your agent can run.
- HTTP Request Tool: make calls to fetch a website or data from an API.

The next three examples highlight the Call n8n Workflow Tool:

- Chat with Google Sheets
- Call an API to fetch data

- Set up a human fallback

You can also learn how to let AI dynamically specify parameters for tools with the `$fromAI()` function.

Chat with a Google Sheet using AI

Use n8n to bring your own data to AI. This workflow uses the Chat Trigger to provide the chat interface, and the Call n8n Workflow Tool to call a second workflow that queries Google Sheets.

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.
- Call 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). The AI model uses the tool to access information beyond its built-in dataset.

Using the example

–8<– “_snippets/examples-color-key.md”

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 Call 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.

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.
- Call 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). The AI model uses the tool 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

–8<– "_snippets/examples-color-key.md"

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 Call n8n Workflow Tool to call a second workflow that handles checking for email addresses and sending the Slack message.

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.
- Call 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

–8<– "_snippets/examples-color-key.md"

Let AI specify the tool parameters

When configuring tools connected to the Tools Agent, many parameters can be filled in by the AI model itself. The AI model will use the context from the task and information from other connected tools to fill in the appropriate details.

There are two ways to do this, and you can switch between them.

Let the model fill in the parameter

Each appropriate parameter field in the tool's editing dialog has an extra button at the end:

[Image: image showing stars icon to the right of parameter field]

On activating this button, the AI Agent will fill in the expression for you, with no need for any further user input. The field itself is filled in with a message indicating that the parameter has been defined automatically by the model.

If you want to define the parameter yourself, click on the 'X' in this box to revert to user-defined values. Note that the 'expression' field will now contain the expression generated by this feature, though you can now edit it further to add extra details as described in the following section.

Use the `$fromAI()` function

The `$fromAI()` function uses AI to dynamically fill in parameters for tools connected to the Tools AI agent.

To use the `$fromAI()` function, call it with the required `key` parameter:

```
{{ $fromAI('email') }}
```

The `key` parameter and other arguments to the `$fromAI()` function aren't references to existing values. Instead, think of these arguments as hints that the AI model will use to populate the right data.

For instance, if you choose a key called `email`, the AI Model will look for an email address in its context, other tools, and input data. In chat workflows, it may ask the user for an email address if it can't find one elsewhere. You can optionally pass other parameters like `description` to give extra context to the AI model.

Parameters

The `$fromAI()` function accepts the following parameters:

Parameter	Type	Required?	Description
`key`	string	:white_check_mark:	A string representing the key or name of the argument. This must be between 1 and 64 characters in length and can only contain lowercase letters, uppercase letters, numbers, underscores, and hyphens.
`description`	string	:x:	A string describing the argument.
`type`	string	:x:	A string specifying the data type. Can be string, number, boolean, or json (defaults to string).
			The default value to

`defaultValue`	any	:x:	use for the argument.

Examples

As an example, you could use the following `$fromAI()` expression to dynamically populate a field with a name:

```
$fromAI("name", "The commenter's name", "string", "Jane Doe")
```

If you don't need the optional parameters, you could simplify this as:

```
$fromAI("name")
```

To dynamically populate the number of items you have in stock, you could use a `$fromAI()` expression like this:

```
$fromAI("numItemsInStock", "Number of items in stock", "number", 5)
```

If you only want to fill in parts of a field with a dynamic value from the model, you can use it in a normal expression as well. For example, if you want the model to fill out the `subject` parameter for an e-mail, but always pre-fix the generated value with the string 'Generated by AI:', you could use the following expression:

```
Generated by AI: {{ $fromAI("subject") }}
```

Templates

You can see the `$fromAI()` function in action in the following templates:

- Angie, Personal AI Assistant with Telegram Voice and Text
- Automate Customer Support Issue Resolution using AI Text Classifier
- Scale Deal Flow with a Pitch Deck AI Vision, Chatbot and QDrant Vector Store

---

What are vector databases?

Vector databases store information as numbers:

> A vector database is a type of database that stores data as high-dimensional vectors, which are mathematical representations of features or attributes. (source)

This enables fast and accurate similarity searches. With a vector database, instead of using conventional database queries, you can search for relevant data based on semantic and contextual meaning.

A simplified example

A vector database could store the sentence "n8n is a source-available automation tool that you can self-host", but instead of storing it as text, the vector database stores an array of dimensions (numbers

between 0 and 1) that represent its features. This doesn't mean turning each letter in the sentence into a number. Instead, the vectors in the vector database describe the sentence.

Suppose that in a vector store `0.1` represents `automation tool`, `0.2` represents `source available`, and `0.3` represents `can be self-hosted`. You could end up with the following vectors:

Sentence	Vector (array of dimensions)
n8n is a source-available automation tool that you can self-host	[0.1, 0.2, 0.3]
Zapier is an automation tool	[0.1]
Make is an automation tool	[0.1]
Confluence is a wiki tool that you can self-host	[0.3]

Demonstrating the power of similarity search

Qdrant provides vector search demos to help users understand the power of vector databases. The food discovery demo shows how a vector store can help match pictures based on visual similarities.

> This demo uses data from Delivery Service. Users may like or dislike the photo of a dish, and the app will recommend more similar meals based on how they look. It's also possible to choose to view results from the restaurants within the delivery radius. (source)

For full technical details, refer to the Qdrant demo-food-discovery GitHub repository.

Embeddings, retrievers, text splitters, and document loaders

Vector databases require other tools to function:

- Document loaders and text splitters: document loaders pull in documents and data, and prepare them for embedding. Document loaders can use text splitters to break documents into chunks.
- Embeddings: these are the tools that turn the data (text, images, and so on) into vectors, and back into raw data. Note that n8n only supports text embeddings.
- Retrievers: retrievers fetch documents from vector databases. You need to pair them with an embedding to translate the vectors back into data.

Populate a Pinecone vector database from a website

Use n8n to scrape a website, load the data into Pinecone, then query it using a chat workflow. This workflow uses the HTTP node to get website data, extracts the relevant content using the HTML node, then uses the Pinecone Vector Store node to send it to Pinecone.

Key features

This workflow uses:

- HTTP node: fetches website data.
- HTML node: simplifies the data by extracting the main content from the page.
- Pinecone Vector Store node and Embeddings OpenAI: transform the data into vectors and store it in Pinecone.
- Chat Trigger and Question and Answer Chain to query the vector database.

Using the example

–8<– “_snippets/examples-color-key.md”

---

n8n public REST API

Using n8n’s public API, you can programmatically perform many of the same tasks as you can in the GUI. This section introduces n8n’s REST API, including:

- How to authenticate
- Paginating results
- Using the built-in API playground (self-hosted n8n only)
- Endpoint reference

n8n provides an n8n API node to access the API in your workflows.

Learn about REST APIs

The API documentation assumes you are familiar with REST APIs. If you’re not, these resources may be helpful:

- KnowledgeOwl’s guide to working with APIs: a basic introduction, including examples of how to call REST APIs.
- IBM Cloud Learn Hub - What is an Application Programming Interface (API): this gives a general, but technical, introduction to APIs.
- IBM Cloud Learn Hub - What is a REST API?: more detailed information about REST APIs.
- MDN web docs - An overview of HTTP: REST APIs work over HTTP and use HTTP verbs, or methods, to specify the action to perform.

API authentication

n8n uses API keys to authenticate API calls.

API Scopes

Users of enterprise instances can limit which resources and actions a key can access with scopes. API key scopes allow you specify the exact level of access a key needs for its intended purpose.

Non-enterprise API keys have full access to all the account's resources and capabilities.

Create an API key

1. Log in to n8n.
2. Go to **Settings** > **n8n API**.
3. Select **Create an API key**.
4. Choose a **Label** and set an **Expiration** time for the key.
5. If on an enterprise plan, choose the **Scopes** to give the key.
6. Copy **My API Key** and use this key to authenticate your calls.

Call the API using your key

Send the API key in your API call as a header named `X-N8N-API-KEY`.

For example, say you want to get all active workflows. Your curl request will look like this:

```
# For a self-hosted n8n instance
curl -X 'GET' \
  '<N8N_HOST>:<N8N_PORT>/<N8N_PATH>/api/v<version-number>/workflows?
active=true' \
  -H 'accept: application/json' \
  -H 'X-N8N-API-KEY: <your-api-key>'

# For n8n Cloud
curl -X 'GET' \
  '<your-cloud-instance>/api/v<version-number>/workflows?
active=true' \
  -H 'accept: application/json' \
  -H 'X-N8N-API-KEY: <your-api-key>'
```

Delete an API key

1. Log in to n8n.
2. Go to **Settings** > **n8n API**.
3. Select **Delete** next to the key you want to delete.
4. Confirm the delete by selecting **Delete Forever**.

API pagination

The default page size is 100 results. You can change the page size limit. The maximum permitted size is 250.

When a response contains more than one page, it includes a cursor, which you can use to request the next pages.

For example, say you want to get all active workflows, 150 at a time.

Get the first page:

```
# For a self-hosted n8n instance
curl -X 'GET' \
  '<N8N_HOST>:<N8N_PORT>/<N8N_PATH>/api/v<version-number>/workflows?
active=true&limit=150' \
  -H 'accept: application/json' \
  -H 'X-N8N-API-KEY: <your-api-key>'

# For n8n Cloud
curl -X 'GET' \
  '<your-cloud-instance>/api/v<version-number>/workflows?
active=true&limit=150' \
  -H 'accept: application/json' \
  -H 'X-N8N-API-KEY: <your-api-key>'
```

The response is in JSON format, and includes a `nextCursor` value. This is an example response.

```
{
  "data": [
    // The response contains an object for each workflow
    {
      // Workflow data
    }
  ],
  "nextCursor": "MTIzZTQ1NjctZTg5Yi0xMmQzLWE0NTYtNDI2NjE0MTc0MDA"
}
```

Then to request the next page:

```
# For a self-hosted n8n instance
curl -X 'GET' \
  '<N8N_HOST>:<N8N_PORT>/<N8N_PATH>/api/v<version-number>/workflows?
active=true&limit=150&cursor=MTIzZTQ1NjctZTg5Yi0xMmQzLWE0NTYtNDI2NjE
0MTc0MDA' \
  -H 'accept: application/json'

# For n8n Cloud
curl -X 'GET' \
  '<your-cloud-instance>/api/v<version-number>/workflows?
active=true&limit=150&cursor=MTIzZTQ1NjctZTg5Yi0xMmQzLWE0NTYtNDI2NjE
0MTc0MDA' \
  -H 'accept: application/json'
```

Using an API playground

This documentation site provides a playground to test out calls. Self-hosted users also have access to a built-in playground hosted as part of their instance.

Documentation playground

You can test API calls from this site's API reference. You need to set your server's base URL and instance name, and add an API key.

n8n uses Scalar's open source API platform to power this functionality.

Built-in playground

The n8n API comes with a built-in Swagger UI playground in self-hosted versions. This provides interactive documentation, where you can try out requests. The path to access the playground depends on your hosting.

n8n constructs the path from values set in your environment variables:

```
N8N_HOST:N8N_PORT/N8N_PATH/api/v<api-version-number>/docs
```

The API version number is `1`. There may be multiple versions available in the future.

The API includes built-in documentation about credential formats. This is available using the `credentials` endpoint:

```
N8N_HOST:N8N_PORT/N8N_PATH/api/v<api-version-
number>/credentials/schema/{credentialTypeName}
```

---

n8n Embed

n8n Embed is part of n8n's paid offering. Using Embed, you can white label n8n, or incorporate it in your software as part of your commercial product.

For more information about when to use Embed, as well as costs and licensing processes, refer to Embed on the n8n website.

Support

The community forum can help with various issues. If you are a current Embed customer, you can also contact n8n support, using the email provided when you bought the license.

Russia and Belarus

n8n Embed isn't available in Russia and Belarus. Refer to n8n's blog post Update on n8n cloud accounts in Russia and Belarus for more information.

Prerequisites

–8<– "_snippets/embed-license.md"

The requirements provided here are an example based on n8n Cloud and are for illustrative purposes only. Your requirements may vary depending on the number of users, workflows, and executions. Contact n8n for more information.

Component	Sizing	Supported
CPU/vCPU	Minimum 10 CPU cycles, scaling as needed	Any public or private cloud
Database	512 MB - 4 GB SSD	SQLite or PostgreSQL
Memory	320 MB - 2 GB	

CPU considerations

n8n isn't CPU intensive so even small instances (of providers such as AWS and GCP) should be enough for most use cases. Usually, memory requirements supersede CPU requirements, so focus resources there when planning your infrastructure.

Database considerations

n8n uses its database to store credentials, past executions, and workflows.

A core feature of n8n is the flexibility to choose a database. All the supported databases have different advantages and disadvantages, which you have to consider individually and pick the one that best suits your needs. By default n8n creates an SQLite database if no database exists at the given location.

n8n recommends that every n8n instance have a dedicated database. This helps to prevent dependencies and potential performance degradation. If it isn't possible to provide a dedicated database for every n8n instance, n8n recommends making use of Postgres's schema feature.

For Postgres, the database must already exist on the DB-instance. The database user for the n8n process needs to have full permissions on all tables that they're using or creating. n8n creates and maintains the database schema.

Best practices

- SSD storage.
- In containerized cloud environments, ensure that the volume is persisted and mounted when stopping/starting a container. If not, all data is lost.
- If using Postgres, don't use the `tablePrefix` configuration option. It will be deprecated in the near future.
- Pay attention to the changelog of new versions and consider reverting migrations before downgrading.
- Set up at least the basic database security and stability mechanisms such as IP allow lists and backups.

Memory considerations

An n8n instance doesn't typically require large amounts of available memory. For example an n8n Cloud instance at idle requires ~100MB. It's the nature of your workflows and the data being processed that determines your memory requirements.

For example, while most nodes just pass data to the next node in the workflow, the Code node creates a pre-processing and post-processing copy of the data. When dealing will large binary files, this can consume all available resources.

Deployment

–8<– "_snippets/embed-license.md"

See the hosting documentation for detailed setup options.

User data

n8n recommends that you follow the same or similar practices used internally for n8n Cloud: Save user data using Rook and, if an n8n server goes down, a new instance starts on another machine using the same data.

Due to this, you don't need to use backups except in case of a catastrophic failure, or when a user wants to reactivate their account within your prescribed retention period (two weeks for n8n Cloud).

Backups

n8n recommends creating nightly backups by attaching another container, and copying all data to this second container. In this manner, RAM usage is negligible, and so doesn't impact the amount of users you can place on the server.

Restarting

If your instance is down or restarting, missed executions (for example, Cron or Webhook nodes) during this time aren't recoverable. If it's important for you to maintain 100% uptime, you need to build another proxy in front of it which caches the data.

Configuration

–8<– "_snippets/embed-license.md"

Authentication

You can secure n8n by setting up User management, n8n's built-in authentication feature.

n8n supports LDAP and SAML.

Credential overwrites

To offer OAuth login to users, it's possible to overwrite credentials on a global basis. This credential data isn't visible to users but the backend uses it automatically.

In the Editor UI, n8n hides all overwritten fields by default. This means that users are able to authenticate using OAuth by pressing the "connect" button on the credentials.

n8n offers two ways to apply credential overwrites: using Environment Variable and using the REST API.

Using environment variables

You can set credential overwrites using environment variable by setting the `CREDENTIALS_OVERWRITE_DATA` to `{ CREDENTIAL_NAME: { PARAMETER: VALUE }}`.

Using REST APIs

The recommended way is to load the data using a custom REST endpoint. Set the `CREDENTIALS_OVERWRITE_ENDPOINT` to a path under which this endpoint should be made available. You can set `CREDENTIALS_OVERWRITE_ENDPOINT_AUTH_TOKEN` to require a token for accessing the endpoint. When this token is configured, the endpoint is only accessible if the token is included in the `Authorization` header as a `Bearer` token.

For example:

1. Activate the endpoint by setting the environment variable in the environment n8n runs under:

```
export CREDENTIALS_OVERWRITE_ENDPOINT=send-credentials
```

2. A JSON file with the credentials to overwrite is then needed. For example, a `oauth-credentials.json` file to overwrite credentials for Asana and GitHub could look like this:

```
{
    "asanaOAuth2Api": {
        "clientId": "<id>",
        "clientSecret": "<secret>"
    },
    "githubOAuth2Api": {
        "clientId": "<id>",
        "clientSecret": "<secret>"
    }
}
```

3. Then apply it to the instance by sending it using curl:

```
curl -H "Content-Type: application/json" --data @oauth-credentials.json http://localhost:5678/send-credentials
```

In case `CREDENTIALS_OVERWRITE_ENDPOINT_AUTH_TOKEN` is set to `secure-token`, the curl command will be:

```sh
curl -H "Content-Type: application/json" -H "Authorization: Bearer
secure-token" --data @oauth-credentials.json
http://localhost:5678/send-credentials
```

Persistence

To store credential overwrites in the database and propagate them automatically to all workers in multi-instance/queue mode, enable:

```
export CREDENTIALS_OVERWRITE_PERSISTENCE=true
```

When enabled, n8n stores the encrypted overwrites in the `settings` table and broadcasts a `reload-overwrite-credentials` event so that workers reload the latest values. When disabled, overwrites remain in memory on the process that loaded them and aren't propagated to workers or preserved across restarts.

Environment variables

n8n has many environment variables you can configure. Here are the most relevant environment variables for your hosted solution:

Variable	Type	Default	Desc
`EXECUTIONS_TIMEOUT`	Number	`-1`	Sets (in se work n8n s execu overr indiv up to in `EXECU` Set `E` to `-1`
`EXECUTIONS_DATA_PRUNE`	Boolean	`true`	Whet of pa rollin
`EXECUTIONS_DATA_MAX_AGE`	Number	`336`	The e hours delet
`EXECUTIONS_DATA_PRUNE_MAX_COUNT`	Number	`10000`	Maxi execu the d limit
`NODES_EXCLUDE`	Array of strings	-	Spec not t exam node secu aren' `NODES` `[\"n8`

			base. \"n8n base.
NODES_INCLUDE	Array of strings	-	Spec load.
N8N_TEMPLATES_ENABLED	Boolean	true	Enab temp disab
N8N_TEMPLATES_HOST	String	https://api.n8n.io	Chan your temp that work libra provi endp respo n8n's Work more

Backend hooks

It's possible to define external hooks that n8n executes whenever a specific operation runs. You can use these, for example, to log data, change data, or forbid an action by throwing an error.

Available hooks

Hook	Arguments	Description
credentials.create	[credentialData: ICredentialsDb]	Called before new credentials get created. Use to restrict the number of credentials.
credentials.delete	[id: credentialId]	Called before credentials get deleted.
credentials.update	[credentialData: ICredentialsDb]	Called before existing credentials are saved.
frontend.settings	[frontendSettings: IN8nUISettings]	Gets called on n8n startup. Allows you to, for example, overwrite frontend data like the displayed OAuth URL.
n8n.ready	[app: App]	Called once n8n is ready. Use to, for example, register

		custom API endpoints.
`n8n.stop`		Called when an n8n process gets stopped. Allows you to save some process data.
`oauth1.authenticate`	`[oAuthOptions: clientOAuth1.Options, oauthRequestData: {oauth_callback: string}]`	Called before an OAuth1 authentication. Use to overwrite an OAuth callback URL.
`oauth2.callback`	`[oAuth2Parameters: {clientId: string, clientSecret: string | undefined, accessTokenUri: string, authorizationUri: string, redirectUri: string, scopes: string[]}]`	Called in an OAuth2 callback. Use to overwrite an OAuth callback URL.
`workflow.activate`	`[workflowData: IWorkflowDb]`	Called before a workflow gets activated. Use to restrict the number of active workflows.
`workflow.afterCreate`	`[workflowId: string]`	Called after a workflow gets created.
`workflow.afterDelete`	`[workflowId: string]`	Called after a workflow gets deleted.
`workflow.afterUpdate`	`[workflowData: IWorkflowBase]`	Called after an existing workflow gets saved.
`workflow.create`	`[workflowData: IWorkflowBase]`	Called before a workflow gets created. Use to restrict the number of saved workflows.
`workflow.delete`	`[workflowId: string]`	Called before a workflow gets delete.
`workflow.postExecute`	`[run: IRun, workflowData: IWorkflowBase]`	Called after a workflow gets executed.
`workflow.preExecute`	`[workflow: Workflow: mode: WorkflowExecuteMode]`	Called before a workflow gets executed. Allows you to count or limit the number of workflow executions.

`workflow.update`	`[workflowData: IWorkflowBase]`	Called before an existing workflow gets saved.
`workflow.afterArchive`	`[workflowId: string]`	Called after you archive a workflow.
`workflow.afterUnarchive`	`[workflowId: string]`	Called after you restore a workflow from the archive.

Registering hooks

Set hooks by registering a hook file that contains the hook functions. To register a hook, set the environment variable `EXTERNAL_HOOK_FILES`.

You can set the variable to a single file:

```
EXTERNAL_HOOK_FILES=/data/hook.js
```

Or to contain multiple files separated by a colon:

```
EXTERNAL_HOOK_FILES=/data/hook1.js:/data/hook2.js
```

Backend hook files

Hook files are regular JavaScript files that have the following format:

```
        module.exports = {
            "frontend": {
                "settings": [
                    async function (settings) {
                        settings.oauthCallbackUrls.oauth1 =
'https://n8n.example.com/oauth1/callback';
                        settings.oauthCallbackUrls.oauth2 =
'https://n8n.example.com/oauth2/callback';
                    }
                ]
            },
            "workflow": {
                "activate": [
                    async function (workflowData) {
                        const activeWorkflows = await
this.dbCollections.Workflow.count({ active: true });

                        if (activeWorkflows > 1) {
                            throw new Error(
                                'Active workflow limit reached.'
                            );
                        }
                    }
                ]
            }
        }
```

Backend hook functions

A hook or a hook file can contain multiple hook functions, with all functions executed one after another.

If the parameters of the hook function are objects, it's possible to change the data of that parameter to change the behavior of n8n.

You can also access the database in any hook function using `this.dbCollections` (refer to the code sample in Backend hook files.

Frontend external hooks

Like backend external hooks, it's possible to define external hooks in the frontend code that get executed by n8n whenever a user performs a specific operation. You can use them, for example, to log data and change data.

Available hooks

Hook	Description
`credentialsEdit.credentialTypeChanged`	Called when an existing credential's type changes.
`credentials.create`	Called when someone creates a new credential.
`credentialsList.dialogVisibleChanged`	
`dataDisplay.nodeTypeChanged`	
`dataDisplay.onDocumentationUrlClick`	Called when someone selects the help documentation link.
`execution.open`	Called when an existing execution opens.
`executionsList.openDialog`	Called when someone selects an execution from existing Workflow Executions.
`expressionEdit.itemSelected`	
`expressionEdit.dialogVisibleChanged`	
`nodeCreateList.filteredNodeTypesComputed`	
`nodeCreateList.nodeFilterChanged`	Called when someone makes any changes to the node panel filter.
`nodeCreateList.selectedTypeChanged`	
`nodeCreateList.mounted`	
`nodeCreateList.destroyed`	
`nodeSettings.credentialSelected`	
`nodeSettings.valueChanged`	
`nodeView.createNodeActiveChanged`	
`nodeView.addNodeButton`	
`nodeView.createNodeActiveChanged`	
`nodeView.mount`	
`pushConnection.executionFinished`	
`showMessage.showError`	
`runData.displayModeChanged`	
`workflow.activeChange`	

`workflow.activeChangeCurrent`	
`workflow.afterUpdate`	Called when someone updates an existing workflow.
`workflow.open`	
`workflowRun.runError`	
`workflowRun.runWorkflow`	Called when a workflow executes.
`workflowSettings.dialogVisibleChanged`	
`workflowSettings.saveSettings`	Called when someone saves the settings of a workflow.

Registering hooks

You can set hooks by loading the hooks script on the page. One way to do this is by creating a hooks file in the project and adding a script tag in your `editor-ui/public/index.html` file:

Frontend hook files

Frontend external hook files are regular JavaScript files which have the following format:

```
window.n8nExternalHooks = {
  nodeView: {
    mount: [
      function (store, meta) {
        // do something
      },
    ],
    createNodeActiveChanged: [
      function (store, meta) {
        // do something
      },
      function (store, meta) {
        // do something else
      },
    ],
    addNodeButton: [
      function (store, meta) {
        // do something
      },
    ],
  },
};
```

Frontend hook functions

You can define multiple hook functions per hook. Each hook function is invoked with the following arguments arguments:

- `store`: The Vuex store object. You can use this to change or get data from the store.
- `metadata`: The object that contains any data provided by the hook. To see what's passed, search for the hook in the `editor-ui` package.

Workflow management in Embed

–8<– “_snippets/embed-license.md”

When managing an embedded n8n deployment, spanning across teams or organizations, you will likely need to run the same (or similar) workflows for multiple users. There are two available options for doing so:

Solution	Pros	Cons
Create a workflow for each user	No limitation on how workflow starts (can use any trigger)	Requires managing multiple workflows.
Create a single workflow, and pass it user credentials when executing	Simplified workflow management (only need to change one workflow).	To run the workflow, your product must call it

Workflow per user

There are three general steps to follow:

- Obtain the credentials for each user, and any additional parameters that may be required based on the workflow.
- Create the n8n credentials for this user.
- Create the workflow.

1. Obtain user credentials

Here you need to capture all credentials for any node/service this user must authenticate with, along with any additional parameters required for the particular workflow. The credentials and any parameters needed will depend on your workflow and what you are trying to do.

2. Create user credentials

After all relevant credential details have been obtained, you can proceed to create the relevant service credentials in n8n. This can be done using the Editor UI or API call.

Using the Editor UI

1. From the menu select **Credentials > New**.
2. Use the drop-down to select the **Credential type** to create, for example *Airtable*. [Image: Create New Credentials drop-down]
3. In the **Create New Credentials** modal, enter the corresponding credentials details for the user, and select the nodes that will have access to these credentials. [Image: Create New Credentials modal]

4. Click **Create** to finish and save.

Using the API

The frontend API used by the Editor UI can also be called to achieve the same result. The API endpoint is in the format: `https://<n8n-domain>/rest/credentials`.

For example, to create the credentials in the Editor UI example above, the request would be:

```
POST https://<n8n-domain>/rest/credentials
```

With the request body:

```
{
   "name":"MyAirtable",
   "type":"airtableApi",
   "nodesAccess":[
      {
         "nodeType":"n8n-nodes-base.airtable"
      }
   ],
   "data":{
      "apiKey":"q12we34r5t67yu"
   }
}
```

The response will contain the ID of the new credentials, which you will use when creating the workflow for this user:

```
{
   "data":{
      "name":"MyAirtable",
      "type":"airtableApi",
      "data":{
         "apiKey":"q12we34r5t67yu"
      },
      "nodesAccess":[
         {
            "nodeType":"n8n-nodes-base.airtable",
            "date":"2021-09-10T07:41:27.770Z"
         }
      ],
      "id":"29",
      "createdAt":"2021-09-10T07:41:27.777Z",
      "updatedAt":"2021-09-10T07:41:27.777Z"
   }
}
```

3. Create the workflow

Best practice is to have a “base” workflow that you then duplicate and customize for each new user with their credentials (and any other details).

You can duplicate and customize your template workflow using either the Editor UI or API call.

Using the Editor UI

1. From the menu select **Workflows** > **Open** to open the template workflow to be duplicated.
2. Select **Workflows** > **Duplicate**, then enter a name for this new workflow and click **Save**. [Image: Duplicate workflow]
3. Update all relevant nodes to use the credentials for this user (created above).
4. **Save** this workflow set it to **Active** using the toggle in the top-right corner.

Using the API

1. Fetch the JSON of the template workflow using the endpoint: `https://<n8n-domain>/rest/workflows/<workflow_id>`

```
GET https://<n8n-domain>/rest/workflows/1012
```

The response will contain the JSON data of the selected workflow:


```
        {
          "data": {
            "id": "1012",
            "name": "Nathan's Workflow",
            "active": false,
            "nodes": [
              {
                "parameters": {},
                "name": "Start",
                "type": "n8n-nodes-base.start",
                "typeVersion": 1,
                "position": [
                  130,
                  640
                ]
              },
              {
                "parameters": {
                  "authentication": "headerAuth",
                  "url": "https://internal.users.n8n.cloud/webhook/custom-
erp",
                  "options": {
                    "splitIntoItems": true
                  },
                  "headerParametersUi": {
                    "parameter": [
                      {
                        "name": "unique_id",
                        "value": "recLhLYQbzNSFtHNq"
                      }
                    ]
                  }
                },
                "name": "HTTP Request",
                "type": "n8n-nodes-base.httpRequest",
                "typeVersion": 1,
                "position": [
                  430,
                  300
                ],
```

```
    "credentials": {
      "httpHeaderAuth": "beginner_course"
    }
  },
  {
    "parameters": {
      "operation": "append",
      "application": "appKBGQfbm6NfW6bv",
      "table": "processingOrders",
      "options": {}
    },
    "name": "Airtable",
    "type": "n8n-nodes-base.airtable",
    "typeVersion": 1,
    "position": [
      990,
      210
    ],
    "credentials": {
      "airtableApi": "Airtable"
    }
  },
  {
    "parameters": {
      "conditions": {
        "string": [
          {
            "value1": "={{$json[\"orderStatus\"]}}",
            "value2": "processing"
          }
        ]
      }
    },
    "name": "IF",
    "type": "n8n-nodes-base.if",
    "typeVersion": 1,
    "position": [
      630,
      300
    ]
  },
  {
    "parameters": {
      "keepOnlySet": true,
      "values": {
        "number": [
          {
            "name": "=orderId",
            "value": "={{$json[\"orderID\"]}}"
          }
        ],
        "string": [
          {
            "name": "employeeName",
            "value": "={{$json[\"employeeName\"]}}"
          }
        ]
      },
      "options": {}
    },
```

```
      "name": "Set",
      "type": "n8n-nodes-base.set",
      "typeVersion": 1,
      "position": [
        800,
        210
      ]
    },
    {
      "parameters": {
        "functionCode": "let totalBooked = items.length;\nlet
bookedSum = 0;\n\nfor(let i=0; i < items.length; i++) {\n  bookedSum
= bookedSum + items[i].json.orderPrice;\n}\nreturn [{json:
{totalBooked, bookedSum}}]\n"
      },
      "name": "Function",
      "type": "n8n-nodes-base.function",
      "typeVersion": 1,
      "position": [
        800,
        400
      ]
    },
    {
      "parameters": {
        "webhookUri":
"https://discord.com/api/webhooks/865213348202151968/oD5_WPDQwtr22Vj
d_82QP3-_4b_lGhAeM7RynQ8Js5DzyXrQEnj0zeAQIA6fki1JLtXE",
        "text": "=This week we have {{$json[\"totalBooked\"]}}
booked orders with a total value of {{$json[\"bookedSum\"]}}. My
Unique ID: {{$node[\"HTTP
Request\"].parameter[\"headerParametersUi\"][\"parameter\"][0]
[\"value\"]}}"
      },
      "name": "Discord",
      "type": "n8n-nodes-base.discord",
      "typeVersion": 1,
      "position": [
        1000,
        400
      ]
    },
    {
      "parameters": {
        "triggerTimes": {
          "item": [
            {
              "mode": "everyWeek",
              "hour": 9
            }
          ]
        }
      },
      "name": "Cron",
      "type": "n8n-nodes-base.cron",
      "typeVersion": 1,
      "position": [
        220,
        300
      ]
```

```
  }
],
"connections": {
  "HTTP Request": {
    "main": [
      [
        {
          "node": "IF",
          "type": "main",
          "index": 0
        }
      ]
    ]
  },
  "Start": {
    "main": [
      []
    ]
  },
  "IF": {
    "main": [
      [
        {
          "node": "Set",
          "type": "main",
          "index": 0
        }
      ],
      [
        {
          "node": "Function",
          "type": "main",
          "index": 0
        }
      ]
    ]
  },
  "Set": {
    "main": [
      [
        {
          "node": "Airtable",
          "type": "main",
          "index": 0
        }
      ]
    ]
  },
  "Function": {
    "main": [
      [
        {
          "node": "Discord",
          "type": "main",
          "index": 0
        }
      ]
    ]
  },
  "Cron": {
```

```
            "main": [
              [
                {
                  "node": "HTTP Request",
                  "type": "main",
                  "index": 0
                }
              ]
            ]
          }
        },
        "createdAt": "2021-07-16T11:15:46.066Z",
        "updatedAt": "2021-07-16T12:05:44.045Z",
        "settings": {},
        "staticData": null,
        "tags": []
      }
    }
```


1. Save the returned JSON data and update any relevant credentials and fields for the new user.

2. Create a new workflow using the updated JSON as the request body at endpoint: `https://<n8n-domain>/rest/workflows`

```
POST https://<n8n-domain>/rest/workflows/
```

The response will contain the ID of the new workflow, which you will use in the next step.

1. Lastly, publish the new workflow:

```
PATCH https://<n8n-domain>/rest/workflows/1012
```

Passing the additional value `active` in your JSON payload:

```
// ...
"active":true,
"settings": {},
"staticData": null,
"tags": []
```

Single workflow

There are four steps to follow to implement this method:

- Obtain the credentials for each user, and any additional parameters that may be required based on the workflow. See Obtain user credentials above.
- Create the n8n credentials for this user. See Create user credentials above.
- Create the workflow.
- Call the workflow as needed.

Create the workflow

The details and scope of this workflow will vary greatly according to the individual use case, however there are a few design implementations to keep in mind:

- This workflow must be triggered by a Webhook node.
- The incoming webhook call must contain the user's credentials and any other workflow parameters required.
- Each node where the user's credentials are needed should use an expression so that the node's credential field reads the credential provided in the webhook call.
- Save and publish the workflow, ensuring the production URL is selected for the Webhook node. Refer to webhook node for more information.

Call the workflow

For each new user, or for any existing user as may be needed, call the webhook defined as the workflow trigger and provide the necessary credentials (and any other workflow parameters).

Workflow templates

–8<– "_snippets/embed-license.md"

n8n provides a library of workflow templates. When embedding n8n, you can:

- Continue to use n8n's workflow templates library (this is the default behavior)
- Disable workflow templates
- Create your own workflow templates library

Disable workflow templates

–8<– "_snippets/workflows/templates/disable-templates.md"

Use your own workflow templates library

–8<– "_snippets/workflows/templates/custom-templates-library.md"

Add your workflows to the n8n library

–8<– "_snippets/workflows/templates/submit-templates.md"

White labelling

–8<– "_snippets/embed-license.md"

White labelling n8n means customizing the frontend styling and assets to match your brand identity. The process involves changing two packages in n8n's source code github.com/n8n-io/n8n:

- packages/frontend/@n8n/design-system: n8n's storybook design system with CSS styles and Vue.js components
- packages/frontend/editor-ui: n8n's Vue.js frontend build with

Vite.js

Prerequisites

You need the following installed on your development machine:

–8<– “_snippets/integrations/creating-nodes/prerequisites.md”

Create a fork of n8n’s repository and clone your new repository.

```
git clone https://github.com/<your-organization>/n8n.git n8n
cd n8n
```

Install all dependencies, build and start n8n.

```
npm install
npm run build
npm run start
```

Whenever you make changes you need to rebuild and restart n8n. While developing you can use `npm run dev` to automatically rebuild and restart n8n anytime you make code changes.

Theme colors

To customize theme colors open packages/frontend/@n8n/design-system and start with:

- packages/frontend/@n8n/design-system/src/css/_tokens.scss
- packages/frontend/@n8n/design-system/src/css/_tokens.dark.scss

At the top of `_tokens.scss` you will find `--color-primary` variables as HSL colors:

```
@mixin theme {
    --color-primary-h: 6.9;
    --color-primary-s: 100%;
    --color-primary-l: 67.6%;
```

In the following example the primary color changes to #0099ff. To convert to HSL you can use a color converter tool.

```
@mixin theme {
    --color-primary-h: 204;
    --color-primary-s: 100%;
    --color-primary-l: 50%;
```

[Image: Example Theme Color Customization]

Theme logos

To change the editor’s logo assets look into packages/frontend/editor-ui/public and replace:

- favicon-16x16.png
- favicon-32x32.png
- favicon.ico
- n8n-logo.svg

- n8n-logo-collapsed.svg
- n8n-logo-expanded.svg

Replace these logo assets. n8n uses them in Vue.js components, including:

- MainSidebar.vue: top/left logo in the main sidebar.
- Logo.vue: reused in other components.

In the following example replace `n8n-logo-collapsed.svg` and `n8n-logo-expanded.svg` to update the main sidebar's logo assets.

[Image: Example Logo Main Sidebar]

If your logo assets require different sizing or placement you can customize SCSS styles at the bottom of MainSidebar.vue.

```
        .logoItem {
            display: flex;
            justify-content: space-between;
            height: $header-height;
            line-height: $header-height;
            margin: 0 !important;
            border-radius: 0 !important;
            border-bottom: var(--border-width-base) var(--border-style-base)
var(--color-background-xlight);
            cursor: default;

            &:hover, &:global(.is-active):hover {
                background-color: initial !important;
            }

            * { vertical-align: middle; }
            .icon {
                height: 18px;
                position: relative;
                left: 6px;
            }

        }
```

Text localization

To change all text occurrences like `n8n` or `n8n.io` to your brand identity you can customize n8n's English internationalization file: packages/frontend/@n8n/i18n/src/locales/en.json.

n8n uses the Vue I18n internationalization plugin for Vue.js to translate the majority of UI texts. To search and replace text occurrences inside `en.json` you can use Linked locale messages.

In the following example add the `_brand.name` translation key to white label n8n's AboutModal.vue.

```
        {
            "_brand.name": "My Brand",
            //replace n8n with link to _brand.name
            "about.aboutN8n": "About @:_brand.name",
            "about.n8nVersion": "@:_brand.name Version",
        }
```

[Image: Example About Modal Localization]

Window title

To change n8n's window title to your brand name, edit the following:

- packages/frontend/editor-ui/index.html
- packages/frontend/editor-ui/src/composables/useDocumentTitle.ts

The following example replaces all occurrences of `n8n` and `n8n.io` with `My Brand` in `index.html` and `useDocumentTitle.ts`.

```
<!DOCTYPE html>
<html lang="en">
<head>

    <title>My Brand - Workflow Automation</title>
</head>
```

```
import { useSettingsStore } from '@/stores/settings.store';

// replace n8n
const DEFAULT_TITLE = 'My Brand';
const DEFAULT_TAGLINE = 'Workflow Automation';
```

[Image: Example Window Title Localization]