Get ALL AI Automation + Agent Templates >>

Full Guide - Master 90% of n8n by Learning Just These 13 Nodes

(Follow the instructions in this video: https://youtu.be/6XVnvLu2epc)

Automations and Agents Mentioned in Video:

AI Agent Template: AI Voice Agent Calls New Leads Instantly and Books Appointments
Tutorial: https://youtu.be/zvUmgNJRZlk
Template: https://fabimarkl.com/automation-templates/#lead-caller

AI Automation Template: Clones successful "Text-on-Video" Instagram Reels With Your Face
Tutorial: https://youtu.be/nCP9b4xxNmw
Template: https://fabimarkl.com/automation-templates/#reel-remix

AI Automation Template: Turns Photos to Money Making Reels, Shorts & TikToks
Tutorial: https://youtu.be/cPREhs-SlMc
Template: https://fabimarkl.com/automation-templates/#photo-to-video

AI Agent Template: Handles Product Inquiries, Checks Orders & Records Support Tickets via WhatsApp
Tutorial: https://youtu.be/EJ9FFQlhcmw
Template: https://fabimarkl.com/automation-templates/#wa-agent

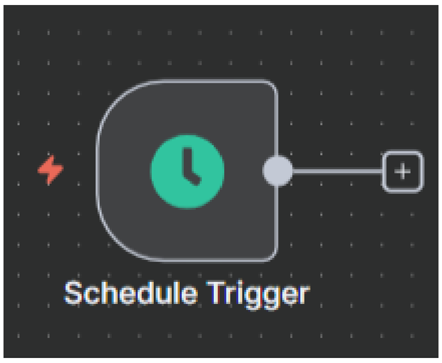
Links:

Free Trial for N8N Account:
https://n8n.io/

**n8n link is affiliate link that will give me a small percentage of every sale, but have no effect on the price you pay or their free trial.*

Click Here to Get AI Automation Template for Instant Setup >>

Schedule Trigger Node



Doc:
https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.scheduletrigger/

Imagine you have a task that needs to happen every single day at the same time - like posting to Instagram at 9 AM, or checking for new orders every hour. You don't want to sit at your computer and manually click "run" every time. That's what the Schedule Trigger does - it automatically starts your automation at whatever time or interval you choose, without you having to do anything.

Think of it like setting an alarm clock for your automation. Once you set it up, it just runs on its own schedule forever until you turn it off.

When You Would Actually Use This

Here are real examples from my templates that people buy:

Daily social media posting: You want to post content to Instagram, TikTok, or LinkedIn every single day at 9 AM. The Schedule Trigger starts the automation at 9 AM, then the automation generates the post and publishes it. You wake up and your content is already posted.

Weekly report generation: Every Monday at 8 AM, you want to generate a report of last week's sales or leads. Schedule Trigger runs every Monday, pulls the data, creates the report, and emails it to you.

Hourly data checking: You're monitoring a website for new products, job listings, or price changes. Schedule Trigger checks every hour automatically so you don't miss anything.

Monthly invoicing: On the first day of every month, you want to create invoices for all your clients. Schedule Trigger runs on the 1st, generates all invoices, and sends them out.

The key thing to understand: this is for anything that needs to happen on a predictable schedule that you can set in advance. It's NOT for things that happen based on someone doing something (like filling out a form) - we'll cover those triggers later.

How to Actually Set This Up (Step by Step)

Let me walk you through this like you've never touched n8n before.

Step 1: Add the node to your canvas

When you're in n8n, click the plus icon (+) to add a new node. Type "schedule" in the search box. You'll see "Schedule Trigger" - click it. It appears as a purple node on your canvas.

Step 2: Decide what kind of schedule you want

When you click on the node, you'll see settings on the right side. The first important choice is "Mode" - this determines what type of schedule you're setting. There are two options:

Interval Mode: This means "run every X amount of time" - like every 3 hours, or every 2 days. You use this when you think in terms of "how often should this repeat?"

Cron Mode: This is for specific times - like "every Monday at 9 AM" or "the 1st of every month at midnight." You use this when you think in terms of "what exact time should this happen?"

For most people starting out, Interval mode is easier to understand, so let's start there.

Step 3: Set up an Interval schedule (the easier option)

Let's say you want your automation to run every single day at 9 AM. Here's exactly what you do:

Click on "Trigger Interval" dropdown. You'll see options: Seconds, Minutes, Hours, Days, Weeks, Months. Choose "Days" because we want it to run every day.

Then you'll see "Trigger Every" - type "1" because we want every 1 day (not every 2 days or 3 days, just every single day).

Below that, you'll see "Trigger at Hour" - type "9" because we want 9 AM.

And "Trigger at Minute" - type "0" because we want exactly 9:00, not 9:30 or 9:15.

That's it. Now your automation will run every single day at exactly 9:00 AM.

Step 4: Understanding what happens when it runs

Every day at 9 AM, this node will activate and send a signal to whatever node comes next in your workflow. It's like pressing a "start" button automatically. The Schedule Trigger itself doesn't do anything except start the process - all the actual work happens in the nodes you connect after it.

Common Mistakes People Make (And How to Avoid Them)

Mistake 1: Not checking your timezone

Your n8n instance runs on a specific timezone. If you set your schedule for 9 AM but your n8n server is in a different timezone than you are, it might actually run at 2 PM your time. To check your timezone, go to your n8n settings and look for "Timezone" to make sure it matches where you are.

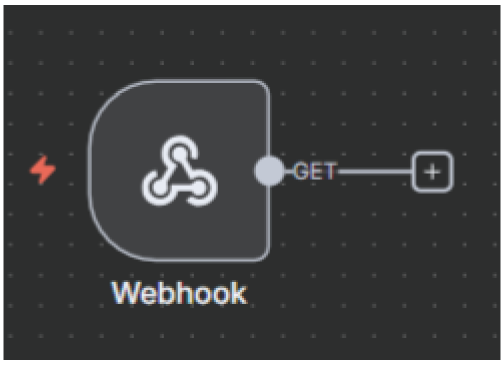
Mistake 2: Setting intervals too frequently

I've seen people set their automation to run every 10 seconds because they want fast results. But here's the problem: if your automation calls an external service (like an AI image generator or social media API), those services usually have limits on how many times you can call them per hour. If you run too frequently, you'll hit those limits and get blocked. Unless you have a specific reason to run every few seconds, stick to every few minutes or hours at minimum.

Mistake 3: Forgetting to activate the workflow

```
This seems obvious, but I've done this myself many times. You
set up your schedule, test it manually, and it works great. But
then you close n8n and nothing happens at the scheduled time.
Why? Because the workflow needs to be "Active" (toggle the
switch at the top of your workflow to ON). If it's not active,
schedules won't run.
```

Webhook Node



Docs:
https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.webhook/#webhook-urls

```
A webhook is like a mailbox for your automation. When you create
a webhook in n8n, you get a special web address (a URL - that's
the thing that starts with https://). When someone or something
sends information to that address, your automation immediately
wakes up and starts running.

Think of it like this: Imagine you have a doorbell at your house.
When someone presses the doorbell button, it rings inside and you
know someone's there. A webhook is the same thing - it's a
"doorbell" for your automation. When something "presses the
button" by sending data to your webhook URL, your automation
immediately knows "hey, something happened, time to start
working."

The difference between this and the Schedule Trigger is timing:
Schedule Trigger runs on a clock you set. Webhook runs instantly
whenever something sends it information.
```

When You Would Actually Use This

Here are real situations where webhooks are perfect:

Form submissions: Someone fills out a contact form on your website and clicks "Submit." The instant they click submit, the form sends all their information (name, email, message) to your webhook URL. Your automation immediately receives it and can send you an email notification, add them to your database, or send them an automated reply. You don't have to wait for a schedule - it happens right away.

Payment notifications: Someone buys your product. Your payment system (like Stripe or PayPal) can send a webhook to your n8n with the purchase details. Your automation instantly receives "Hey, John Smith just paid $99 for Product X" and can immediately send the download link, add them to your customer list, or send a thank you email.

App integrations: You use a tool like Typeform for surveys, Calendly for bookings, or Shopify for your store. All these tools can send webhooks when something happens - a form is submitted, a meeting is booked, or an order is placed. Your n8n automation receives this instantly and can do whatever you need with that information.

Other automations calling this one: Sometimes you have one n8n workflow that needs to trigger another n8n workflow. The first workflow can send data to the second workflow's webhook to wake it up.

The key thing to understand: webhooks are for things that happen unpredictably, when you don't know exactly when they'll occur. Someone might fill out your form at 2 AM or 3 PM - you can't schedule that. But with a webhook, whenever it happens, your automation handles it immediately.

How to Actually Set This Up (Step by Step)

Step 1: Add the webhook node

Click the plus (+) in n8n, search for "webhook," and add the Webhook node. It'll appear on your canvas.

Step 2: Choose the HTTP Method

This is where people get confused, so let me explain this clearly.

When you click on your webhook node, you'll see a setting called "HTTP Method" with options like GET, POST, PUT, DELETE. What the heck are these?

Think of HTTP Methods as different types of messages you can send. The two you'll use 99% of the time are:

GET: This means "I'm asking for information." Imagine you're calling a friend and saying "Hey, what's the weather like there?" You're not sending them information, you're asking for it. GET is used when something wants to retrieve data from your webhook.

POST: This means "I'm sending you information." Now imagine your friend calls you and says "Hey, I just wanted to let you know I'm coming to visit on Friday." They're sending you information. POST is used when something is sending data to your webhook - like form submissions, payment notifications, basically anytime someone's giving your automation new information to process.

How do you know which to use?

Look at the documentation (instructions) from whatever service is sending the webhook. For example:

- If you're connecting a form (like Typeform, Google Forms, or a contact form), it will say in their settings "we send data via POST" - so you choose POST
- If you're connecting a calendar booking system, it will tell you what method it uses
- If you're not sure, POST is the safe default because 90% of webhooks that send data use POST

Step 3: Set the path (optional but recommended)

You'll see a field called "Path." This is just a name for your webhook to make it easier to remember what it's for.

For example, if this webhook is for form submissions, you might name it "contact-form" If it's for payment notifications, name it "payment-received"

Your webhook URL will then be:
https://your-n8n-instance.com/webhook/contact-form

You don't have to set a path, but it makes it way easier to organize multiple webhooks.

Step 4: Get your webhook URL

Once you save your node, you'll see two URLs appear:

Test URL: This is for testing while you're building. It starts with "webhook-test" in the address.

Production URL: This is the real one you'll use when your workflow is live and active.

Here's the important part that trips everyone up: THE TEST URL ONLY WORKS WHEN YOU'RE ACTIVELY WORKING ON THE WORKFLOW. If you close n8n or navigate away, the test URL stops working. That's why it's only for testing.

When you're ready to actually use this webhook for real, you need to:

1. Activate your workflow (toggle the switch to ON at the top)
2. Copy the Production URL (not the test URL)
3. Give that production URL to whatever service is sending the webhook

Step 5: Test it

Before you start using it for real, you should test if it works. Here's the easiest way:

Open a new tab in your browser. Paste your test URL into the address bar. Add `?name=John` to the end of it. It will look like: `https://your-n8n.com/webhook-test/abc123?name=John`

Press Enter.

If you see any response (even if it says "Workflow executed successfully" or shows some data), that means your webhook is working! If you go back to n8n, you should see an execution in your execution list showing that the webhook received the data.

Common Mistakes People Make

Mistake 1: Using the test URL in production

This is the #1 mistake. Someone sets up their webhook, copies the test URL, gives it to their form or app, and it works great during testing. Then they close n8n and suddenly nothing works. Why? Because test URLs only work while you're actively testing.

Always use the production URL for anything real, and make sure your workflow is activated.

Mistake 2: Wrong HTTP method

Someone sets their webhook to GET but the form is sending POST data. Result: the webhook never receives anything. Always check what method the service uses. When in doubt, try POST first - it's the most common for receiving data.

Mistake 3: Not handling the response

Some services that send webhooks expect a response back. If they don't get one, they think the webhook failed and keep trying to send it over and over. Your webhook should usually be set to "Respond to Webhook" mode with a 200 response code (that means "success, I got your message"). This is usually the default, but double-check it's enabled.

Mistake 4: No authentication

Once your webhook URL is out there, technically anyone who knows that URL could send data to it. For testing and personal projects, this is usually fine. But if you're handling sensitive data (payments, personal information), you should add authentication. This means the sender needs to include a secret code (usually called an API key) with every request to prove they're authorized. Most services that send webhooks have an option to include an authentication header - you generate a random password, tell the service to include it in every webhook, and configure your webhook to check for it.

Real Example From My Templates

In my "WhatsApp AI Agent" template, I use a webhook to receive messages from WhatsApp. Here's how it works:

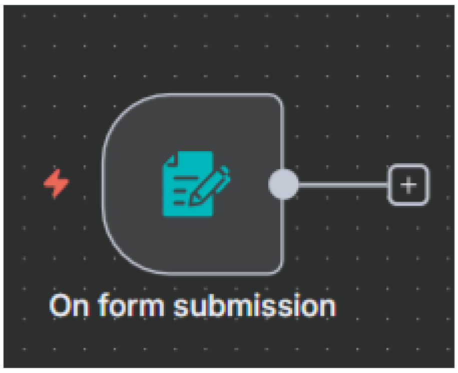
WhatsApp's API sends a webhook every time someone messages my business number. The webhook contains the message text, who sent it, and when. My n8n webhook immediately receives this, then:

1. The AI reads the message
2. Decides what to respond
3. Sends the response back to WhatsApp

All of this happens in seconds because the webhook wakes up the automation instantly when a message arrives. If I used a Schedule

```
Trigger checking every 5 minutes, people would wait 5 minutes for
responses. With a webhook, it's immediate.
```

Form Trigger Node



The Form Trigger creates an actual form that people can fill out - you know, like those forms you see on websites where you type in your name, email, and a message, then click Submit. This node creates that entire form for you inside n8n, gives you a link to the form, and when someone fills it out and hits submit, it immediately starts your automation with all the information they entered.

The beautiful thing about this is you don't need to know how to code a form, host it on a website, or connect it to anything. n8n creates the whole form for you. You just decide what fields you want (name, email, phone, etc.), and n8n handles everything else.

When You Would Actually Use This

Lead capture: You want people to give you their contact information so you can follow up with them. You create a form with fields for Name, Email, Phone, and "What service are you interested in?" When someone fills it out, your automation immediately adds them to your Google Sheet, sends you a notification email, and sends them an automated "Thanks, we'll contact you soon" response.

Booking requests: You run a service business (salon, consulting, coaching) and want people to request appointments. Your form asks for their name, email, preferred date/time, and what service they want. When they submit, your automation checks your calendar for availability and either confirms the booking or suggests alternative times.

Contact forms: Instead of people emailing you directly, you give them a form. They fill it out, your automation sends you a formatted notification with all their info, and automatically sends them a "We received your message" reply.

Survey or feedback collection: You want to know what people think about your product or service. You create a form with rating questions and a comment box. When people submit, their responses go into a Google Sheet automatically for you to review later.

The advantage over Webhook: With a Webhook, someone else has to create the form and then send the data to your webhook URL. With Form Trigger, n8n creates the entire form for you. It's simpler if you don't already have a form somewhere else.

How to Actually Set This Up (Step by Step)

Step 1: Add the Form Trigger node

Click the plus (+) button in n8n, type "form" in the search, and add the "Form Trigger" node to your canvas.

Step 2: Give your form a title

When you click on the Form Trigger node, the first thing you'll see is "Form Title." This is the heading people will see at the top of your form. Make it clear what the form is for:

- "Contact Us"
- "Book a Free Consultation"
- "Get a Quote"
- "Request More Information"

This isn't just decoration - it tells people what they're filling out and why.

Step 3: Add form fields

This is where you decide what information you want to collect. Click "Add Field" and you'll see different field types:

Text: For short answers like names. You might add a Text field called "Full Name"

Email: Specifically for email addresses. n8n will automatically check that what they type looks like a real email (has an @ and a domain). Add one called "Email Address"

Number: For numbers only, like "How many people will attend?" or "What's your budget?"

Dropdown: Gives people a list of options to choose from. Perfect for things like:

- "What service do you need?" with options: Haircut, Color, Both
- "Preferred time?" with options: Morning, Afternoon, Evening

Text Area: For longer responses, like "Tell us more about your project" or "Any special requests?"

For each field you add, you need to set:

Field Label: What the person sees ("Your Name", "Email Address", "Phone Number")

Field Name: What it's called in your automation (this can be the same as the label, but without spaces: "name", "email", "phone"). This is what you'll reference later when you use this information in other nodes.

Required: Toggle this ON if they must fill this out, or OFF if it's optional. I always make Name and Email required, but leave things like Phone optional.

Step 4: Customize the submit button (optional)

You can change what the submit button says. Default is "Submit" but you could change it to:

- "Send Message"
- "Request Quote"
- "Book Now"
- Whatever makes sense for your form

Step 5: Set the success message

After someone submits the form, what should they see? You can customize this message. Default is "Form submitted successfully" but you might want:

- "Thanks! We'll get back to you within 24 hours."
- "Your booking request has been received. Check your email for confirmation."
- "Got it! You're on the list."

This is important because it tells them their submission worked and what happens next.

Step 6: Get your form URL

Once you've set up your fields, you'll see two URLs appear (just like with Webhook):

Test URL: For testing while building **Production URL:** The real URL to share with people

Same rule applies - the production URL only works when your workflow is active (toggled ON).

Copy the production URL and that's the link you share with people. You can:

- Put it on your website as a "Contact" button
- Share it in your email signature
- Post it on social media
- Send it directly to people

When anyone clicks that link, they see your form, fill it out, click submit, and boom - your automation starts running with their information.

Step 7: Test your form

Before you share it with anyone, open the production URL yourself in a new browser tab. Fill out the form with test information. Click submit. Then go back to n8n and check your Executions list - you should see a new execution with the data you just submitted. This confirms everything is working.

Common Mistakes People Make

Mistake 1: Too many fields

I see people create forms with 15 different fields because they want to collect everything. Here's the problem: every field you add increases the chance someone will give up and not complete the form. Studies show that every additional field reduces completion rates by 10-20%. Only ask for what you absolutely need. You can always follow up and ask for more information later. For most use cases, Name + Email + one question is enough.

Mistake 2: Not marking important fields as required

If Email is optional and someone doesn't fill it in, how are you going to contact them? Make sure anything you truly need is marked as Required. At minimum, Email should always be required if you plan to respond to people.

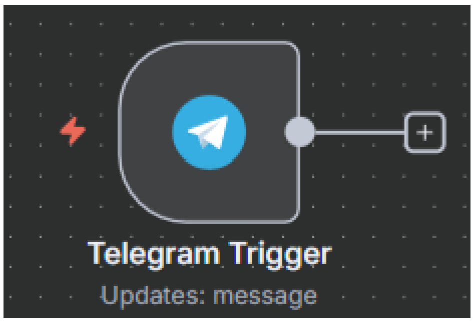
Mistake 3: Confusing field labels

If you label a field "Info" - what kind of info? Be specific. Instead of "Details" say "Tell us what you need" or "Describe your project." The clearer you are, the better responses you'll get.

Mistake 4: Not testing the whole flow

Just because the form works doesn't mean your whole automation works. Test the entire thing: fill out the form, submit it, then check if the data went into your Google Sheet correctly, if the email notification was sent, etc. Test the whole path before launching it to real users.

Telegram Trigger



Docs:
https://docs.n8n.io/integrations/builtin/trigger-nodes/n8n-nodes-base.telegramtrigger/

Telegram is a messaging app, like WhatsApp or iMessage, and you can create "bots" in Telegram - basically automated accounts that can receive messages and respond automatically. The Telegram Trigger connects n8n to a Telegram bot, so whenever someone sends a message to your bot, it immediately wakes up your automation and passes along whatever they sent.

The cool part is Telegram can handle more than just text. Someone can send:

- Text messages ("Hey, I need help")
- Voice messages (they record their voice)
- Photos or images
- Files and documents
- Location data

Your Telegram Trigger receives all of this and your automation can process it. This makes it incredibly powerful because you can control your automations from your phone, anywhere in the world, just by sending a message to your bot.

When You Would Actually Use This

Mobile control of automations: You're away from your computer but want to trigger an automation. You send a message to your Telegram bot saying "generate report" and your automation immediately creates and emails you a report. Or you send "post to Instagram" and it creates and posts content. Your phone becomes a remote control for your automations.

Voice note processing: You're driving and have an idea. You can't type, but you can talk. You send a voice message to your Telegram bot. Your automation converts your voice to text using AI, then uses that text to create a social media post, add it to your to-do list, or whatever you need. I use this all the time - way easier than typing on a phone.

AI chatbots: You want an AI assistant you can message anytime. You message your Telegram bot with questions like "What's on my calendar today?" or "Send me a summary of recent leads" and the AI Agent (connected to your Telegram Trigger) responds with the information. It's like having ChatGPT but connected to your actual business data.

Image-based automations: You take a photo with your phone and send it to your Telegram bot. The automation receives the image and can do things like: generate a product description from the photo, create social media posts about it, extract text from the image if it's a document, or use AI to enhance/edit the photo.

Content creation on the go: You're traveling but need to keep posting content. You send a message to your Telegram bot with a topic or image, and the automation generates a complete post with caption, posts it to all your social media, and confirms back to you in Telegram that it's done. You never need to open Instagram or TikTok.

The key advantage of Telegram Trigger over other triggers is that it's mobile-first and conversational. You're not filling out forms or clicking through a web interface - you're just messaging like you would message a friend, and your automation handles it.

How to Actually Set This Up (Step by Step)

This setup has two parts: first you create the Telegram bot (don't worry, it's easy), then you connect it to n8n.

Part 1: Create Your Telegram Bot

Step 1: Open Telegram and find BotFather

BotFather is Telegram's official bot for creating other bots. It sounds weird but that's really how it works.

Open your Telegram app and in the search bar at the top, type "BotFather" (one word, capital B and F). You'll see a bot with a blue checkmark - that's the official one. Click on it to open a chat with BotFather.

Step 2: Create a new bot

In your chat with BotFather, type this exact message: `/newbot` and send it.

BotFather will respond asking you to choose a name for your bot. This is the display name people will see - it can be anything you want. For example:

- "My AI Assistant"
- "Content Generator Bot"
- "Business Helper"

Type the name you want and send it.

Step 3: Choose a username

Next, BotFather asks you to choose a username. This is different from the name - it's like a handle that must be unique across all of Telegram. It must end with "bot" at the end. For example:

- myaiassistant_bot
- contentgen_bot

- bizhelper_bot

The username can't have spaces. If the username you want is taken, BotFather will tell you and you'll need to try a different one.

Step 4: Get your bot token

Once you choose a username that's available, BotFather will respond with a long message of congratulations and a line that says "Use this token to access the HTTP API:" followed by a long string of numbers and letters that looks something like this:

`123456789:ABCdefGHIjklMNOpqrsTUVwxyz`

This is your bot token - it's like a password that lets n8n connect to and control your bot. Copy this entire token. You'll need it in a moment.

Important: Keep this token secret. Anyone who has this token can control your bot and see the messages it receives. Don't share it publicly or post it anywhere.

Step 5: Start a chat with your bot (important!)

This is a step people often forget. In Telegram, search for your bot using the username you just created (like @myaiassistant_bot). Open a chat with it and click the "Start" button at the bottom. This activates the bot for your account. If you skip this step, your bot won't be able to send you messages later.

Part 2: Connect the Bot to n8n

Step 1: Add Telegram Trigger node

Back in n8n, click the plus (+) button and search for "Telegram Trigger". Add it to your canvas.

Step 2: Create credentials

When you click on the Telegram Trigger node, you'll see it needs "Credentials". Click on "Select Credentials" and then click "Create New Credentials."

A popup appears asking for your credentials. You'll see a field called "Access Token" - this is where you paste that long bot token you copied from BotFather. Paste it in, give your credentials a name like "My Telegram Bot", and click "Create."

Step 3: Choose what types of messages to receive

Back in the Telegram Trigger settings, you'll see "Updates" - this lets you choose what kinds of messages will trigger your automation. The options are:

message: Regular text messages, voice messages, photos - basically everything someone sends. This is the most common option because you usually want to respond to anything people send.

callback_query: For interactive buttons (advanced, ignore this for now)

edited_message: If someone edits a message they already sent

For most use cases, just leave it on "message" to capture everything.

Step 4: Additional settings (optional)

You'll see "Additional Fields" - you can mostly ignore these when starting out. The default settings work fine. But here's what they mean if you're curious:

Download Files: If someone sends you a file or photo, should n8n download it to process? Usually yes.

Binary Property: What to call the file data in your workflow. The default "data" is fine.

Step 5: Activate and test

Make sure your workflow is activated (toggle switch at the top is ON).

Now go to your Telegram app and send a message to your bot. Type anything - "Hello" or "Test" - and send it.

Go back to n8n and look at your Executions list (clock icon on the left sidebar). You should see a new execution that just ran. Click on it and you'll see the data that came from your Telegram message - your message text, your user ID, the timestamp, everything.

If you see that data, congratulations! Your Telegram Trigger is working.

Understanding the Data You Receive

When the Telegram Trigger receives a message, it gives you a bunch of information. Let me explain what you actually get so you know how to use it:

Message text: If someone typed a message, the text is in `{{ $json.message.text }}`

This is the actual words they typed. You'll use this to understand what they want.

Voice message: If someone sent a voice note, you'll get a file ID in `{{ $json.message.voice.file_id }}`

You can use this ID to download the voice file and convert it to text using AI.

Photo: If someone sent a photo, the file ID is in `{{ $json.message.photo }}`

You can download and process the image.

User information: You get details about who sent the message:

- Their name: `{{ $json.message.from.first_name }}`
- Their username: `{{ $json.message.from.username }}`
- Their user ID: `{{ $json.message.from.id }}`

This is useful for tracking who's using your bot or personalizing responses.

Chat ID: This is in `{{ $json.message.chat.id }}`

You need this if you want to send a reply back to the person. When you use a "Send Message" node later in your workflow, you'll use this chat ID to know where to send the reply.

Common Mistakes People Make

Mistake 1: Not clicking "Start" on the bot

After creating your bot, if you don't open a chat with it and click "Start", the bot can't initiate messages to you. It will receive messages but can't send replies. Always start a chat with your own bot first.

Mistake 2: Sharing the bot publicly before it's ready

Once your bot is created, anyone can search for it and message it. If you haven't finished your automation, they'll message it and nothing will happen. Keep the username private until your workflow is complete and tested.

Mistake 3: Token in plaintext

Some people paste their bot token directly into other nodes instead of using credentials. This is bad because if you export your workflow to share with someone, your token is visible. Always use the credentials system - it keeps tokens encrypted and separate from your workflow.

Mistake 4: Not handling different message types

People send text, voice, and photos. If your automation only expects text, it will break when someone sends a voice message. Use a Switch node after your Telegram Trigger to route different message types to different processing paths. For example:

- If text message → process text
- If voice message → convert to text first, then process
- If photo → extract information from image

Mistake 5: Workflow not activated

The Telegram Trigger only works when your workflow is active. If the toggle at the top is OFF, your bot will receive messages but nothing will happen. After testing, make sure you activate the workflow so it runs 24/7.

Twilio Trigger (For WhatsApp)

Docs:
https://docs.n8n.io/integrations/builtin/trigger-nodes/n8n-nodes-base.twiliotrigger/

```
Twilio is a communication platform that lets you connect WhatsApp
to your automations. The Twilio Trigger in n8n acts as a bridge -
when someone sends a WhatsApp message to your business number
(connected through Twilio), it immediately wakes up your
automation and passes along whatever they sent.

WhatsApp is more widely used than Telegram in many parts of the
world, especially for business communication. Your customers are
already on WhatsApp, so you don't need to ask them to download
another app. The Twilio Trigger lets you build automated WhatsApp
systems that can handle customer inquiries, bookings, support
requests - all through the messaging app they already use every
day.

Like Telegram, WhatsApp through Twilio can handle more than just
text:

    ● Text messages ("I want to book an appointment")
    ● Images (customer sends product photo)
    ● Documents (customer sends invoice or receipt)
    ● Voice messages (though less common on WhatsApp business)
    ● Location sharing

Your Twilio Trigger receives all of this and your automation can
process it, making it perfect for customer-facing business
automations.
```

When You Would Actually Use This

Customer service automation: A customer messages your WhatsApp business number with a question. Your AI Agent (connected to the Twilio Trigger) immediately responds with answers, checks order status, or routes them to the right department. They get instant responses 24/7 without waiting for business hours.

Appointment booking: Someone messages "I want to book a massage for tomorrow." Your automation understands the request, checks your Google Calendar for availability, offers time slots, and books the appointment - all through a natural WhatsApp conversation. No forms to fill out, no website to navigate.

Lead qualification: When someone messages your business on WhatsApp, your automation asks qualifying questions: "What service are you interested in? What's your budget? When do you need this?" It collects all the information and adds qualified leads to your CRM or Google Sheets automatically.

Order updates: Customer places an order. Your automation sends WhatsApp updates automatically: "Order confirmed," "Order shipped," "Out for delivery." They get real-time updates in the app they check constantly, not buried in email.

Support ticket system: Customer reports an issue via WhatsApp. Your automation logs it, assigns a ticket number, estimates resolution time, and keeps them updated on progress - all without human intervention until the issue needs hands-on attention.

The key advantage of WhatsApp (via Twilio Trigger) over other channels is: **your customers are already there**. They don't need to download anything new, create accounts, or learn a new platform. They just message your business like they'd message a friend.

How to Actually Set This Up (Step by Step)

This setup has three parts: create a Twilio account, get a WhatsApp-enabled number, then connect it to n8n.

Part 1: Create Your Twilio Account

Step 1: Sign up for Twilio

Go to twilio.com and click "Sign Up" (or "Try Twilio Free"). You'll need to provide:

- Email address

- Password
- Phone number (for verification)

Twilio will send you a verification code. Enter it to verify your account.

Step 2: Complete the initial setup

Twilio will ask you a few questions:

- "What do you want to build?" → Select "Messaging" or "Chatbots"
- "What's your use case?" → Select whatever fits (customer support, notifications, etc.)
- "Will you write code?" → You can select "No" (n8n handles the code)

These questions just help Twilio show you relevant resources - your answers don't limit what you can build.

Step 3: Navigate to WhatsApp section

Once logged in, look for the left sidebar menu. Find "Messaging" and click on it. Then look for "Try it out" → "Send a WhatsApp message."

Alternatively, go directly to: Console → Develop → Messaging → Try WhatsApp

Part 2: Get Your WhatsApp Sandbox (For Testing)

Before you can use WhatsApp with your own business number, Twilio provides a "sandbox" - a test environment where you can build and test your automation for free.

Step 1: Join the Twilio WhatsApp Sandbox

Twilio will show you a WhatsApp number (usually starts with +1 415...) and a code (usually "join [something-something]").

Example: "Send 'join happy-dolphin' to +1 415 523 8886"

Step 2: Add the Twilio sandbox to your WhatsApp

Open WhatsApp on your phone. Start a new message to the Twilio number shown (like +1 415 523 8886). Send the exact message they tell you (like "join happy-dolphin").

You'll get a confirmation message from Twilio saying you're connected to the sandbox.

Important: This sandbox is for testing only. Messages can only be sent to/from phone numbers that have "joined" the sandbox. For production use with real customers, you'll need to set up a proper business number (covered in "Going to Production" below).

Step 3: Get your Account SID and Auth Token

These are like your username and password for Twilio's API. You'll need them to connect n8n.

In the Twilio console, look at the top right. You should see your "Account Info" with:

- **Account SID:** A long string starting with "AC..." (like AC1234567890abcdef...)
- **Auth Token:** Click "View" to reveal it (another long string)

Copy both of these. Keep them secret - they give full access to your Twilio account.

Part 3: Connect Twilio to n8n

Step 1: Add Twilio Trigger node

In n8n, click the plus (+) button and search for "Twilio". You'll see "Twilio Trigger" - add it to your canvas.

Step 2: Create credentials

Click on the Twilio Trigger node. You'll see it needs "Credentials." Click "Select Credentials" → "Create New Credentials."

A popup appears asking for:

- **Account SID:** Paste the Account SID you copied from Twilio
- **Auth Token:** Paste the Auth Token you copied from Twilio

Give your credentials a name like "My Twilio Account" and click "Create."

Step 3: Configure the Trigger

Back in the Twilio Trigger settings, you'll see:

Webhook URL: This is the URL that Twilio will send data to when someone messages your WhatsApp number. Copy this URL - you'll need it in a moment.

There are two URLs shown:

- **Test URL:** For testing while building
- **Production URL:** For when your workflow is live

Copy the **Production URL** (you'll use Test URL only during initial setup).

Step 4: Set the Webhook in Twilio

Go back to the Twilio console in your browser.

Navigate to: Messaging → Try WhatsApp → Sandbox Settings

Look for "When a message comes in" - this is where you tell Twilio where to send incoming messages.

Paste your n8n Webhook URL (the Production URL you just copied) into this field.

Make sure the dropdown next to it is set to "HTTP POST" (this should be the default).

Click "Save" at the bottom.

Step 5: Test it

Make sure your n8n workflow is activated (toggle switch at the top is ON).

Go to WhatsApp on your phone. Send a message to the Twilio sandbox number (the one you joined earlier). Type anything - "Hello" or "Test message."

Go back to n8n. Look at your Executions list (clock icon on the left sidebar). You should see a new execution that just ran.

Click on it and you'll see the data from your WhatsApp message - the message text, the sender's phone number, timestamp, everything.

If you see that data: **Success!** Your Twilio Trigger is working.

Understanding the Data You Receive

When the Twilio Trigger receives a WhatsApp message, here's what data you get:

Message text: The actual message content is in `{{ $json.Body }}`

This is what the person typed. Use this to understand what they're asking for.

Sender's phone number: In `{{ $json.From }}`

Format will be like: whatsapp:+4915123456789

This identifies who sent the message. You can use it to track conversations, look up customer info, or reply to them.

Your business number: In `{{ $json.To }}`

The WhatsApp number they messaged (your Twilio number).

Message SID: In `{{ $json.MessageSid }}`

A unique ID for this specific message. Useful for logging or tracking message history.

Media (if sent): If someone sends an image or document:

- `{{ $json.NumMedia }}` tells you how many media files were sent
- `{{ $json.MediaUrl0 }}` contains the URL to download the first media file
- `{{ $json.MediaContentType0 }}` tells you the file type (image/jpeg, application/pdf, etc.)

Account info:

- `{{ $json.AccountSid }}` - Your Twilio Account SID
- Confirms which Twilio account this came through

Common Mistakes People Make

Mistake 1: Using Test URL in production

The Test URL only works while you're actively working on the workflow. If you close n8n or navigate away, it stops working. Always use the Production URL in Twilio's webhook settings for real use.

Mistake 2: Workflow not activated

Even with the webhook set up correctly, if your n8n workflow isn't active (toggle is OFF), nothing happens when messages arrive. Always activate your workflow after setup.

Mistake 3: Sandbox limitations

The Twilio sandbox only works with phone numbers that "joined" it. You can't send messages to random customers. For real business use, you need to go to production (see below).

Mistake 4: Not handling "From" number format

The phone number comes formatted as "whatsapp:+4915123456789" (with "whatsapp:" prefix). If you're saving it to a database or using it in other nodes, you might need to remove that prefix. Use a Code node or Set node to clean it:

javascript

```
const phoneNumber = $json.From.replace('whatsapp:', '');

return { phoneNumber };
```

Mistake 5: Forgetting to reply

If someone messages your WhatsApp and your automation doesn't respond, they don't know if it worked. Always include a response, even if it's just "Message received, processing..." so people know the bot is working.

Mistake 6: Credentials exposed

Never share your Account SID and Auth Token publicly. If you export your workflow to share, make sure credentials aren't included. n8n's credential system keeps them separate, but be careful when sharing screenshots or code.

Going to Production (Real WhatsApp Business Number)

The sandbox is great for testing, but for real customer-facing use, you need a proper WhatsApp Business number. Here's what's involved:

Requirements:

- A dedicated phone number (can't use your personal WhatsApp number)
- Business verification with Meta (Facebook)
- WhatsApp Business API approval

Steps (simplified):

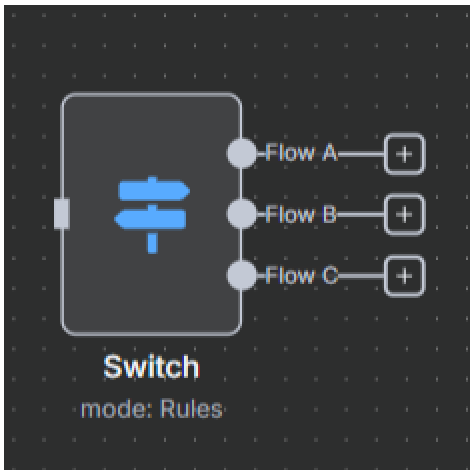
1. In Twilio console, go to Messaging → WhatsApp → Senders
2. Click "New Sender" or "Request Access"
3. Follow Twilio's process to connect a phone number to WhatsApp Business API
4. Submit business information to Meta for verification (can take days to weeks)
5. Once approved, update your webhook URL in the production WhatsApp sender settings

Cost:

- Twilio WhatsApp messages have costs (typically €0.005-0.02 per message depending on country)
- The sandbox is free for testing

Note: Production setup is more complex and takes time. Start with the sandbox to build and test your automation. Once it works perfectly, then go through production approval.

Switch Node



Docs:
https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.switch/#switch

Imagine you're sorting mail. You have a pile of envelopes and you need to put each one in the right box: bills go in one box, letters from friends go in another box, and advertisements go in the trash. You look at each envelope, decide what type it is, and send it to the correct place.

That's exactly what the Switch node does with your data. It looks at information coming through your automation and asks "what type of thing is this?" Then based on what it finds, it sends that data down different paths to be processed differently.

The key difference between Switch and other nodes is that Switch can have as many outputs as you want - three paths, five paths, ten paths, whatever you need. Each path is for a different category or type of data.

When You Would Actually Use This

Let me give you real examples so this makes sense:

Handling different message types in Telegram: Someone messages your Telegram bot. Did they send text, a voice message, or a photo? These all need to be handled differently. The Switch node checks what type of message it is:

- If it's text → send it directly to the AI to process

- If it's voice → first convert voice to text, THEN send to AI
- If it's photo → extract information from the image first, THEN process

Without the Switch node, you'd have one path trying to handle all three types, which would break constantly.

Categorizing leads by interest: Someone fills out your form asking about services. In the form they selected "Web Design," "SEO," or "Social Media Management." The Switch node checks which service they picked:

- If Web Design → send email with web design portfolio and pricing
- If SEO → send different email with SEO case studies
- If Social Media → send social media examples and packages

Each category gets a completely different email because they're interested in different things.

Routing customer support tickets: You receive support messages. The Switch node checks keywords in the message:

- If message contains "cancel" or "refund" → high priority path, send to manager immediately
- If message contains "how do I" or "question" → send to knowledge base bot
- If message contains "bug" or "not working" → create bug ticket, send to tech team
- Everything else → standard support path

Prioritizing by value: You get new leads. The Switch node checks how much they said their budget is:

- If budget over $10,000 → high-value path, send to senior salesperson
- If budget $1,000-$10,000 → mid-tier path, standard sales flow
- If budget under $1,000 → low-priority path, send automated resources

Email categorization: Your automation reads incoming emails. The Switch node uses AI to categorize them:

- If it's a sale inquiry → sales path
- If it's customer support → support path

- If it's a job application → HR path
- If it's spam → delete path

The common thread in all these examples is: you have one piece of data coming in, but it could be several different types of things, and each type needs to be handled completely differently. That's when you use Switch.

How to Actually Set This Up (Step by Step)

Step 1: Add the Switch node

Click the plus (+) button in n8n, search for "Switch", and add it to your canvas. Place it after the node that's giving you the data you need to evaluate (like after a Telegram Trigger, after a Form Trigger, etc.).

Step 2: Understand the Mode setting

When you click on the Switch node, the first thing you see is "Mode". There are a few options but the one you'll use 99% of the time is "Rules". This lets you create custom conditions for each path.

Select "Rules" mode.

Step 3: Create your first routing rule

Now you'll see "Rules" with a section to add conditions. Click "Add Routing Rule" to create your first path.

Let's use a real example: you want to route Telegram messages differently based on whether they're text or voice.

For your first rule:

1. Click "Add Condition"
2. In the condition, you need to specify what to check. Click in the field and you'll see you can type {{ and it will show you available data
3. Type: `{{ $json.message.text }}`
4. For the operator, choose "exists" (this checks if text exists at all)
5. Below that, you'll see "Output" - this names the path. Type something descriptive like "Text Message"

What this rule says is: "If this message has text in it, send it down the 'Text Message' path."

Step 4: Create your second routing rule

Click "Add Routing Rule" again to create another path.

For voice messages:

1. Add Condition
2. Type: `{{ $json.message.voice.file_id }}`
3. Operator: "exists"
4. Output name: "Voice Message"

Step 5: Add as many rules as you need

You can keep adding more routing rules. If you wanted to also handle photos:

1. Add Routing Rule
2. Condition: `{{ $json.message.photo }}` exists
3. Output: "Photo"

The Switch node will now have three outputs: one for text, one for voice, one for photos.

Step 6: Connect the outputs to different paths

This is where the magic happens. Each output of your Switch node can connect to completely different nodes.

From the "Text Message" output, you might connect directly to an AI node because text is ready to process.

From the "Voice Message" output, you might connect to a node that converts voice to text first, THEN to the AI node.

From the "Photo" output, you might connect to an AI vision node that describes the image first.

Each path does what's appropriate for that type of data.

Step 7: What if nothing matches?

You'll notice the Switch node also has a "Fallback" output at the bottom. If none of your rules match (like someone sends a sticker or a location that you didn't create rules for), the data goes to the Fallback output. You can connect this to a simple node that sends a message like "Sorry, I can only handle text, voice, and photos."

Understanding Conditions and Operators

This is where people get confused, so let me explain all the operators you can use when creating rules:

exists: Checks if something is there at all. Use this when you just need to know "does this field have any data?"

Example: `{{ $json.email }}` exists = true if an email address was provided, regardless of what the email is

does not exist: The opposite. True if the field is empty or missing.

equals: Checks if something exactly matches a specific value.

Example: `{{ $json.status }}` equals "approved" = only true if status is exactly the word "approved"

not equals: True if it's anything OTHER than the specified value.

contains: Checks if a text field includes certain words anywhere inside it.

Example: `{{ $json.message }}` contains "urgent" = true if the word "urgent" appears anywhere in the message, like "This is urgent" or "urgent matter" or "urgently need help"

does not contain: True if the specified text is NOT found anywhere.

starts with: Checks if text begins with specific characters.

Example: `{{ $json.command }}` starts with "/" = true for "/start" or "/help" but not for "start/"

ends with: Checks if text finishes with specific characters.

greater than / less than: For numbers. Checks if a number is bigger or smaller than your specified value.

Example: `{{ $json.budget }}` greater than 5000 = true if budget is 5001 or higher

is empty / is not empty: Similar to exists but specifically checks for empty strings.

Choose the operator based on what you're checking for. Most of the time you'll use "exists", "equals", or "contains" because those cover 90% of use cases.

Using Multiple Conditions in One Rule (AND vs OR)

Sometimes one condition isn't enough. You might need to check multiple things to decide which path to use.

At the bottom of each rule, you'll see "Combinator" with options for "AND" and "OR". Here's what these mean:

AND: ALL conditions must be true for this rule to match.

Example: You want high-priority leads who:

- Have a budget over $10,000 AND
- Selected "Enterprise" plan AND
- Are located in USA

If any one of these is false, they don't go down this path.

OR: ANY condition being true is enough to match this rule.

Example: You want to route urgent messages that:

- Contain "urgent" OR
- Contain "emergency" OR
- Contain "asap"

If the message has even one of these words, it goes down the urgent path.

Use AND when you're being picky (must meet all criteria). Use OR when you're being inclusive (meets any criteria).

Common Mistakes People Make

Mistake 1: Overlapping rules

Someone creates rules that could match the same data. For example:

- Rule 1: If message contains "help"

- Rule 2: If message contains "please"

What happens if someone sends "Please help"? Both rules match! The Switch node will use whichever rule comes first in your list. This can cause confusing behavior. Make your rules mutually exclusive (meaning only one can be true at a time) when possible.

Mistake 2: Not handling the fallback

You create rules for three scenarios but there's actually a fourth scenario you didn't think of. Data hits the Switch, doesn't match any rule, goes to Fallback, and you didn't connect anything to Fallback. The automation just stops there and nothing happens. Always connect the Fallback output to something - even if it's just a node that logs "unexpected data type received."

Mistake 3: Wrong data path

You're checking `{{ $json.message.text }}` but the actual data is in `{{ $json.body.message }}`. The rule never matches because you're looking in the wrong place. Always check your data structure first by looking at the input data from the previous node. Click on the previous node, look at what data it outputs, and make sure your Switch is checking the correct path to that data.

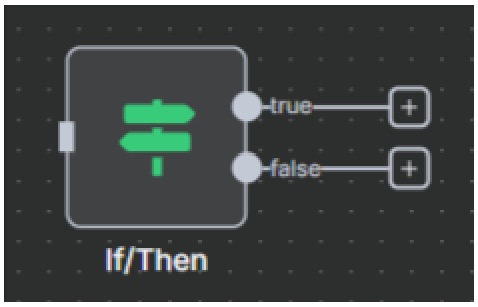
Mistake 4: Case sensitivity

You're checking if `{{ $json.status }}` equals "Approved" but the actual data says "approved" (lowercase a). These don't match! Text comparisons are case-sensitive by default. Either use "contains" which is more forgiving, or make sure your text matches exactly including capitalization.

Mistake 5: Too many rules

Someone creates 15 different routing rules. This works but becomes a nightmare to manage. If you find yourself creating more than 5-6 rules, consider if there's a simpler way to structure your logic, or use multiple Switch nodes in sequence to make it more manageable.

IF/Then Node



Doc: https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.if/#if

The IF node is like a bouncer at a club checking IDs. It asks one simple question: "Does this meet my requirement?" If yes, you go in one door. If no, you go in a different door. That's it. Just two options: yes or no, true or false, pass or fail.

Think about decisions you make in real life that are binary (only two options):

- "Is it raining? If yes, bring umbrella. If no, don't bring umbrella."
- "Do I have milk? If yes, make coffee. If no, go buy milk first."
- "Is the person over 18? If yes, allowed. If no, not allowed."

The IF node does exactly this in your automation. It checks something, and based on whether it's true or false, sends your data down one of two different paths.

When You Would Actually Use This

Let me show you real situations where IF nodes are perfect:

Validating email addresses: Someone filled out your form. Before you do anything else, you want to check: did they actually provide an email?

- IF email exists → continue with the automation, add them to your list, send them a welcome email
- IF email does not exist → stop the automation, send yourself an alert that someone submitted a form without an email

Checking if payment succeeded: You processed a payment. The payment processor tells you if it worked or failed.

- IF payment successful → send the product, add customer to your database, send receipt
- IF payment failed → send customer an email saying payment didn't go through, ask them to try again

Filtering by budget: Someone requested a quote and told you their budget.

- IF budget is over $5,000 → send to your premium sales process
- IF budget is under $5,000 → send automated response with standard pricing

Verifying form completeness: Before processing a form submission, check if all required fields are filled.

- IF all required fields have data → process the submission
- IF any required field is empty → send error message, don't process

Checking business hours: Someone wants to book a call.

- IF current time is between 9 AM and 5 PM → show them available times today
- IF current time is outside business hours → show them times for tomorrow

Confirming actions: Your voice agent talked to a customer and asked "Do you want to confirm this booking?"

- IF they said yes/confirmed → book the appointment, send confirmation
- IF they said no/declined → don't book, ask what they'd like to change

The key pattern here is: you need to make a decision that has exactly two outcomes. Not three, not ten - just two. Pass or fail. Yes or no. True or false. That's what IF is for.

The Difference Between IF and Switch (This Confuses Everyone)

I'm going to explain this really clearly because this is the number one thing people ask me:

IF Node:

- Two outputs only: True and False
- Use when you have a binary decision (yes/no question)
- Simple and clean for single checks
- Example: "Is this person a premium user? Yes or No"

Switch Node:

- Multiple outputs: as many as you want (3, 5, 10, whatever)
- Use when you're categorizing into multiple groups
- Better for complex routing
- Example: "What type of user is this? Free, Pro, or Enterprise"

Here's the clearest way to decide which to use:

Count how many different outcomes you need:

- **2 outcomes** = use IF
- **3+ outcomes** = use Switch

Let me give you a concrete example of the same scenario using each:

Using IF: "Is the lead's budget over $10,000?"

- True path → high-value process
- False path → standard process

Using Switch for the same thing: You could technically use Switch with:

- Rule 1: budget over $10,000 → high-value process
- Fallback → standard process

Both work, but IF is simpler and clearer when you only have two options.

When Switch is clearly better: "What's the lead's budget tier?"

- Rule 1: budget over $50,000 → enterprise process
- Rule 2: budget $10,000-$50,000 → professional process
- Rule 3: budget $1,000-$10,000 → small business process

- Fallback: budget under $1,000 → self-service process

You can't do this with IF because IF only has two paths. You'd need multiple IF nodes stacked together, which gets messy fast.

Rule of thumb: IF for yes/no decisions. Switch for categorization.

How to Actually Set This Up (Step by Step)

Step 1: Add the IF node

Click the plus (+) button, search for "IF", and add it to your canvas. Connect it after the node that has the data you want to check.

Step 2: Add your condition

Click on the IF node. You'll see "Conditions" with the option to "Add Condition". Click it.

Now you need to tell it what to check. This has three parts:

Part 1: What field are you checking?

Click in the first field and you can type `{{` to see available data. Let's say you want to check if an email exists. You'd type:

```
{{ $json.email }}
```

This tells the IF node "look at the email field in the data that just came through."

Part 2: What operator do you want to use?

The middle dropdown is your operator - it's how you want to check the data. The most common ones are:

is not empty: Checks if the field has any data at all. Use this to verify someone filled out a required field.

equals: Checks if the value exactly matches something specific. Use this for things like status checks ("equals approved") or yes/no questions ("equals yes").

contains: Checks if text includes certain words. Use this for searching within messages or descriptions.

greater than / less than: For numbers. Use this for budget checks, age verification, quantity limits.

exists: Similar to "is not empty" but also checks if the field itself is present in the data.

Choose the operator that makes sense for what you're checking.

Part 3: What value are you comparing to? (if needed)

Some operators need a comparison value. For example:

If you chose "equals", you need to say what it should equal. Like `{{ $json.status }}` equals "approved" - you need to type "approved" in the third field.

If you chose "greater than", you need to say what number. Like `{{ $json.budget }}` greater than 5000 - you need to type 5000 in the third field.

If you chose "exists" or "is not empty", you don't need a third field - these operators just check if something is there or not.

Step 3: Add more conditions if needed (optional)

Sometimes one check isn't enough. Click "Add Condition" again to add more checks.

Then you need to decide if this is an AND or OR situation:

AND (all conditions must be true):

Use this when you're being strict. For example, you want leads who:

- Have an email address AND
- Have a budget over $1,000 AND
- Are in the USA

All three must be true for the data to go down the True path. If even one is false, it goes to False path.

OR (any condition being true is enough):

Use this when you're being flexible. For example, you want to catch urgent messages that:

- Contain "urgent" OR

- Contain "emergency" OR
- Contain "asap"

If any one of these is true, it goes to the True path. Only if none are true does it go to False.

At the bottom of your conditions, you'll see "Combine" - select "AND" or "OR" based on which logic you need.

Step 4: Connect the True and False paths

Now your IF node has two outputs:

True output: Connect this to whatever should happen when the condition is met. This could be adding the data to your database, sending an email, processing further, etc.

False output: Connect this to whatever should happen when the condition is NOT met. This could be sending an error message, logging the issue, redirecting to a different process, or just stopping the automation.

Important: Always connect BOTH outputs. Don't leave the False path hanging. Even if False means "do nothing," connect it to a node that explicitly ends the workflow or logs that nothing was done. This makes it clear the path was intentional, not forgotten.

Step 5: Test both paths

Before you rely on this automation, test it with data that will go True AND data that will go False. Make sure both paths work correctly. It's really common to build the True path carefully but forget to test what happens on the False path, and then be surprised when something fails.

Common Mistakes People Make

Mistake 1: Only handling the True path

This is the most common mistake. Someone builds a beautiful True path - if the email exists, add them to the database, send welcome email, log it, everything perfect. But they don't connect the False path at all. So when someone submits a form without an email, the automation just stops dead with no error message, no logging, nothing. You don't even know it happened.

Always handle both paths. The False path is often where you send error alerts, log problems, or send "Please complete your information" messages.

Mistake 2: Wrong operator choice

Someone wants to check if a field has data, so they use "equals" and check if it equals... what? They're not sure. The correct operator here is "is not empty" or "exists". These check for the presence of data without needing to know the specific value.

Use "equals" only when you know the exact value you're expecting, like checking if status equals "approved" or if answer equals "yes".

Mistake 3: Case sensitivity

You check if `{{ $json.response }}` equals "Yes" but the actual data says "yes" or "YES". These don't match because text comparison is case-sensitive. Either:

- Use "contains" instead which is more forgiving
- Convert everything to lowercase first using a function
- Make sure your comparison values match exactly including capitalization

Mistake 4: Using IF when Switch would be better

Someone needs to route data to one of four different paths based on priority level. They stack multiple IF nodes:

- IF high priority → Path A
- ELSE IF medium priority → Path B
- ELSE IF low priority → Path C
- ELSE → Path D

This works but it's messy and hard to read. Switch would be much cleaner for this because it's designed for multiple paths.

Mistake 5: Complex nested IFs

Someone builds IF inside of IF inside of IF. After three layers, even they can't follow the logic anymore. If you find yourself nesting IFs, stop and consider:

- Can this be simplified?
- Should I use a Switch instead?

- Should I use code to handle the logic?

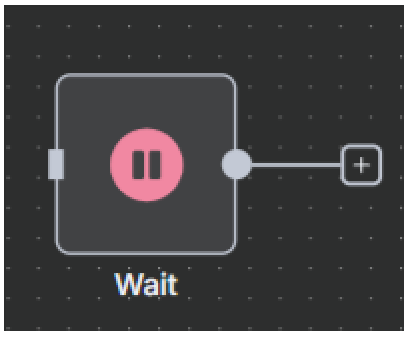
Generally, if you need more than two IFs connected in sequence, there's probably a cleaner way.

Mistake 6: Not understanding the data structure

You're checking `{{ $json.email }}` but the actual data structure is `{{ $json.body.contact.email }}` because the data is nested deeper. The IF node never finds the email because you're looking in the wrong place.

Always check your data structure first. Click on the node before the IF node, look at the output data, and trace exactly where the field you want is located. Copy the exact path to that field.

Wait Node



Doc:
https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.wait/#wait

The Wait node is like a pause button for your automation. When your data reaches a Wait node, the automation stops and waits for a specific amount of time before continuing to the next step. It's that simple.

Think about situations in real life where waiting is important:

- You put cookies in the oven and wait 12 minutes before taking them out
- You send an email to someone and wait 24 hours before following up
- You plant seeds and wait 2 weeks before checking for sprouts

The Wait node does this in your automation. It creates delays between actions. This might sound simple and maybe even useless, but it's actually incredibly powerful for creating natural, well-timed automations.

When You Would Actually Use This

Let me show you real scenarios where Wait nodes are essential:

Sending reminder messages: Someone books an appointment for tomorrow at 3 PM. You want to remind them an hour before. Your automation:

1. Books the appointment
2. Wait 23 hours
3. Send reminder: "Your appointment is in 1 hour!"
4. Wait 1 hour

5. Send: "Your appointment is starting now. Here's the meeting link."

Without Wait nodes, all these messages would send immediately when they booked, which would be useless.

Multi-step follow-up sequences: Someone downloads your free guide. You want to nurture them over time:

1. Immediately send the download link
2. Wait 2 days
3. Send: "Did you enjoy the guide? Here's a related article"
4. Wait 3 days
5. Send: "Ready to take it further? Check out our course"
6. Wait 5 days
7. Send: "Last chance for early bird discount"

This creates a natural sequence that feels like a real person following up, not a robot spamming.

Rate limiting API calls: You need to process 100 items but the API you're using only allows 1 request per second. Your automation:

1. Process item
2. Wait 1 second
3. Process next item
4. Wait 1 second
5. Continue...

This prevents you from getting blocked for making too many requests too fast.

Building suspense in automated interactions: Your AI chatbot is conversing with someone. You want it to feel natural, not instant. After the person sends a message:

1. Wait 2 seconds (simulates "typing...")
2. Send response

Those 2 seconds make it feel like a real person is thinking and typing, not a bot responding instantly.

Delayed content publishing: You create 7 social media posts at once on Sunday. But you want them to go out throughout the week:

1. Post Monday content
2. Wait 24 hours

3. Post Tuesday content
4. Wait 24 hours
5. Continue...

Giving people time to take action: You send someone a payment link. You want to give them time to pay before taking further action:

1. Send payment link
2. Wait 2 hours
3. Check if payment was received
4. If not, send reminder
5. Wait 24 hours
6. Check again
7. If still not paid, cancel order

The key thing to understand: Wait is about timing and pacing. It makes your automations feel more natural and less robotic. It also solves practical problems like rate limits and giving people time to respond.

How to Actually Set This Up (Step by Step)

Step 1: Add the Wait node

Click the plus (+) button, search for "Wait", and add it to your canvas wherever you need a delay. It goes between the action that just happened and the action that should happen after waiting.

Step 2: Choose your wait type

When you click on the Wait node, you'll see several options for how long to wait. Let me explain each one:

Option 1: Wait for Duration (most common)

This is the simplest and most used option. You specify "wait for X amount of time" and that's it.

When you select this, you'll see two fields:

Amount: How many units to wait (like 5, or 30, or 2)

Unit: What unit of time

- Seconds: for very short waits (useful for rate limiting or simulating typing)

- Minutes: for short delays (waiting a few minutes between steps)
- Hours: for longer delays (waiting hours before follow-up)
- Days: for multi-day sequences

Examples:

- Wait 30 seconds = Amount: 30, Unit: Seconds
- Wait 2 hours = Amount: 2, Unit: Hours
- Wait 3 days = Amount: 3, Unit: Days

Option 2: Wait Until Specific Date/Time

This is for when you want to wait until a particular moment arrives. Instead of "wait for 2 hours," this is "wait until 3:00 PM on Friday."

You would use this when:

- Someone books an appointment and you have the exact appointment time - you want to send a reminder at that specific time
- You're scheduling content to post at a specific time
- An event happens at a known future date and you want to trigger something then

When you select this option, you'll see a field where you can specify the exact datetime. You can:

- Type a specific date and time manually
- Use data from a previous node, like `{{ $json.appointmentTime }}`

For example, if someone books an appointment and the booking system gives you the appointment time in a field called `appointmentTime`, you can tell the Wait node:

"Wait until `{{ $json.appointmentTime }}`"

The automation will pause until that exact moment arrives.

Step 3: Understanding what happens during the wait

This is important to understand: when data hits a Wait node, the automation pauses but n8n doesn't forget about it. n8n stores that data and sets a timer. When the timer expires, n8n wakes up the automation and continues with the next node.

This means:

- Your n8n instance must be running when the wait time ends
- For cloud n8n, this is automatic - it's always running
- For self-hosted n8n, make sure your server doesn't shut down during waits

Step 4: Connect what happens after the wait

After the Wait node, connect whatever should happen next. This could be:

- Send a message node
- Another processing step
- A check to see if something changed during the wait
- Whatever your automation needs to do after waiting

Common Mistakes People Make

Mistake 1: Using Wait when Schedule Trigger is better

Someone wants their automation to post to social media every day at 9 AM. They try to build:

1. Run automation
2. Post content
3. Wait 24 hours
4. Post content
5. Wait 24 hours
6. Continue forever...

This doesn't work well because if anything breaks the loop, the whole schedule breaks. Use Schedule Trigger for predictable recurring tasks. Use Wait for delays within a single automation run.

Mistake 2: Very long waits on self-hosted n8n

Someone is self-hosting n8n on their laptop and creates a Wait for 30 days. Then they close their laptop or shut it down. When the 30 days pass, n8n isn't running so the automation never continues.

For very long waits (more than a few days), make sure your n8n instance will be running when the wait ends. Cloud n8n handles this automatically. Self-hosted n8n needs to be on a server that stays running.

Mistake 3: Not accounting for timezones

Someone sets "Wait until" a specific time but doesn't realize their n8n instance is in a different timezone. They want to send something at 9 AM their local time, but it goes out at 3 PM because the server is in a different timezone.

Always check what timezone your n8n instance uses. If you're using cloud n8n, it might be UTC (Universal Time). You may need to adjust your times accordingly.

Mistake 4: Waiting between every single step unnecessarily

Someone adds Wait nodes everywhere because they think it makes their automation better. They wait 5 seconds between every single node. This just makes their automation slow for no reason.

Only use Wait when there's an actual reason to wait:

- To comply with rate limits
- To give humans time to take action
- To create natural pacing in communications
- To wait for a specific time to arrive

Don't add waits just because. Every wait makes your automation slower.

Mistake 5: Not handling what might change during the wait

Someone sends a payment link, waits 2 hours, then automatically sends a reminder. But what if the person already paid during those 2 hours? Now they get a reminder saying they haven't paid, even though they have.

After a Wait, always check the current state before taking action. Don't assume nothing changed during the wait.

Better flow:

1. Send payment link
2. Wait 2 hours
3. Check if payment was received during the wait
4. IF payment received → send thank you

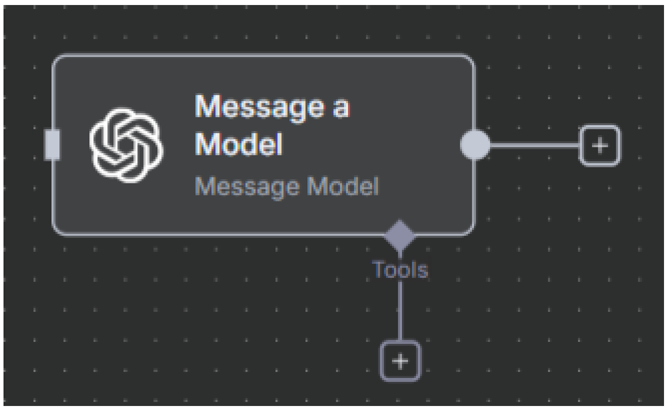
5. IF payment not received → send reminder

Mistake 6: Forgetting workflow must be active

Wait nodes only work when your workflow is activated (toggle is ON). If your workflow is inactive, when the wait time ends, nothing happens because the workflow isn't running.

After building an automation with Wait nodes, make sure to activate it so the waits actually trigger their next steps.

OpenAI / Message a Model Node



Doc:
https://docs.n8n.io/integrations/builtin/app-nodes/n8n-nodes-langchain.openai/text-operations/#openai-text-operations

This node lets you send a message to ChatGPT (or other AI models like Claude) and get an intelligent response back. Think of it like having a conversation with ChatGPT, but it happens automatically inside your automation instead of you typing into the ChatGPT website.

You write a prompt (instructions), the AI reads it, thinks about it, and gives you back text. That text could be:

- A social media caption
- An email response
- A summary of a long document
- A product description
- A translation
- An answer to a question
- Anything that requires writing or thinking

The key thing to understand is this: the AI doesn't DO anything except think and write. It can't call other tools, can't access your calendar, can't send emails, can't check databases. It just reads your prompt and writes a response. That's it. But that simple capability is incredibly powerful when used correctly.

When You Would Actually Use This

Let me show you real situations where this node is perfect:

Writing social media captions: You have a product photo and want to post it to Instagram. Instead of manually writing the caption, you send the AI: "Write an Instagram caption for [product name]. It's [description]. Make it engaging and include emoji."

The AI writes: "Loving our new bamboo sunglasses 🕶️✨ Eco-friendly, lightweight, and stylish - perfect for those sunny days! Who else is ready for summer? 🌴☀️"

You get that back automatically, then your automation posts it.

Generating email responses: Someone sends you a support question: "How do I reset my password?"

You send to AI: "Write a helpful response to this customer question: [their question]. Be friendly and professional."

AI writes a complete, polite response that your automation sends back to them.

Summarizing long content: You receive a 10-page report every week. Instead of reading the whole thing, you send it to AI: "Summarize this report in 3 bullet points focusing on key metrics and action items."

AI gives you:

- Revenue increased 12% vs last quarter
- Customer satisfaction scores dropped in region B (investigate causes)
- New product launch scheduled for next month

Creating variations of content: You wrote one product description. You need 5 different versions for A/B testing. You send to AI: "Rewrite this product description 5 different ways, each with a different tone: professional, casual, humorous, luxury, and minimalist."

AI gives you 5 completely different versions instantly.

Extracting information from unstructured text: Someone filled out a form with a giant paragraph of text rambling about what they need. You send to AI: "Extract the key information from this message: their industry, their budget, their timeline, and their main pain point. Format as: Industry: [X], Budget: [Y], Timeline: [Z], Pain Point: [W]"

AI analyzes the paragraph and gives you back structured data.

Translations: You need to translate your content into 5 languages. You send to AI: "Translate this text into Spanish, French, German, Italian, and Portuguese. Return as: Spanish: [text], French: [text], etc."

AI translates it all instantly.

Writing scripts or outlines: You're creating a video. You send to AI: "Create a 60-second video script about [topic]. Include: hook (5 seconds), main points (45 seconds), call to action (10 seconds)."

AI gives you a complete script formatted exactly how you asked.

The pattern here is: any time you need AI to read something and write something back, this is the node. It's NOT for having the AI take actions or use tools - that's what AI Agent is for (we'll cover that next). This is pure prompt-in, text-out.

How to Actually Set This Up (Step by Step)

Step 1: Add the OpenAI node

Click the plus (+) button, search for "OpenAI" and add the node to your canvas. (You can also use Claude, Gemini, or other AI models - the setup is similar. I'll use OpenAI as the example but the concepts apply to all.)

Step 2: Set up your API credentials

AI models aren't free - you need an API key from OpenAI (or whichever AI provider you're using). Here's how to get one:

Go to platform.openai.com and create an account. Go to the API keys section. Click "Create new secret key". Copy the key (it looks like: sk-abc123xyz...). This is your API key.

Back in n8n, in your OpenAI node, click on "Credentials" and select "Create New Credentials". Paste your API key in the field. Give it a name like "OpenAI API". Click Create.

Important: Keep this key secret. Anyone with this key can use your OpenAI account and you'll be charged for their usage. Never share it publicly or commit it to code repositories.

Step 3: Choose "Message a Model" as the resource

In the OpenAI node settings, you'll see "Resource" dropdown. Select "Message a Model". This is the option for sending prompts and getting text responses back.

Step 4: Select your AI model

You'll see "Model" dropdown with options like:

- gpt-4o: The smartest, most capable, but more expensive
- gpt-4o-mini: Good balance of quality and cost (this is what I use most)
- gpt-3.5-turbo: Faster and cheaper but less capable

For most tasks, gpt-4o-mini is perfect. It's smart enough for captions, emails, summaries - basically everything except really complex reasoning tasks. Save gpt-4o for when you need the absolute best quality.

Step 5: Write your prompt

This is the most important part. Your prompt is the instructions you give the AI. The better your prompt, the better your results.

In the "Messages" section, click "Add Message". You'll see you can add different types of messages:

System Message (optional but recommended): This sets the AI's role and behavior. Think of it as giving the AI its job description.

Examples:

- "You are an expert Instagram marketing copywriter"
- "You are a helpful customer support agent"
- "You are a professional translator specializing in business content"

User Message (required): This is your actual prompt - what you want the AI to do.

Examples:

- "Write an Instagram caption for this product: {{ $json.productName }}. Key features: {{ $json.features }}. Keep it under 150 characters and include relevant emoji."
- "Summarize this text in 3 bullet points: {{ $json.content }}"
- "Translate this into Spanish: {{ $json.englishText }}"

Notice the `{{ $json.fieldName }}` parts - these pull in data from previous nodes in your workflow. This is how you make your prompts dynamic.

Step 6: Configure options (optional)

You'll see additional settings:

Temperature: Controls creativity vs consistency

- 0 = Very consistent, deterministic, same input always gives same output (good for translations, data extraction)
- 1 = Very creative, varied, same input gives different outputs (good for creative writing, brainstorming)
- 0.7 = Balanced (good default for most tasks)

Max Tokens: Limits how long the response can be

- 1 token ≈ 0.75 words
- If you want a short caption, set to 100
- If you want a full article, set to 2000+
- Default 1024 is good for most tasks

Step 7: Connect and test

Connect the OpenAI node after whatever node provides the data you need. Execute the workflow and check the output. The AI's response will be in `{{ $json.message.content }}` or `{{ $json.choices[0].message.content }}` depending on the setup.

Writing Good Prompts (This Makes or Breaks Your Results)

Most people's prompts are too vague. Here's how to write prompts that get you exactly what you want:

Bad prompt: "Write a caption"

This is too vague. A caption for what? How long? What tone? The AI will guess and you'll get mediocre results.

```
Good prompt:

Write an Instagram caption for {{ $json.productName }}.

Product details:
- Type: {{ $json.productType }}
- Key feature: {{ $json.keyFeature }}
- Target audience: {{ $json.targetAudience }}

Requirements:
- Under 150 characters
- Include 2-3 relevant emoji
- End with a question to boost engagement
- Don't use hashtags
- Casual, friendly tone

Example style: "Loving our new bamboo sunglasses 🕶️✨ Perfect for
sunny days! Who's ready for summer? 🌴"
```

This prompt is specific about:
- What to write (Instagram caption)
- For what (specific product)
- Constraints (under 150 characters, no hashtags)
- Style (casual, friendly)
- Format (include emoji, end with question)
- Example (shows what good looks like)

The AI now knows exactly what you want.

Key principles for good prompts:

1. Be specific about format
Instead of: "List the benefits"
Say: "List 3 benefits, formatted as: Benefit 1: [text], Benefit
2: [text], Benefit 3: [text]"

2. Give examples
Show the AI what good output looks like. If you want a certain
style, give an example of that style.

3. Set constraints
- Length limits (under 200 words, exactly 5 bullet points)
- What to include ("must mention the price")
- What to avoid ("don't use technical jargon", "no hashtags")
```
```

4. Specify tone
- Professional, casual, friendly, formal, humorous, empathetic, urgent, calm
- The AI will match whatever tone you specify

5. Use structured data
If you're giving the AI data to work with, format it clearly:
```
Product Name: {{ $json.name }}
Price: {{ $json.price }}
Features: {{ $json.features }}

Now write a description based on this data.
```

6. Request specific output format
Tell the AI exactly how to structure the response:
- "Respond with only the caption text, nothing else"
- "Format your response as JSON: {caption: "...", hashtags: ["...", "..."]}"
- "Start with 'Subject:' then the subject line, then 'Body:' then the email body"

### Common Mistakes People Make

Mistake 1: Vague prompts

"Write something good about this product." What does "good" mean? How long? What style? The AI will give you mediocre, generic output.

Be specific. Always include: what to write, how long, what tone, any requirements or constraints.

Mistake 2: Not testing with different data

Your prompt works great with one product, but when different data comes through, it breaks or gives weird results. Always test your prompts with various types of input data - short names, long names, missing fields, special characters. Make sure it handles edge cases.

Mistake 3: Forgetting to use dynamic data

Someone writes: "Write a caption for bamboo sunglasses"

```
But their automation processes different products. They forgot to
use `{{ $json.productName }}` to pull in the actual product name.
Now every product gets a caption about bamboo sunglasses.

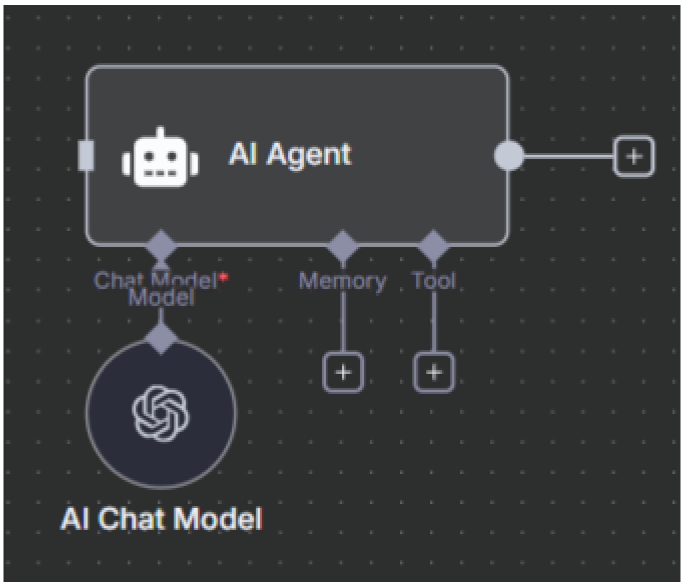
Make your prompts dynamic by pulling in data from previous nodes.

Mistake 4: Not handling AI's extra text

Sometimes the AI adds extra commentary like "Here's your
caption:" before the actual caption, or "I hope this helps!"
after. If you're using that text directly in another node, this
extra stuff breaks things.

Either:
- Tell the AI: "Output ONLY the caption text, no introduction or
commentary"
- Or use a Code node after to strip out extra text
```

AI Agent Node



Docs:
https://docs.n8n.io/integrations/builtin/cluster-nodes/root-nodes/n8n-nodes-langchain.agent/#ai-agent-node

Okay, this is where things get really powerful, but I need to explain it clearly because people get confused about the difference between AI Agent and the OpenAI node we just covered.

Remember how the OpenAI "Message a Model" node is like having a one-way conversation with ChatGPT? You send a prompt, it sends back text, done. The AI can't DO anything except read and write.

AI Agent is completely different. It's like having ChatGPT as an actual assistant who can use tools and take actions. The AI Agent can:

- Have back-and-forth conversations (not just one prompt and done)
- Decide which tools to use based on what's needed
- Access your Google Sheets, Calendar, Gmail, or any other connected tools
- Think through multi-step problems
- Remember context from earlier in the conversation

Think of it like this:

Message a Model = You hand someone a note, they write back a response, end of interaction.

AI Agent = You hire an assistant, they have access to your office tools, they can check your calendar, read your emails, write in your databases, and have ongoing conversations with you or your customers.

The AI Agent has the language model built inside it - you don't need a separate AI model node. You just pick which AI model to use (GPT-4, Claude, etc.) in the Agent's settings.

When You Would Actually Use This

AI Agent is for situations where the AI needs to think, make decisions, and take actions - not just write text.

Conversational customer service bot: A customer messages you on WhatsApp: "Do you have any appointments available tomorrow?"

The AI Agent:

1. Reads the question
2. Thinks: "I need to check the calendar"
3. Uses the Google Calendar tool

4. Sees what's available
5. Responds: "Yes! I have 2 PM and 4 PM available. Which works better for you?"

Customer: "2 PM works"

AI Agent:

1. Reads the response
2. Thinks: "I need to book this"
3. Uses the Google Calendar tool to create the appointment
4. Uses Google Sheets tool to log the booking
5. Responds: "Perfect! You're booked for 2 PM tomorrow. I sent a confirmation to your email."

Notice how the AI decided on its own which tools to use and when. You didn't program "if they say 2 PM, do X". The AI figured it out.

Personal assistant via Telegram: You message your bot: "What's on my calendar today?"

AI Agent:

1. Uses Google Calendar tool
2. Reads your schedule
3. Responds: "You have 3 meetings today: 9 AM with Sarah, 1 PM team standup, and 4 PM client call with Acme Corp."

You: "Move the team standup to 2 PM"

AI Agent:

1. Uses Google Calendar tool to update the meeting
2. Responds: "Done! Standup moved to 2 PM."

You didn't build separate flows for "read calendar" vs "update calendar". The Agent figures out what you want and uses the appropriate tool.

Email management assistant: You message: "Summarize my unread emails from today"

AI Agent:

1. Uses Gmail tool to fetch unread emails
2. Reads through them
3. Responds: "You have 12 unread emails. 3 are sales inquiries, 5 are newsletters (probably can ignore), 2 are

from your team about the project deadline, and 2 are meeting confirmations."

You: "Reply to the sales inquiries saying I'll get back to them tomorrow"

AI Agent:

1. Identifies which emails are sales inquiries
2. Drafts appropriate responses
3. Uses Gmail tool to send the replies
4. Responds: "Sent replies to all 3 sales inquiries letting them know you'll respond tomorrow."

Data research assistant: You message: "How many leads did we get last week and what was their average budget?"

AI Agent:

1. Uses Google Sheets tool
2. Reads your leads database
3. Filters for last week
4. Calculates the average budget
5. Responds: "Last week you got 23 leads with an average budget of $4,200."

You: "How does that compare to the week before?"

AI Agent:

1. Goes back to Google Sheets
2. Gets previous week's data
3. Responds: "The week before you had 19 leads with average budget of $3,800. So you're up 4 leads and $400 in average budget."

The pattern here: the AI needs to access tools, make decisions about which tool to use, and potentially have multi-turn conversations. That's AI Agent territory.

The Key Difference from Message a Model (This is Critical)

Let me make this crystal clear because this is the most common confusion:

Message a Model:

- Input: Your prompt
- Output: Text response
- No tools, no actions, no conversation
- Use for: Writing captions, generating content, translations, summaries

AI Agent:

- Input: A conversation or question
- Output: Text response AND/OR actions taken with tools
- Can use tools (Google Sheets, Calendar, APIs, etc.)
- Can have multi-turn conversations
- Thinks and decides which tool to use
- Use for: Chatbots, assistants, anything requiring actions or decisions

Example to illustrate:

Task: "Create a social media caption for this product"

With Message a Model: You send the product details in a prompt, AI writes the caption, you get the text back. Done. Perfect for this task.

Task: "Check my calendar and reschedule tomorrow's 2 PM meeting to Friday at the same time"

With Message a Model: Can't do it. The AI has no access to your calendar and can't take actions.

With AI Agent: AI reads your request, uses Google Calendar tool to check tomorrow's meetings, finds the 2 PM one, uses the tool again to reschedule it to Friday, confirms it's done. Perfect for this task.

How to Actually Set This Up (Step by Step)

Step 1: Add the AI Agent node

Click the plus (+) button, search for "AI Agent", and add it to your canvas.

Step 2: Choose your AI model

This is different from Message a Model - you don't need a separate AI Chat Model node. The model selection is built right into the AI Agent.

Click on the AI Agent node. You'll see settings for the language model. Under "Model" you'll see options:

- GPT-4o: Most capable, more expensive
- GPT-4o-mini: Best balance (this is what I use)
- Claude models: Good alternative to GPT
- Other options depending on your n8n setup

Pick your model. For most use cases, GPT-4o-mini is perfect.

Step 3: Write the system prompt

This is THE most important part of setting up an AI Agent. The system prompt is like the AI's job description and rulebook. It determines how the AI behaves, what its personality is, and how it makes decisions.

Here's what a good system prompt looks like:

```
You are a booking assistant for a spa business.

Your job:
1. Help customers book appointments
2. Answer questions about services and pricing
3. Check appointment availability

Available services:
- Massage ($80, 60 minutes)
- Facial ($100, 90 minutes)
- Manicure ($40, 45 minutes)

Business hours: 9 AM - 6 PM, Monday-Saturday
Closed Sundays

How to help customers:
- Be friendly and professional
- Always confirm appointment details before booking
- If they ask about availability, check the calendar first
- If no times are available, suggest the next available day

Important rules:
- Never book appointments outside business hours
- Always get customer's name, email, and phone before booking
```

~~~~
- Double-check which service they want before confirming
```

Notice this prompt tells the AI:
- Who it is (booking assistant for a spa)
- What it's supposed to do (book appointments, answer questions)
- What information it needs to know (services, prices, hours)
- How to behave (friendly, professional)
- Specific rules to follow (don't book outside hours, collect
contact info)

The better your system prompt, the better your AI Agent performs.

**Step 4: Connect tools**

This is what makes AI Agent powerful - you give it tools it can
use. Scroll down in the AI Agent settings and you'll see "Tools".
This is where you connect whatever the AI needs access to.

Let me show you how to connect a tool using Google Calendar as an
example:

Click "Add Tool" and you'll see a list of available tools:
- Google Calendar
- Google Sheets
- Gmail
- HTTP Request (for custom APIs)
- And many more

Let's say you want the AI to check and book appointments. Click
"Google Calendar".

You'll need to set up credentials (connect your Google account).
Then you choose what actions the AI can take:
- Read events? (check availability)
- Create events? (book appointments)
- Update events? (reschedule)
- Delete events? (cancel)

Enable what you want the AI to be able to do. Be careful here -
only enable what the AI actually needs. If you don't want the AI
to be able to delete events, don't enable that permission.

You can connect multiple tools. For a complete booking assistant,
you might connect:
- Google Calendar (for appointments)
~~~~

- Google Sheets (for logging bookings)
- Gmail (for sending confirmations)

The AI will automatically decide which tool to use based on what the person asks for.

Step 5: Set up the chat interface (if needed)

If you're building a chatbot, you need to enable the chat interface. In the AI Agent settings, look for "Chat" options.

You'll see "Has Memory" - toggle this ON if you want the AI to remember the conversation. This is important for multi-turn conversations. If someone says "Book that appointment" referring to a time they mentioned 3 messages ago, the AI needs memory to know what "that appointment" means.

Step 6: Connect to a trigger

AI Agents are typically triggered by:
- Telegram Trigger (for Telegram bots)
- Webhook (for chatbots on websites or other apps)
- Form Trigger (for form-based interactions)

Connect your AI Agent after whichever trigger makes sense for your use case.

Step 7: Test with real conversations

This is critical - don't just test one message. Have full conversations with your AI Agent:
- Ask it to do things
- Give it incomplete information and see if it asks for clarification
- Try to confuse it or ask for things outside its scope
- See if it uses the right tools at the right times

Testing AI Agents requires more thorough testing than other nodes because they make decisions on their own.

Writing Good System Prompts for AI Agents

Your system prompt makes or breaks your AI Agent. Here's how to write one that works:

1. Start with identity and purpose

```
You are [who/what]
Your job is [primary purpose]
```

Example:
```
You are a customer support agent for an e-commerce store.
Your job is to help customers with orders, returns, and product
questions.
```

2. List available information the AI needs
```
Product catalog:
- Widget A: $29, ships in 2 days
- Widget B: $49, ships in 3 days

Return policy: 30 days, full refund

Shipping: Free over $50, otherwise $5.99
```

Give the AI all the information it needs to answer questions. If it doesn't know your prices, it might make them up.

3. Explain how to use tools
```
You have access to these tools:
- Order tracking: Use this when customers ask where their order
is
- Cancel order: Use this when customers want to cancel
- Issue refund: Use this when approving returns

Always confirm with the customer before canceling orders or
issuing refunds.
```

Tell the AI when to use each tool and any precautions.

4. Set behavioral guidelines
```
How to interact:
- Be friendly but professional
- Keep responses under 3 sentences when possible
- If you don't know something, say so - don't make up answers
```

- Always ask for order number before looking up orders
```

5. Define boundaries
```
What you CAN do:
- Answer product questions
- Track orders
- Process returns

What you CANNOT do:
- Give discounts (tell them to email support@company.com)
- Change shipping addresses after order is placed
- Provide medical advice about products
```

Clear boundaries prevent the AI from promising things you can't deliver.

6. Include examples of good behavior (optional but helpful)
```
Example conversation:
Customer: "Where's my order?"
You: "I can help you track that! What's your order number?"
Customer: "12345"
You: "Thanks! Order 12345 shipped yesterday and should arrive in 2 days. Here's your tracking link: [link]"
```

**Common Mistakes People Make**

****Mistake 1: Vague system prompt****

System prompt: "You are a helpful assistant."

That's it. The AI has no idea what it's supposed to help with, what tools it has, what information it can provide, or what boundaries it should respect. Every response will be generic and unreliable.

Always be specific about role, purpose, available information, and rules.

****Mistake 2: Not testing multi-turn conversations****
```

~~~
Someone tests their booking agent with: "Book an appointment for
tomorrow at 2 PM"

It works! They think they're done.

But what if someone says:
- "Do you have anything available tomorrow?" (needs to check
first)
- "What times do you have?" (vague)
- "Actually cancel that, I need next week instead" (needs to
remember what "that" refers to)

AI Agents need thorough testing because they handle dynamic
conversations.

**Mistake 3: Giving too many tools**

Someone connects 15 different tools "just in case." Now the AI is
overwhelmed with choices and sometimes picks the wrong tool or
takes forever to decide.

Only connect tools the AI will actually need for its specific
job. If it's a booking assistant, it needs Calendar. It probably
doesn't need Gmail AND Slack AND Notion AND...

**Mistake 4: Not setting max iterations**

AI Agents can get stuck in loops where they keep calling tools
over and over. In the settings, there's usually a "Max
Iterations" option. Set this to something reasonable like 10.
This means "after 10 tool uses, stop and give me what you have."
Prevents infinite loops.

**Mistake 5: Forgetting to handle errors**

The AI tries to use Google Calendar but the credentials expired.
Or tries to book an appointment but the time is already taken.
What should it do?

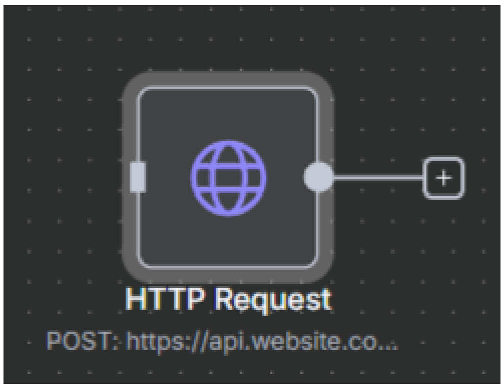
In your system prompt, tell the AI how to handle errors:
```
If a tool fails:
- Tell the customer politely what went wrong
- Suggest alternatives
- Don't keep retrying the same action
```
~~~

```
Mistake 6: Not using memory when needed

You're building a chatbot but forget to enable "Has Memory". Now
every message the AI treats like a brand new conversation.
Customer says "Book appointment for 2 PM" and AI responds.
Customer says "Actually make it 3 PM" and AI has no idea what
"it" refers to because it doesn't remember the previous message.

For any multi-turn conversation, enable memory.
```

HTTP Request Node



Doc:
https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.httprequest/#http-request-node

Okay, let's break this down in the simplest way possible because HTTP Request sounds technical but it's actually a straightforward concept.

Every time you use an app or website, that app is talking to a server somewhere on the internet. When you post to Instagram, Instagram's app sends a message to Instagram's server saying "Hey, post this photo with this caption." When you search Google, your browser sends a message to Google's server saying "Hey, show me results for this search term."

These messages back and forth are called HTTP requests. It's just the language that computers use to talk to each other over the internet.

The HTTP Request node lets YOUR automation send these same kinds of messages to ANY service on the internet that has an API (which is basically a way for other programs to interact with it).

Think of it like this: n8n has built-in nodes for popular services like Google Sheets, Slack, Instagram. But there are millions of other services out there that n8n doesn't have specific nodes for. The HTTP Request node is your universal connector - it lets you talk to ANY service, even ones that don't have a dedicated n8n node.

When You Would Actually Use This

Connecting to services without n8n nodes: You want to use an AI image generator called Midjourney, or a video generator called Runway, or a voice AI service called ElevenLabs. n8n doesn't have built-in nodes for these. But they all have APIs. The HTTP Request node lets you connect to them.

Posting to social media that doesn't have nodes: You want to automatically post to TikTok, but n8n doesn't have a TikTok node. TikTok has an API. You use HTTP Request to post.

Calling your own custom services: You built your own web app or service and want n8n to interact with it. HTTP Request lets you send data to your app.

Accessing specialized AI services: You want to use Claude AI instead of OpenAI, or you want to use Anthropic's vision API, or Replicate's AI models. HTTP Request connects to any of these.

Fetching data from websites: You want to get current weather data, stock prices, cryptocurrency values, sports scores. These services have APIs you can query with HTTP Request.

Triggering other automations: You have multiple automation tools and want them to talk to each other. HTTP Request can send data from n8n to Make.com, to Zapier, to anywhere.

The common thread: any time you need to connect to a service or API that doesn't have a dedicated n8n node, HTTP Request is your answer.

Understanding the Basics: GET vs POST (You Need to Know This)

When you send an HTTP request, you need to specify what TYPE of request it is. There are several types, but the two you'll use 99% of the time are GET and POST.

GET: "I want to receive information"

Think of GET like asking a question. You're requesting information from the server, but you're not sending any data or changing anything.

Examples of GET requests:

- "What's the current weather in London?" (getting weather data)

- "Show me the list of products in my store" (getting product list)
- "What's the current price of Bitcoin?" (getting price data)
- "Give me information about user #12345" (getting user info)

When you use GET, you're basically saying "Hey server, send me this information please."

POST: "I'm sending you information to process"

Think of POST like handing someone a form you filled out. You're sending data TO the server and asking it to do something with that data.

Examples of POST requests:

- "Here's an image, please generate a video from it" (sending image data)
- "Here's text, please translate it to Spanish" (sending text data)
- "Create a new customer with this name and email" (sending customer data)
- "Post this content to Instagram" (sending post content)

When you use POST, you're saying "Hey server, I'm giving you some data, please process it and do something."

How do you know which to use?

Simple rule:

- If you're ASKING FOR information = GET
- If you're SENDING information or asking the server to DO something = POST

When you read API documentation (instructions for how to use an API), it will always tell you which method to use for each action. You don't have to guess.

What is an API and API Documentation?

I keep saying "API" - let me explain this clearly because it's important.

API stands for Application Programming Interface. That's a mouthful. Here's what it really means in simple terms:

An API is like a restaurant menu. The menu tells you:

- What dishes are available (what actions you can request)
- What ingredients you need to provide (what data to send)
- What you'll get back (what response to expect)

Every service that has an API publishes API documentation - basically the menu that tells you how to interact with their service.

Example: Let's say you want to use an AI image generator's API.

Their API documentation might say:

Endpoint: https://api.imageai.com/generate

Method: POST

Headers needed: Authorization: Bearer YOUR_API_KEY

Body:

The prompt text for the image

The width you want (like 1024)

The height you want (like 1024)

Response: You'll get back a URL to the generated image

This is telling you:

- **Where to send the request:** https://api.imageai.com/generate
- **What method to use:** POST (because you're sending data)
- **What authentication you need:** Your API key in the Authorization header
- **What data to send:** The prompt and image dimensions in the body
- **What you'll get back:** A URL to the generated image

When you set up an HTTP Request node, you're basically translating these API instructions into n8n's settings.

How to Actually Set This Up (Step by Step)

Let me walk you through setting up an HTTP Request using a real example: calling an AI image generator.

Step 1: Find the API documentation

Let's say you want to use Replicate (an AI service) to generate images. Go to Replicate's website and find their API documentation. It will show you the details you need.

The documentation says:

- URL: https://api.replicate.com/v1/predictions
- Method: POST
- You need an API token for authentication
- Send your prompt in the request body

Step 2: Add the HTTP Request node

Click plus (+), search for "HTTP Request", add it to your canvas.

Step 3: Set the Method

In the HTTP Request node settings, the first option is "Method". Click the dropdown and select POST (because we're sending data - the image prompt).

Step 4: Enter the URL

In the "URL" field, paste:
https://api.replicate.com/v1/predictions

This is where the request will be sent.

Step 5: Set up Authentication

Most APIs require authentication - a way to prove you're allowed to use their service. The most common method is an API key.

The API documentation will tell you where to get your API key. For Replicate, you'd go to their website, create an account, go to settings, and copy your API key.

Now in n8n, look for "Authentication" settings in the HTTP Request node. Select the authentication type. Common options:

Header Auth: You put your API key in a header (most common)

Bearer Token: A specific type of header auth where you use "Bearer YOUR_TOKEN"

For this example, Replicate uses Bearer Token. Select that, then paste your API key in the Token field.

Step 6: Set Headers (if needed)

Headers are like labels on an envelope - they give extra information about your request.

The most common header you'll need is "Content-Type" which tells the server what format your data is in.

In the HTTP Request node, enable "Send Headers" and add:

- Name: Content-Type
- Value: application/json

This tells the server "Hey, I'm sending you data in JSON format."

Step 7: Set the Body (for POST requests)

This is where you put the actual data you're sending. For POST requests, you usually send data in the body.

Enable "Send Body" and choose the body type. Most modern APIs use JSON, so select "JSON".

Now you write the data you're sending. For example:

You specify which AI model version to use

You include the input data like the prompt text

You might include other settings like image size

Notice you can use things like this format to pull in data: two curly braces, dollar sign, json, dot, and then the field name. This pulls in data from a previous node, making your request dynamic.

Step 8: Execute and check the response

Run your workflow. The HTTP Request will execute and return data. Look at the response in n8n.

The response usually contains:

- Status code (200 = success, 400 = error, etc.)
- Body (the actual data the server sent back)

If you got a 200 status code, it worked! The response body will have the data you need (like the generated image URL).

If you got an error code, check:

- Is your API key correct?
- Is the URL correct?
- Is your body data formatted correctly?
- Did you include all required headers?

Common Mistakes People Make

Mistake 1: Wrong HTTP method

The API documentation says to use POST, but someone used GET. Result: the request fails because GET requests don't send body data.

Always use the exact method the API documentation specifies.

Mistake 2: Missing or wrong headers

Many APIs require specific headers, especially:

- Authorization (your API key)
- Content-Type (usually application/json)

If you forget these, the API rejects your request. Always check the API docs for required headers.

Mistake 3: Malformed JSON in the body

Someone writes JSON with a syntax error like having an extra comma at the end or using single quotes instead of double quotes.

This causes the request to fail. JSON is very strict about format:

- Use double quotes, not single quotes
- No trailing commas
- Proper brackets and braces

If you're unsure, use a JSON validator online to check your JSON before putting it in the node.

Mistake 4: Not checking the API documentation carefully

Someone assumes how the API works instead of reading the docs. They send data in the wrong format, to the wrong endpoint, or with the wrong parameters.

Always read the API documentation carefully. Every API is different and has its own requirements.

Mistake 5: Hardcoding data instead of making it dynamic

Someone writes the prompt directly as text like "a beautiful sunset"

But they want to process different prompts. They forgot to use dynamic data by referencing fields from previous nodes.

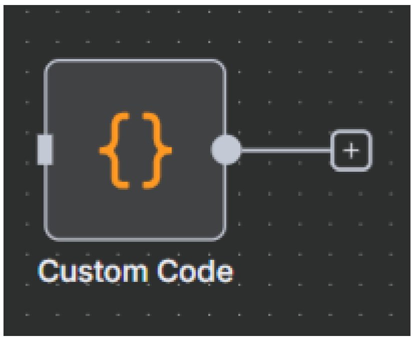
Make your HTTP Requests dynamic by pulling in data from previous nodes.

Mistake 6: Not handling errors

The API request fails (wrong URL, expired API key, service is down), but there's no error handling. The automation just breaks silently.

After an HTTP Request, use an IF node to check if the status code is 200. If not, handle the error (log it, send an alert, retry, etc.).

Custom Code Node



Doc:
https://docs.n8n.io/integrations/builtin/cluster-nodes/sub-nodes/n8n-nodes-langchain.toolcode/#custom-code-tool-node

The Code node lets you write JavaScript code to process your data in ways that other nodes can't easily do. It's like having a programmable Swiss Army knife - you can make it do almost anything you need.

But here's the important part: **you don't need to know how to code to use this node.** You can use ChatGPT or Claude to write the code for you. I do this all the time. I tell the AI what I want to do, it writes the JavaScript, I paste it into the Code node, and it works.

Think of the Code node as a way to handle special cases or complex data transformations that would be really annoying or impossible to do with standard nodes.

When You Would Actually Use This

Splitting text in custom ways: Someone sends you a message in Telegram with multiple pieces of information separated by semicolons: "sunset scene; inspiring text; upbeat music; happy vibe"

You need to split this into separate variables. Sure, you could try to use multiple Edit nodes and text operations, but it would take 5-6 nodes and be messy. Or you use code - one node, done.

Formatting data for specific APIs: An API requires data in a very specific format that's hard to create with standard nodes. Code can format it exactly right in seconds.

Doing math calculations: You need to calculate discounts, taxes, percentages, or complex formulas. Code handles math easily. You could use a calculator node or multiple nodes to do math step by step, but code does it all at once.

Cleaning messy data: Data comes in with extra spaces, weird characters, inconsistent formatting, mixed upper and lowercase. Code can clean it up in one go.

Combining data from multiple fields: You have firstName and lastName in separate fields, need them combined as fullName. You could use a Set node and concatenate, but code is simpler.

Parsing complex data structures: You receive data in a nested, complicated format that n8n's standard nodes struggle with. Code can parse and restructure it easily.

Working with dates and times: You need to calculate "7 days from now" or "convert this timestamp to a readable date" or "check if this date is a weekend." Date math is annoying with regular nodes but simple with code.

Filtering or transforming arrays: You have a list of 50 items and need to filter them based on multiple conditions, or transform each item in a specific way. Code can process all of them efficiently.

The pattern here: any time you think "this would be so simple if I could just write 5 lines of code," that's when you use the Code node. Or when you find yourself chaining together 8 nodes to do something that feels like it should be simpler.

How to Actually Set This Up (Step by Step)

Step 1: Add the Code node

Click plus (+), search for "Code", add it to your canvas.

Step 2: Decide which mode to use

When you click on the Code node, you'll see "Mode" with two options:

Run Once for All Items: The code runs one time with access to ALL items that came from the previous node. Use this when you want to process all data together or when you're transforming the entire dataset.

Run Once for Each Item: The code runs separately for EACH item. If 5 items came from the previous node, the code runs 5 times (once per item). Use this when each item needs independent processing.

For most use cases, "Run Once for All Items" is what you want. I'd say I use this mode 90% of the time.

Step 3: Get the code (from ChatGPT or Claude)

Here's where I'm going to save you so much time. You don't need to learn JavaScript to use the Code node. Just tell an AI what you want to do.

Example prompt to ChatGPT:

"Write n8n code node JavaScript that splits a string by semicolons and returns an object with fields: imagePrompt, textOverlay, musicStyle, and vibe. The input string is in dollar sign input dot first parentheses dot json dot message dot text"

ChatGPT will give you the code. Then you just copy it and paste it into the Code node.

Step 4: Understanding how to access your data in code

This is important. When writing code for n8n (or asking AI to write it), you need to know how to access the data that came from previous nodes.

If you're using "Run Once for All Items" mode:

To get the first item's data: `$input.first().json`

To get all items: `$input.all()`

To get data from a specific previous node by name: `$('Node Name').first().json`

If you're using "Run Once for Each Item" mode:

To get the current item: `$input.item.json`

Examples:

If the previous node gave you data with a field called "message", you access it like:

```
const message = $input.first().json.message;
```

If you want to loop through all items:

```
const items = $input.all();
```

When you ask ChatGPT to write code for you, tell it where the data is. For example: "The text is in dollar sign input dot first parentheses dot json dot message dot caption"

Step 5: Understanding how to return data from code

This is critical - your code MUST return data. If it doesn't return anything, the next node won't receive any data and your automation breaks.

To return a single object:

At the end of your code, write:

```
return { field1: value1, field2: value2 };
```

To return multiple items:

```
return items.map(item => ({ ... }));
```

To return what you received (no changes):

```
return $input.all();
```

Always make sure your code has a return statement at the end.

Step 6: Paste the code and test

Copy the code ChatGPT gave you. Paste it into the Code node's editor. Execute the node and check the output.

If you see data in the output, great! If you see an error, copy the error message, go back to ChatGPT and say "I got this error: [paste error]. Please fix the code." ChatGPT will fix it.

Real Examples with Actual Code

Let me show you real code examples that I actually use in my templates, so you can see what this looks like:

Example 1: Splitting text by delimiter

Scenario: Someone sends a Telegram message like "beach sunset; Make it inspiring; upbeat electronic; summer vibes"

I need to split this into 4 separate fields.

Code:

```
javascript
const text = $input.first().json.message.caption;
const parts = text.split(';').map(part => part.trim());

return {
  imagePrompt: parts[0],
  textOverlay: parts[1],
  musicPrompt: parts[2],
  vibe: parts[3]
};
```

What this does:

- Gets the text from the Telegram message
- Splits it by semicolons into an array
- `.trim()` removes extra spaces from each part
- Returns an object with named fields

Example 2: Cleaning email addresses

Scenario: People fill out a form but their emails have spaces, mixed case, etc. I need clean emails.

Code:

```
javascript
const items = $input.all();

return items.map(item => {
  const cleanEmail = item.json.email
    .toLowerCase()
    .trim()
    .replace(/\s+/g, '');

  return {
    ...item.json,
    email: cleanEmail
  };
});
```

What this does:

- Gets all items from previous node
- For each item, takes the email
- Converts to lowercase
- Removes spaces from beginning/end
- Removes all spaces inside the email
- Returns all the original data plus the cleaned email

Example 3: Calculating totals

Scenario: I have a list of purchases and need to calculate subtotal, tax, and total.

Code:

javascript
```javascript
const items = $input.first().json.items;
const taxRate = 0.08; // 8% tax

const subtotal = items.reduce((sum, item) => sum + item.price,
0);
const tax = subtotal * taxRate;
const total = subtotal + tax;

return {
  subtotal: subtotal.toFixed(2),
  tax: tax.toFixed(2),
  total: total.toFixed(2),
  itemCount: items.length
};
```

What this does:

- Gets array of items
- Adds up all prices to get subtotal
- Calculates tax
- Calculates total
- `.toFixed(2)` makes sure we have exactly 2 decimal places
- Returns formatted numbers

Example 4: Formatting dates

Scenario: I have a timestamp and need it in readable format like "Monday, Jan 15, 2024 at 3:30 PM"

Code:

javascript
```
const timestamp = $input.first().json.appointmentTime;
const date = new Date(timestamp);

const options = {
  weekday: 'long',
  year: 'numeric',
  month: 'short',
  day: 'numeric',
  hour: 'numeric',
  minute: '2-digit',
  hour12: true
};

const formatted = date.toLocaleString('en-US', options);

return {
  originalTimestamp: timestamp,
  formattedDate: formatted,
  dayOfWeek: date.toLocaleDateString('en-US', { weekday: 'long'
}),
  isWeekend: date.getDay() === 0 || date.getDay() === 6
};
```

What this does:

- Converts timestamp to Date object
- Formats it nicely
- Also tells you what day of week
- Checks if it's a weekend
- Returns multiple useful formats

Example 5: Combining first and last name

Scenario: Data has firstName and lastName separate, need fullName.

Code:

javascript
```
const items = $input.all();
```

```
return items.map(item => ({
  fullName: `${item.json.firstName} ${item.json.lastName}`,
  email: item.json.email,
  phone: item.json.phone
}));
```

What this does:

- Gets all items
- For each one, combines firstName and lastName with a space
- Returns new structure with fullName plus original fields

How to Get ChatGPT to Write Code For You

This is the key skill. Here's exactly how to prompt ChatGPT:

Good prompt structure:

```
Write n8n code node JavaScript that [what you want to do].

The input data is in [where the data is].

Return [what format you want back].
```

Real examples of good prompts:

Example 1: "Write n8n code node JavaScript that extracts the email domain from an email address. The email is in dollar sign input dot first parentheses dot json dot email. Return an object with fields: email and domain."

Example 2: "Write n8n code node JavaScript that filters an array to only include items where the price is greater than 100. The array is in dollar sign input dot first parentheses dot json dot products. Return the filtered array."

Example 3: "Write n8n code node JavaScript that takes a phone number and formats it as parenthesis area code close parenthesis space three digits dash four digits. The phone number is in dollar sign input dot first parentheses dot json dot phone. Return an object with originalPhone and formattedPhone."

Example 4: "Write n8n code node JavaScript that counts how many times each word appears in a text string. The text is in dollar sign input dot first parentheses dot json dot content. Return an object where keys are words and values are counts."

What makes these prompts good:

- Specific about what transformation you want
- Tells ChatGPT exactly where the input data is
- Specifies what format to return
- Uses clear, simple language

If the code doesn't work:

Copy the error message from n8n, go back to ChatGPT and say: "I got this error: [paste error]. Please fix the code."

ChatGPT will fix it. Sometimes you need to go back and forth 2-3 times, but it always works eventually.

Common Mistakes People Make

Mistake 1: Not returning data

Someone writes code that does a bunch of processing but forgets to return anything at the end. The next node receives nothing and breaks.

Always end your code with `return { ... };` or `return items;` or whatever makes sense. The code MUST return something.

Mistake 2: Wrong data path

Someone asks ChatGPT to write code but doesn't tell it where the data is correctly. ChatGPT guesses wrong. The code tries to access `$input.first().json.text` but the actual data is at `$input.first().json.message.text`.

Always check your data structure first. Look at the output from the previous node, see exactly where your data is, and tell ChatGPT the correct path.

Mistake 3: Using the wrong mode

Someone uses "Run Once for Each Item" when they should use "Run Once for All Items" or vice versa.

If you need to look at all items together (like counting them, filtering them, sorting them), use "Run Once for All Items".

If each item is completely independent (like cleaning one email at a time), either mode works, but "Run Once for All Items" is usually simpler.

Mistake 4: Not handling missing data

The code assumes a field always exists. But sometimes it doesn't. The code tries to access `item.json.name` but some items don't have a name field. Code crashes.

Always handle missing data. Tell ChatGPT: "Make sure the code doesn't crash if the field is missing or empty."

Mistake 5: Modifying data incorrectly

Someone tries to modify the original data directly instead of creating a new object. This can cause weird bugs.

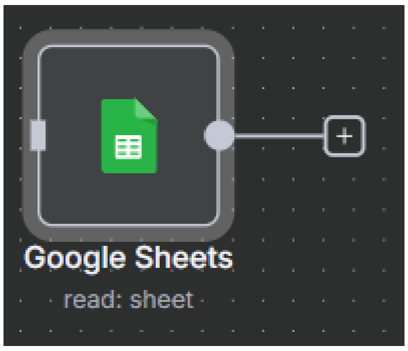
Best practice: always create new objects when transforming data, don't try to modify the originals.

Mistake 6: Making it too complex

Someone writes 50 lines of code to do something that a standard node could do in 5 clicks. Code should be used when it makes things SIMPLER, not more complex.

Before using Code, ask yourself: "Could I do this easily with standard nodes?" If yes, use standard nodes. If it would take 8+ nodes or be super messy, then use Code.

Google Sheets Node



Doc:
https://docs.n8n.io/integrations/builtin/app-nodes/n8n-nodes-base.googlesheets/#google-sheets

The Google Sheets node connects to Google Sheets (basically Excel but online from Google) and lets you read data from sheets or write data to sheets automatically.

Think of Google Sheets as your database that you can actually see and edit. Instead of your data being locked away in some complicated database system, it's in a spreadsheet that you can open in your browser, look at, edit, share with your team, export to Excel, whatever you need.

For most people starting with automation, Google Sheets is the perfect database because:

- It's free
- Everyone knows how to use it
- You can see your data visually
- You can manually edit it if needed
- You can share it with others easily
- You can export it to CSV or Excel

When You Would Actually Use This

Storing leads from forms: Someone fills out your contact form. The Google Sheets node automatically adds their name, email, phone, and message to a spreadsheet. Now you have a growing list of all leads in one place that you can reference anytime.

Logging automation runs: Every time your automation runs, you log the results to a sheet. You can see: when it ran, what it processed, if it succeeded or failed. This gives you a history of everything your automation did.

Tracking social media posts: Your automation generates and posts content daily. The Google Sheets node logs each post: what was posted, when, to which platform, what the engagement was. Now you have a record of all your content in one place.

Managing bookings or appointments: Someone books an appointment through your automation. It gets added to a Google Sheet with: customer name, date, time, service, status. Your team can see all bookings at a glance.

Building simple CRM systems: Store customer information, purchase history, last contact date, notes. You have a simple customer database that your whole team can access.

Reading product lists or content: You maintain a Google Sheet with your products, prices, or content ideas. Your automation reads from this sheet to know what to post, what prices to charge, what content to create.

Creating reports: Your automation collects data from various sources and compiles it into a Google Sheet as a report. Every Monday, you have a fresh report waiting for you.

The pattern: Google Sheets is for any data you want to store, track, or reference. It's your automation's memory and filing system.

How to Actually Set This Up (Step by Step)

Step 1: Create a Google Sheet

Go to sheets.google.com and create a new blank spreadsheet. Give it a name like "Automation Data" or "Lead Database" or whatever makes sense.

Important setup: The first row should be your headers - the names of your columns. For example:

Row 1: Full Name | Email | Phone | Date | Status

This tells the Google Sheets node what each column represents.

Step 2: Share the sheet with n8n

This is important. n8n needs permission to access your sheet.

Click "Share" button in Google Sheets. You need to share it with the Google account that n8n uses. If you're using cloud n8n, you'll connect your Google account in the next step. If self-hosting, you need to set up Google credentials.

Step 3: Add the Google Sheets node

In n8n, click plus (+), search for "Google Sheets", add it to your canvas.

Step 4: Set up credentials

Click on the Google Sheets node. You'll see it needs "Credentials". Click "Select Credentials" then "Create New Credentials."

You'll be taken through Google's authentication flow:

- Sign in to your Google account
- Grant n8n permission to access your sheets
- You'll be redirected back to n8n

Now n8n can access your Google Sheets.

Step 5: Choose your operation

In the Google Sheets node, you'll see "Operation" with several options. The main ones you'll use are:

Append Row: Add a new row to the bottom of your sheet. This is the most common - use it when you want to add new data (new lead, new log entry, new booking, etc.)

Read Rows: Read data from your sheet. Use this when you want your automation to look up information stored in the sheet.

Update Row: Modify an existing row. Use this when you want to change data that's already there (like updating a status from "pending" to "completed").

Delete Row: Remove a row. Use this rarely, usually only when cleaning up old data.

Lookup: Find a specific row based on a value. Use this when you need to find one specific entry (like finding a customer by email).

For this example, let's use "Append Row" since that's the most common.

Step 6: Select your document and sheet

After choosing "Append Row", you'll see "Document" dropdown. Click it and you'll see a list of all your Google Sheets. Select the one you created.

Then "Sheet" dropdown appears. If your spreadsheet has multiple sheets (tabs at the bottom), select which one. Usually it's "Sheet1".

Step 7: Map your data to columns

This is where you tell n8n what data to put in each column.

You'll see "Columns" section. This shows you all the headers from your first row.

For each column, you specify what data to put there. You can:

- Type static text
- Use data from previous nodes by typing two open curly braces and selecting fields

Example:

If your sheet has columns: Full Name | Email | Phone | Date

You'd set:

- Full Name: (two open curly braces) dollar sign json dot name
- Email: (two open curly braces) dollar sign json dot email
- Phone: (two open curly braces) dollar sign json dot phone
- Date: (two open curly braces) dollar sign now

This pulls data from the previous node and adds it to your sheet.

Step 8: Execute and verify

Run your workflow. The Google Sheets node should execute. Then go to your actual Google Sheet in your browser and refresh it. You should see a new row with your data!

If you see it, perfect! If not, check:

- Are your credentials working?
- Is the sheet name correct?
- Are your column mappings correct?
- Did you select the right document?

Different Operations Explained

Let me explain each operation in detail so you know when to use which:

Append Row (Add new data)

When to use: Any time you want to add new information to your sheet. New lead, new log entry, new customer, new booking.

How it works: Takes the data you provide and adds it as a new row at the bottom of your sheet.

Example use: Someone fills out your form. Append Row adds their info as a new row in your leads sheet.

Settings:

- Document: which Google Sheet
- Sheet: which tab in that sheet
- Columns: what data goes in each column

Read Rows (Get data from sheet)

When to use: Your automation needs to look at data in your sheet to make decisions or use that information.

How it works: Reads rows from your sheet and gives you that data to use in later nodes.

Example use: You have a product list in a sheet. Your automation reads it to know what products to post about today.

Settings:

- Document & Sheet: which sheet to read from
- Range: which rows to read (like A1:D100 means columns A through D, rows 1 through 100)
- Options: return all rows, or only rows with data

Update Row (Change existing data)

When to use: You need to modify data that's already in your sheet. Change a status, update a count, mark something complete.

How it works: Finds a specific row and updates the values you specify.

Example use: Customer places an order. You find their row in the sheet and update their "Last Purchase Date" and "Total Spent".

Settings:

- Document & Sheet
- How to find the row (by row number or by looking up a value)
- Which columns to update
- What new values to put

Lookup (Find specific row)

When to use: You need to find one specific row based on some criteria.

How it works: Searches your sheet for a row where a column matches your search value.

Example use: Customer emails you. You lookup their row by email address to see their account info.

Settings:

- Document & Sheet
- Which column to search in
- What value to search for

Delete Row (Remove data)

When to use: Rarely. Usually only for cleanup or when someone explicitly requests deletion.

How it works: Removes a row from your sheet.

Example use: Someone unsubscribes. You find and delete their row from your subscribers sheet.

Best Practices for Using Google Sheets as a Database

1. Always use headers in row 1

Your first row should have clear column names like "Full Name", "Email", "Phone", "Date Added", "Status". This makes it easy to map data and easy for humans to read.

2. Keep column names consistent

Don't change column names after you start using the sheet. If your automation expects "Email" but you rename it to "Email Address", the automation breaks.

3. Don't leave empty rows in the middle

If you manually delete a row, the Google Sheets node can get confused. It's better to mark rows as "deleted" or "inactive" rather than physically deleting them.

4. Use data validation where appropriate

In Google Sheets itself, you can set column types (like making sure Email column actually contains emails). This prevents bad data from getting in.

5. Back up your sheets

If your sheet contains critical business data, make backups. You can set up automatic backups using Google Sheets' version history or by periodically exporting it.

6. Set permissions carefully

Only share your sheet with people who need access. If it contains customer data, protect it appropriately.

7. Consider splitting into multiple sheets

If your sheet gets huge (thousands of rows), Google Sheets can slow down. Consider splitting data by month or year into different sheets.

Common Mistakes People Make

Mistake 1: Column names don't match

Your automation maps to a column called "Email" but in your sheet, the header is "Email Address" (with a space). The node can't find the column and fails.

Solution: Make sure column names in your sheet exactly match what your automation expects. No extra spaces, same capitalization.

Mistake 2: Wrong sheet selected

Your spreadsheet has multiple tabs (sheets). You're adding data to "Sheet1" but looking at "Sheet2" wondering why nothing appears.

Solution: Make sure you're looking at the same sheet tab that your automation is writing to.