Email Scrapper workflow

Hello! Let's dive into **no-code automation**—yes, you read that correctly—**"NO CODE" for building an email scraper**. I'll guide you step by step, so just follow along, and by the end, you'll have a fully working solution!

We can set this up either on our **local desktop using Docker** or on a **cloud server with an n8n instance**. n8n is a powerful **automation workflow engine** that allows you to connect apps, scrape data, and automate tasks without writing complex code.

By the end of this guide, you'll have a fully functional, no-code email scraper workflow that you can run anytime so why so much of wait, Let's dive!!!

Using Docker

1. Pull the latest n8n Docker image

```
docker pull n8nio/n8n
```

2. Run n8n container

```
docker run -it --rm \
--name n8n \
-p 5678:5678 \
-v ~/.n8n:/home/node/.n8n \
n8nio/n8n
```

Server Deployment

Step 1: Connect to your server

```
ssh username@your_server_ip
```

Step 2: Install Docker & Docker Compose

```
sudo apt update
sudo apt install -y docker.io docker-compose
sudo systemctl enable docker
sudo systemctl start docker
```

Step 3: Create n8n directory

```
mkdir n8n && cd n8n
```

Step 4: Create `docker-compose.yml`

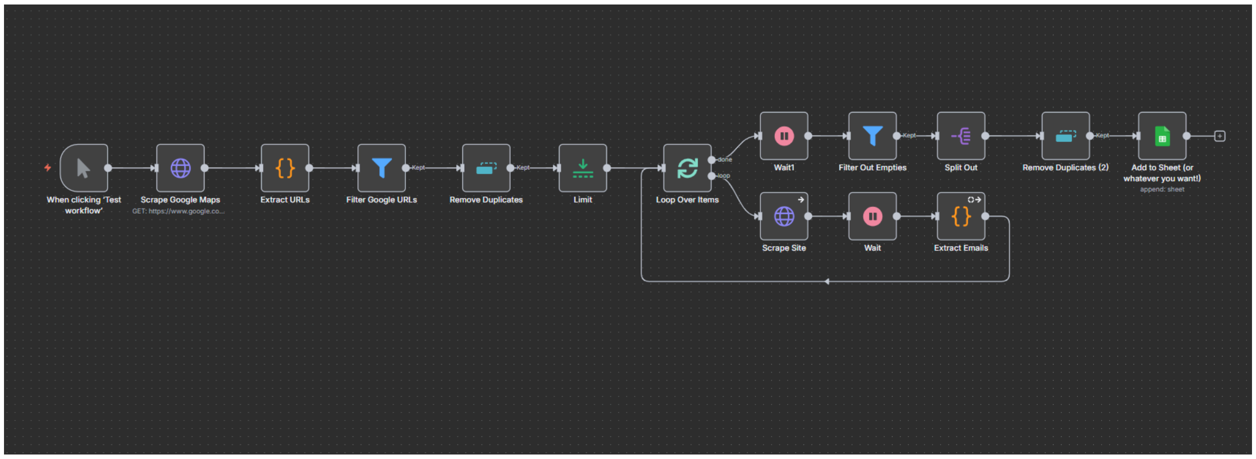
```
version: "3.8"
services:
n8n:
image: n8nio/n8n
restart: always
ports:
- "5678:5678"
environment:
- N8N_BASIC_AUTH_ACTIVE=true
- N8N_BASIC_AUTH_USER=admin
- N8N_BASIC_AUTH_PASSWORD=yourpassword
- N8N_HOST=yourdomain.com
- N8N_PORT=5678
- WEBHOOK_URL=https://yourdomain.com/
volumes:
- ./n8n_data:/home/node/.n8n
```

Step 5: Start n8n

```
docker-compose up -d
```

This automation   scrapes business listings from Google Maps, grabs their **website URLs**, visits those sites to **collect email addresses**, cleans up and **removes duplicates**, and finally puts all the data into a **Google Sheet** . The best part? It’s all done using a **no-code/low-code tool like n8n**, so you don’t have to write a single line of complicated code.

WorkFlow Diagram



Step-by-Step Workflow Description

1. Trigger — “When clicking ‘Test Workflow”

To start the workflow, we’ll use the **trigger button** called **“Trigger”**. This basically tells the workflow, “Hey, start running now!”

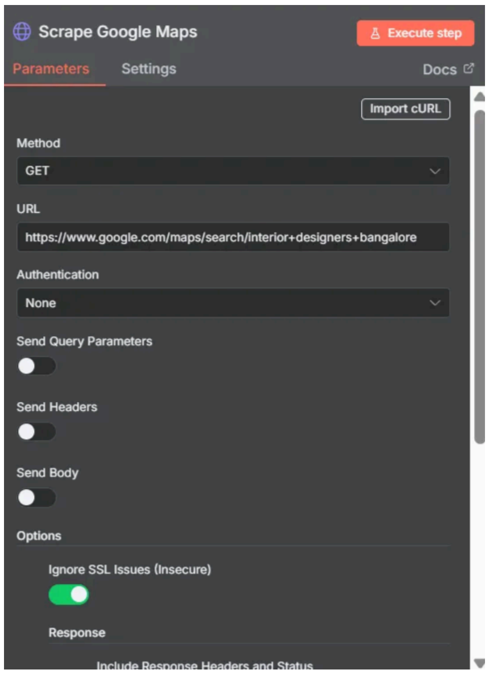
2. Scrape Google Maps

Next, we add an **HTTP Request node**. For this example, I’m using it to get business listings for **interior designers in Bangalore**. This node fetches all the websites of the businesses we want to collect emails from.

Example query:

```
https://www.google.com/maps/search/interior+designers+bangalore
```

Sample output :


Scrape Google Maps

Method: GET

URL: https://www.google.com/maps/search/interior+designers+bangalore

Authentication: None

Send Query Parameters: off

Send Headers: off

Send Body: off

Options

Ignore SSL Issues (Insecure): on

Response

Include Response Headers and Status

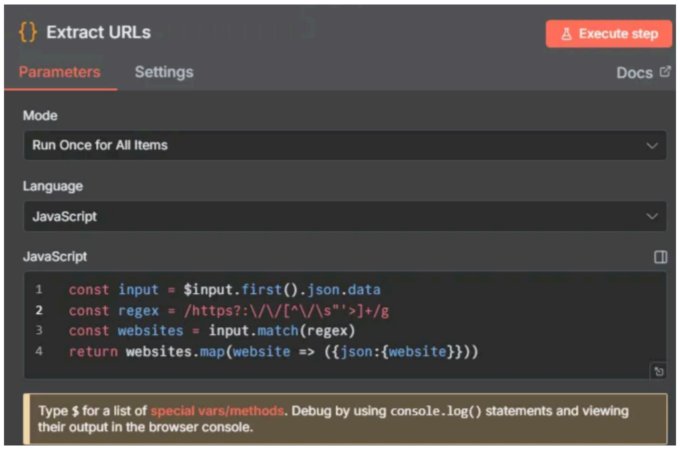
3. Extract URLs

Next, we use a **URL node** to extract the website URLs we need. As mentioned in the previous step, we're still working with **interior designers in Bangalore**. The data we get from the HTTP request comes in **JSON format**, so this node helps us pick out just the URLs we want to work with in the next steps.

Example query:

```
const input = $input.first().json.data
const regex = /https?:\/\/[^\/\s"'>]+/g
const websites = input.match(regex)
return websites.map(website => ({json:{website}}))
```

Sample output:



Mode

Run Once for All Items

Language

JavaScript

JavaScript

```javascript
const input = $input.first().json.data
const regex = /https?:\/\/[^\/\s"'>]+/g
const websites = input.match(regex)
return websites.map(website => ({json:{website}}))
```

Type $ for a list of special vars/methods. Debug by using console.log() statements and viewing their output in the browser console.

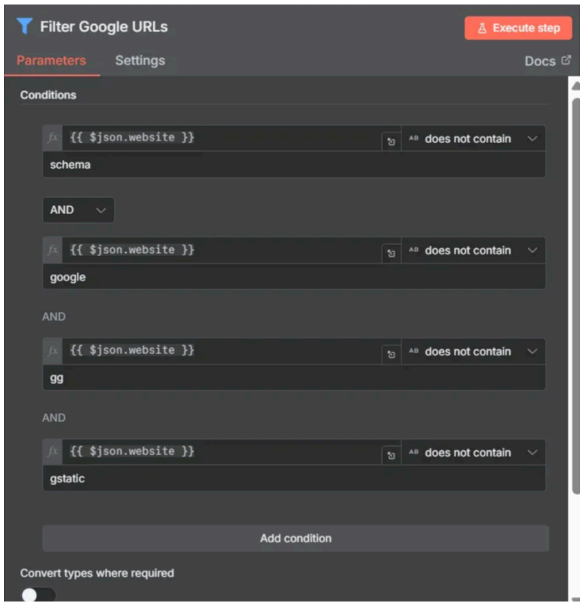
4. Filter Google URLS

After that, we use a **Filter node** to keep only the valid URLs. This step removes all the unnecessary or invalid links, so we're left with just the websites we actually want to scrape for emails.

The node applies the following conditions:

```
{{ $json.website }} does not contain "schema"
AND
{{ $json.website }} does not contain "google"
AND
{{ $json.website }} does not contain "gg"
AND
{{ $json.website }} does not contain "gstatic"
```

sample output:



Conditions

{{ $json.website }} does not contain schema

AND

{{ $json.website }} does not contain google

AND

{{ $json.website }} does not contain gg

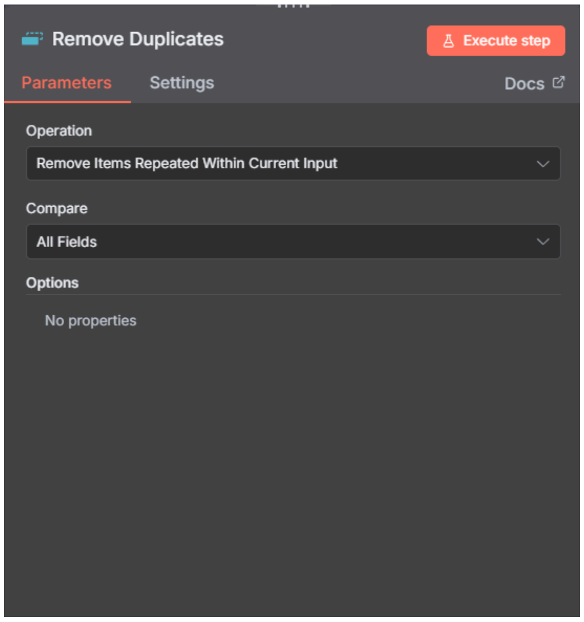
AND

{{ $json.website }} does not contain gstatic

Convert types where required

5. Remove Duplicates

Next, we use the **Remove Duplicates node** to get rid of any repeated website URLs. Sometimes, the same business can appear multiple times in the listings, and we don't want to scrape the same site twice. This step ensures that each website is **unique**, which makes the workflow faster and the final data much cleaner.
Sample out:



Operation: Remove Items Repeated Within Current Input

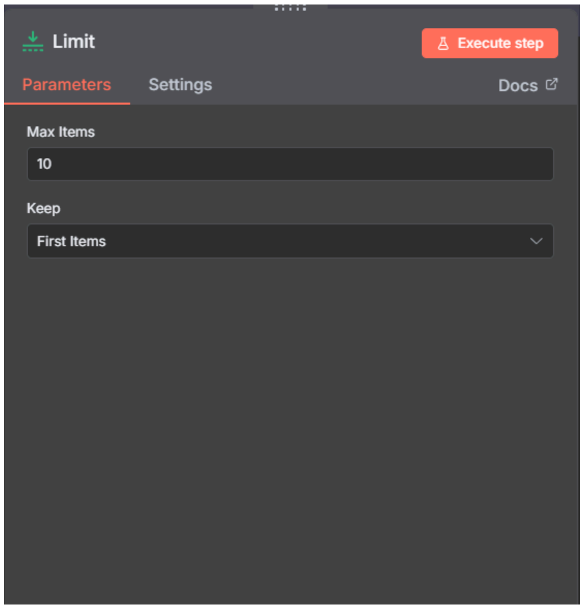
Compare: All Fields

Options

No properties

6. Limit

You can also **optionally limit the number of URLs processed** in a single run, like only taking the first 50 businesses. This is really useful because it helps **control the execution time** of your workflow and prevents overloading APIs or your system. It's a handy way to test your workflow quickly or handle large datasets without slowing everything down.
Sample output:



Max Items
10
Keep
First Items

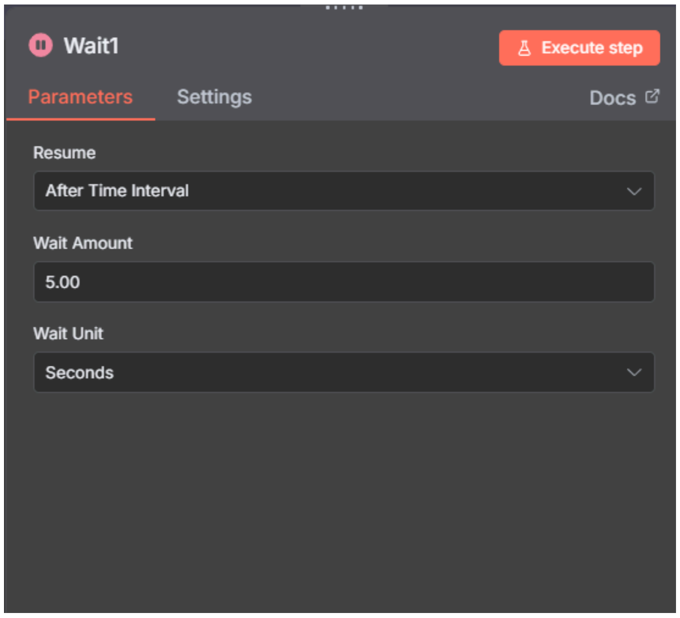
7. Loop Over Items

The workflow then goes through each of the filtered business URLs, one by one (I've set it to process them individually here). For each website, it first **pauses briefly** to avoid hitting rate limits. Then it **scrapes the site for contact information** and **extracts any email addresses** it finds. After that, it waits a little again before moving on to the next URL. This way, everything runs smoothly without overwhelming the websites or your workflow.

8. Wait

Next, we add a **Wait node**. This introduces a **short pause between HTTP requests** to follow good web scraping practices. It helps make sure we don't overload the websites and also **reduces the risk of getting blocked** while the

workflow runs.
Sample output:



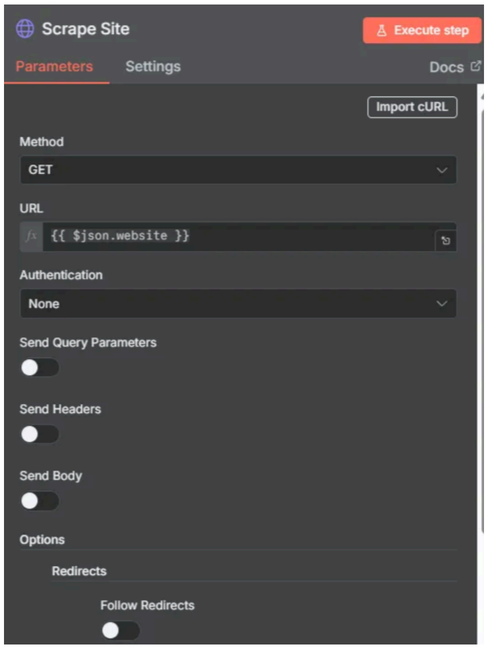
Resume
After Time Interval
Wait Amount
5.00
Wait Unit
Seconds

9. Scrape Site

Through the **HTTP Request node**, the workflow **sends a request to each business website**. This is how it visits each site to grab the information we need, like in our case it's email addresses.

```
{{ $json.website }}
```

Sample output:



Method: GET

URL: {{ $json.website }}

Authentication: None

Send Query Parameters

Send Headers

Send Body

Options

Redirects

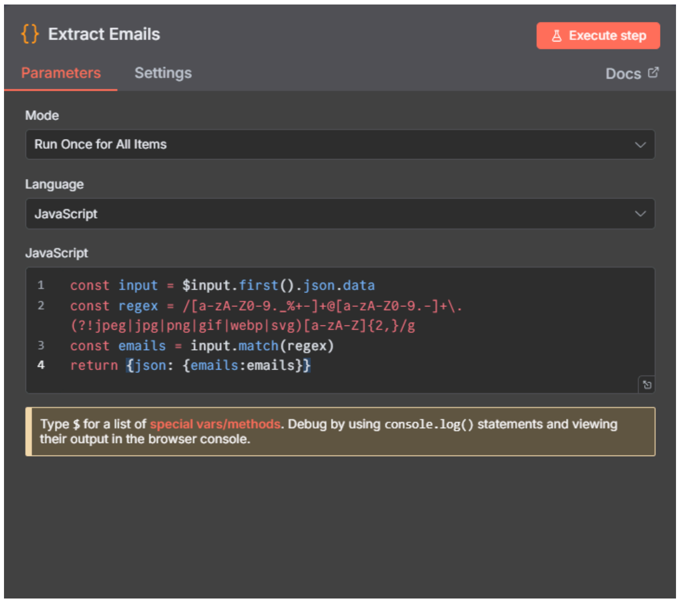
Follow Redirects

10. Extract Emails

Next, we use a **URL node** to scan the content of each website. It goes through the pages using **regex** to find and collect valid email addresses, like `contact@domain.com`, so we can gather all the contact info we need.

```
const input = $input.first().json.data
const regex = /[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.(?!jpeg|jpg|png|gif|webp
|svg)[a-zA-Z]{2,}/g
const emails = input.match(regex)
return {json: {emails:emails}}
```

Sample output:



Mode

Run Once for All Items

Language

JavaScript

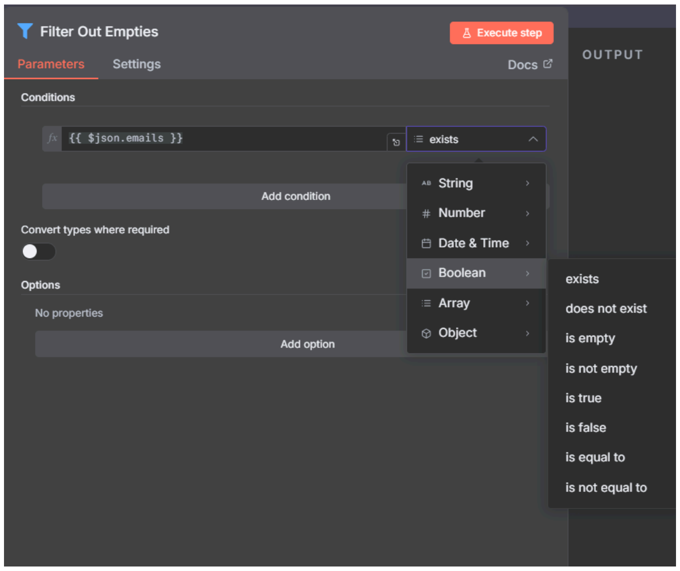
JavaScript

```javascript
const input = $input.first().json.data
const regex = /[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.
(?!jpeg|jpg|png|gif|webp|svg)[a-zA-Z]{2,}/g
const emails = input.match(regex)
return {json: {emails:emails}}
```

Type $ for a list of special vars/methods. Debug by using console.log() statements and viewing their output in the browser console.

11. Filter Out Empties

After extracting emails, we use a **Filter node** to remove any entries that **don't have valid email addresses**. This step is important because sometimes websites might not have emails listed, or the data we grabbed could be empty or irrelevant. Filtering ensures that the workflow only keeps **useful, actionable contacts**, making the final dataset cleaner and more reliable.
Sample output:



Conditions

`{{ $json.emails }}` exists

Convert types where required

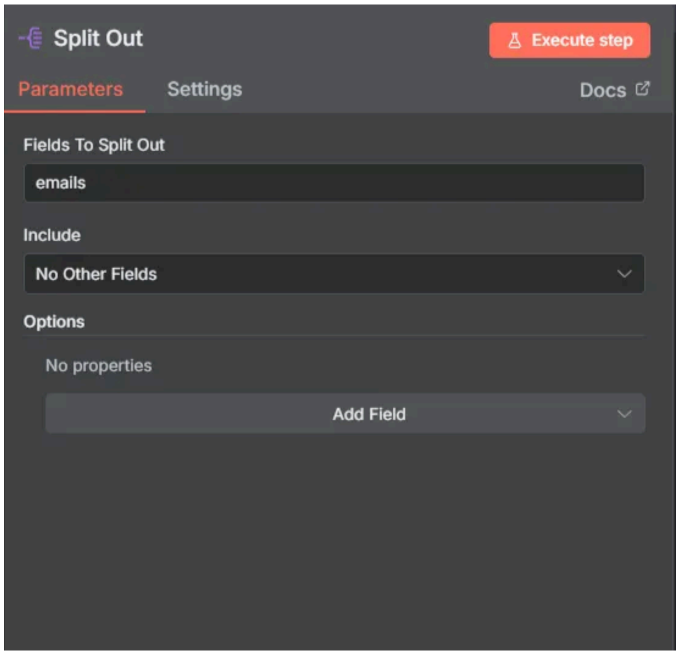
Options

No properties

12. Split Out

- Splits multiple email entries found on a single site into individual rows for easier handling and export.

```
{{ $json.emails }}
```



Fields To Split Out: emails

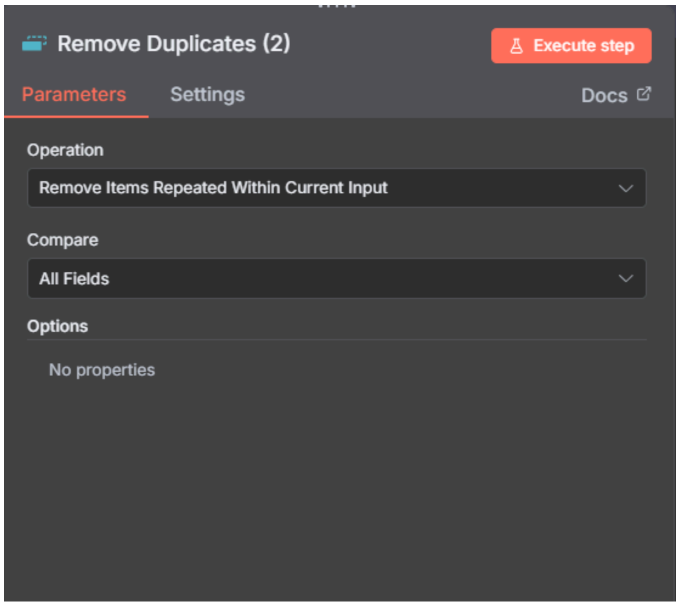
Include: No Other Fields

Options

No properties

13. Remove Duplicates (2)

- Ensures the final list of emails is unique — removes duplicates that may occur across multiple businesses.



Operation
Remove Items Repeated Within Current Input

Compare
All Fields

Options
No properties

Add to Google Sheet — Prerequisites

To allow n8n to read/write data to your Google Sheets, you must enable **Google Sheets API** and generate authentication credentials.

1. Go to Google Cloud Console

- Log in with the **Google account** that owns the Google Sheet or will be used for API access.

2. Create a New Project

1. Click on the project dropdown → **New Project**.
2. Name it (e.g., `n8n Google Sheets Integration`).
3. Click **Create**.

3. Enable Google Sheets API

1. In the left sidebar, go to: **APIs & Services → Library**
2. Search for **Google Sheets API**.
3. Click **Enable**.

> (Optional) You can also enable Google Drive API if you plan to access sheets by file ID or search Google Drive later.

4. Configure OAuth Consent Screen

1. In the left menu → **APIs & Services → OAuth consent screen**.
2. Choose **External** (recommended).
3. Fill out:
    - App name (e.g., `n8n Integration`)
    - User support email (your Gmail)
    - Developer contact email
4. Save and **add test users** (your Gmail ID).
5. Click **Save and Continue** until it's done.

5. Create OAuth 2.0 Credentials

1. Go to **APIs & Services → Credentials → + Create Credentials → OAuth client ID**.
2. Choose **Web Application**.
3. Under "Authorized redirect URIs", add:

    ```
    https://n8n.io
    http://localhost:5678/rest/oauth2-credential/callback
    ```

    (If running on a remote server, replace `localhost` with your domain or IP, e.g.:`https://yourdomain.com/rest/oauth2-credential/callback`)
4. Click **Create**.
5. Copy the generated:
    - **Client ID**
    - **Client Secret**

6. Add Credentials in n8n

1. In your n8n dashboard → **Credentials → + New → Google Sheets OAuth2**.
2. Paste:
    - **Client ID**
    - **Client Secret**
3. Set **Scope** to:

    ```
    https://www.googleapis.com/auth/spreadsheets
    ```

4. Click **Connect** and complete the Google authorization flow.

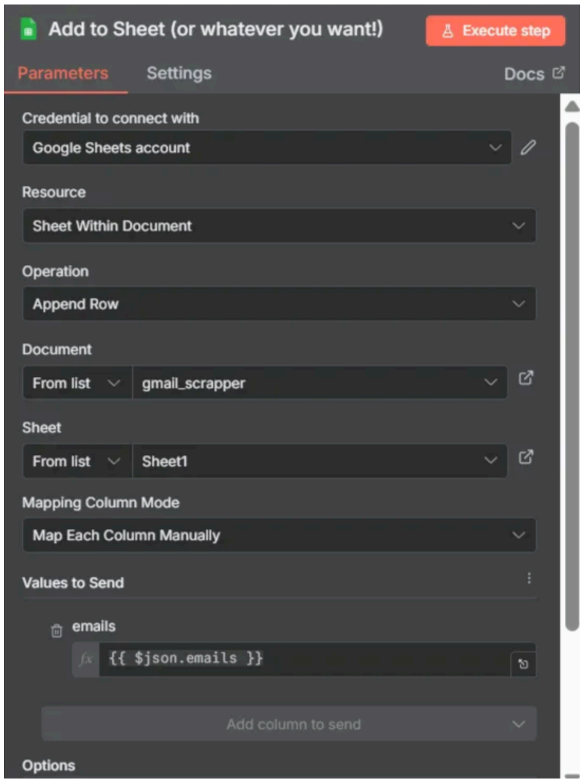
7. Use the Credential in "Add to Google Sheet" Node

- Select the credential you just created.
- Choose your **Spreadsheet ID** or connect dynamically.
- Map the fields (e.g., Business Name, Email, URL).

Once saved, the workflow will automatically append each record to your Google Sheet.

Add to Google Sheet

- Appends the cleaned and unique email records to a Google Sheet.
- Columns typically include:
    - Business Name
    - Website URL
    - Extracted Email(s)
    - Google Maps Link
    - Timestamp (optional)


Add to Sheet (or whatever you want!)

Credential to connect with: Google Sheets account

Resource: Sheet Within Document

Operation: Append Row

Document: From list — gmail_scrapper

Sheet: From list — Sheet1

Mapping Column Mode: Map Each Column Manually

Values to Send

emails

```
{{ $json.emails }}
```

Workflow Features

- 🔄 Fully automated data collection from Google Maps and business sites.
- ✨ Built-in deduplication and data cleaning.
- 📊 Automatic export to Google Sheets.
- 🕒 Configurable delays for compliance and performance control.
- 🔍 Adaptable for any business category or location.