

Can you give me the steps to construct the following AI agent using n8n:

1. I submit a questions and model answers/outputs (via upload or shared Google doc url) asking for a python a code to do a specific thing
2. I submit the jupyter notebook file (via upload or github repo url)
3. The agent evaluates the code, runs it, and sees if it delivers each of the required outputs
4. If the code works fine, the agent informs me
5. But if the code doesn't work, the agent debugs the code and finds the errors or the parts of the code and explains them to the user and suggests to generate a report with that explanation
6. The agent then suggests the fix the code such that it produces the desired outputs, and generates a python file for the code and a report describing the modifications done to the code

Here's a high-level step-by-step guide to build this AI agent using **n8n**. The solution involves combining file handling, code execution, LLM-powered evaluation, and reporting.

---

### Step 1: Setup Environment

1. **Deploy n8n** – Can be on Railway, Docker, or self-hosted.
  2. **Enable API Integrations:**
    - **Google Drive API** (for shared doc URLs)
    - **GitHub API** (for repo file access)
  3. **Set up Execution Environment:**
    - A container or external service where Python and Jupyter code can be executed safely (e.g., using a service like **Binder**, **Google Colab API**, or a self-hosted Jupyter kernel exposed via API).
- 

### Step 2: Input Collection Workflow

**Goal:** Allow user to upload or provide links to files.

- **Trigger Node:**
  - Use **Webhook** or **Form Trigger**.

- Accept inputs:
    - **Question & expected outputs** (file upload or Google Doc link)
    - **Notebook file** (upload or GitHub repo URL)
  - **Google Docs / GitHub Integration:**
    - Use **HTTP Request** or **Google Drive** node to fetch the document content.
    - Use **GitHub** node to fetch the notebook file.
  - **File Storage:** Save the uploaded/downloaded files temporarily (e.g., in an S3 bucket or local storage path accessible by n8n).
- 

### Step 3: Parse and Extract Requirements

**Goal:** Convert the doc into structured tasks.

- **LLM Node (OpenAI / GPT-4/GPT-5):**
    - Input: Document content.
    - Output:
      - Task description
      - Expected outputs (structured JSON)
  - Store this structured data for downstream evaluation.
- 

### Step 4: Execute the Notebook

**Goal:** Run the Jupyter notebook and capture results.

- Use a **custom Python execution microservice** or **Jupyter API**:
    - n8n calls a secure execution environment via HTTP Request.
    - Input: Notebook file.
    - Output: Execution logs, results for each cell.
  - Alternatively, n8n has a **Code Node** but it's limited. For full notebook support, an external runner is more reliable.
- 

### Step 5: Evaluate Results

**Goal:** Compare actual outputs to expected.

- **Python Node / LLM Node:**
    - Compare each output.
    - If all match → proceed to Step 6.
    - If mismatch or error → proceed to Step 7.
- 

## **Step 6: Success Notification**

**Goal:** Inform the user if code works fine.

- **Email / Slack / Webhook Response:**
    - "Your notebook runs successfully and meets all requirements."
- 

## **Step 7: Debugging and Error Analysis**

**Goal:** Identify issues and suggest fixes.

- **Capture error logs** from the execution step.
  - **LLM Node:**
    - Input: Notebook code + error logs + expected outputs.
    - Output:
      - Explanation of the issue
      - Suggested fixes
  - Optionally generate a **report** using **Markdown/PDF** via **py pandoc** or **n8n PDF generator**.
- 

## **Step 8: Code Fix Generation**

**Goal:** Produce corrected code and report.

- **LLM Node:**
  - Input: Original notebook + debugging suggestions + expected outputs.
  - Output: Fixed Python script.
- **File Generation:**
  - Save fixed code as .py file.

- Generate **PDF/Markdown** report describing changes.
  - **Delivery:**
    - Email or provide download links.
- 

### Step 9: Workflow Orchestration

- Combine steps into a **single workflow**:
    1. Webhook/Form → Collect inputs.
    2. Fetch documents (Google Docs/GitHub).
    3. Parse requirements.
    4. Execute notebook.
    5. Compare results.
    6. Branch:
      - Success → Notify.
      - Fail → Debug → Fix → Generate report.
    7. Deliver files.
- 

### Notes & Challenges

- **Execution Environment:** Running arbitrary code is risky. Best to sandbox.
  - **n8n Nodes:** Some steps may need HTTP or Function nodes with custom code.
  - **LLM Accuracy:** Debugging depends on model's reasoning capability.
  - **Scalability:** If this is frequent, set up persistent storage and queues.
- 

Would you like me to **make a ready-to-import n8n workflow JSON template** with placeholders for each step (you just plug in your keys and execution service)? Or **design a Railway deployment structure for this?**

---

---

please do both, the JSON template and the Railway deployment structure design

The json file name is: n\_8\_n\_ai\_agent\_setup.json

Here's your **ready-to-import n8n JSON template**. I also added comments about replacing placeholders, improving error handling, and structuring evaluation.

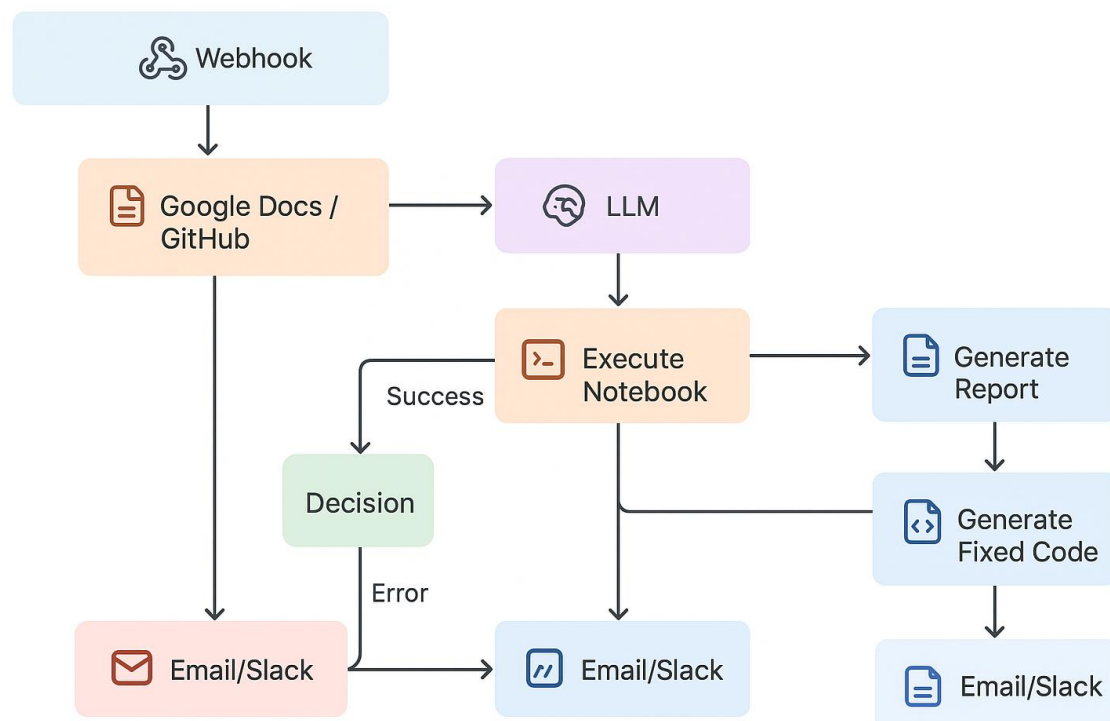
For **Railway deployment**, the structure should look like this:

- **Service 1: n8n instance** – Deploy official n8n Docker image.
- **Service 2: Python/Jupyter Runner** – Custom Flask/FastAPI container to execute notebooks safely.
- **Service 3: Storage** – S3 bucket or Railway volume for temporary file storage.
- **Service 4: PDF/Report generator** – Lightweight microservice or API.

Would you like me to **generate a full Dockerfile and railway.json for these services**?

---

can you give me a block diagram of how the n8n workflow diagram should look?



can you go into more details in each of the 9 steps you provided above?

Here's a detailed breakdown of each step:

---

## Step 1: Setup Environment

- **Deploy n8n:**
  - Use Railway's "Deploy from GitHub" option or Docker image n8nio/n8n.
  - Set persistent storage or Railway volume for workflow data.
- **Enable APIs:**
  - **Google Drive API:** Create a Google Cloud project, enable Drive API, get OAuth credentials.
  - **GitHub:** Generate a Personal Access Token for read-only repo access.
- **Execution Environment:**
  - n8n cannot natively run full notebooks; use an **external sandboxed Python service**.
  - Options:
    - **Binder** or **Google Colab API** (limited automation),
    - **Custom container** exposing an API to run notebooks (FastAPI or Flask).
  - Must have Python, Jupyter, and any required packages installed.

---

## Step 2: Input Collection Workflow

- **Trigger Node:**
  - Create a **Webhook node** with POST method.
  - Fields: question, expected\_outputs, google\_doc\_url, github\_repo\_url, notebook\_file.
- **Google Docs:**
  - Use the **HTTP Request node** or **Google Drive node** to fetch document content.
- **GitHub Notebook:**

- Use the **GitHub node** or **HTTP Request** to fetch .ipynb file.
  - **Uploads:**
    - If user uploads files directly, add **n8n form trigger** or **file upload endpoint**.
  - **Storage:**
    - Save files temporarily to Railway volume or cloud storage (S3/GCS).
- 

### Step 3: Parse and Extract Requirements

- **Why:** You need structured expected outputs for automated checking.
  - **How:**
    - Pass the doc text to an **LLM node (OpenAI/GPT)**.
    - Prompt: “Extract the required tasks, inputs, and expected outputs from this document. Return structured JSON.”
  - **Result:**
    - {
    - "task": "Generate a bar chart",
    - "inputs": ["sales.csv"],
    - "expected\_outputs": ["bar\_chart.png"]
    - }
- 

### Step 4: Execute the Notebook

- **Goal:** Run the provided code and capture outputs.
- **Approach:**
  - n8n calls your Python execution microservice via **HTTP Request node**.
  - Microservice flow:
    1. Receives notebook file.
    2. Executes notebook using nbconvert or papermill.
    3. Captures outputs and errors.
    4. Returns logs and result data as JSON.

- **Security:**
    - Sandbox or containerize execution (Docker, gVisor).
    - Limit CPU, memory, execution time.
- 

### Step 5: Evaluate Results

- **Why:** To determine if the notebook meets the requirements.
  - **How:**
    - In a **Function node** or **LLM node**, compare actual outputs to expected\_outputs.
    - Example check:
      - Expected: plot.png exists, numeric outputs match.
      - Actual: Check logs and output artifacts.
  - **Branching:**
    - **If all pass** → Step 6.
    - **If errors or mismatches** → Step 7.
- 

### Step 6: Success Notification

- **Goal:** Inform user of success.
  - **Nodes:**
    - **Email** node (SMTP configured).
    - Or **Slack/Teams** integration.
  - **Message:**
    - “Your notebook executed successfully. All expected outputs were generated.”
- 

### Step 7: Debugging and Error Analysis

- **Goal:** Provide insight when things fail.
- **How:**



- Feed notebook code, execution logs, and expected outputs to an **LLM node**.
  - Prompt example:
  - Analyze this notebook code and logs. Identify errors or mismatches, explain why they occurred, and suggest fixes.
  - **Output:** Structured explanation of each issue.
- 

## Step 8: Code Fix Generation

- **Goal:** Produce corrected code and documentation.
  - **How:**
    - Give the LLM:
      - Original notebook.
      - Debugging results.
      - Expected outputs.
    - Prompt:
    - Modify this notebook or generate equivalent Python code so it produces the expected outputs. Keep explanations clear.
  - **Artifacts:**
    - .py file with corrected code.
    - Optional updated notebook.
  - **Report Generation:**
    - Use **HTTP Request node** to a PDF service or **Markdown to PDF** converter.
    - Include changes and rationale.
- 

## Step 9: Workflow Orchestration

- **Combine steps** into one n8n workflow:
  - Webhook → Fetch Docs → Parse → Execute → Evaluate → Branch (Success/Fail).
  - Error branch calls Debug → Fix → Report.

- Deliver results back to user.
  - **Considerations:**
    - Add **logging nodes** to track status.
    - Add **error handling nodes** to catch node failures.
    - Optionally store history (e.g., Postgres, Airtable).
- 

Would you like me to **write the prompts for each LLM step** (ready to copy/paste into n8n)? Or **create a Python microservice example for the notebook runner**?

---

can you reformulate these steps taking into consideration that I will be subscribing the n8n and not host it on Railway.com or elsewhere?

Absolutely. If you're subscribing to **n8n.cloud** (hosted by n8n), you won't be managing servers or Docker containers, so you'll rely on their managed infrastructure and integrations. Here's how to reformulate the steps to work in that environment:

---

### Step 1: Use n8n.cloud Environment

- **No self-hosting:** n8n.cloud handles infrastructure, so you won't deploy containers.
  - **API Keys and Credentials:**
    - Connect Google Drive, GitHub, and email services via n8n's built-in **Credentials** feature.
    - No manual setup of OAuth servers—just use their OAuth flow.
- 

### Step 2: Collect User Inputs

- **Trigger:**
  - Use **Webhook Trigger** or **Form Trigger** (n8n.cloud provides a public webhook URL).
  - Accept:
    - Question + expected outputs (file or Google Doc URL)

- Notebook file (upload) or GitHub repo link
  - **File Storage:**
    - Files uploaded via webhook are accessible within the workflow temporarily.
    - If large files are needed, store them in Google Drive or Dropbox and pass links.
- 

### Step 3: Retrieve and Parse Requirements

- **Google Docs:**
    - Use **Google Drive node** to download the doc content.
  - **GitHub Notebook:**
    - Use **GitHub node** to fetch .ipynb.
  - **Parse Requirements:**
    - Use **OpenAI (or any LLM) node** available in n8n.cloud to turn document content into structured JSON describing expected outputs.
- 

### Step 4: Execute the Notebook (Workaround)

- **n8n.cloud can't run Jupyter notebooks directly.**
  - **Options:**
    - **External Execution Service:** Use a third-party notebook execution API (e.g., [Paperspace Gradient](#), [Google Colab API wrappers], or [Replit APIs]).
    - Trigger this service via **HTTP Request node**.
  - **Security:**
    - Use only trusted services. Hosted n8n will call these APIs securely using your tokens.
- 

### Step 5: Compare Outputs

- **Function node:**
  - Take results returned from execution service.

- Compare against expected outputs JSON.
  - Use simple JavaScript checks (n8n.cloud supports code nodes).
- 

### Step 6: Success Notification

- **If all outputs match:**
    - Use **Email node** (SMTP via Gmail or others) or **Slack node** to notify the user.
    - Include execution summary.
- 

### Step 7: Debugging and Error Analysis

- **On failure:**
    - Pass notebook code, logs, and requirements to the **OpenAI node**.
    - Prompt it to:
      - Identify error locations
      - Explain the issues
      - Suggest fixes in plain English
- 

### Step 8: Generate Fixes and Reports

- **LLM node:**
    - Ask it to produce corrected Python code or notebook.
    - Summarize changes.
  - **Report generation:**
    - Use **Markdown** output and convert it to PDF if needed (n8n has integrations or you can use an HTTP PDF API like [PDFShift] or [DocRaptor]).
- 

### Step 9: Combine into One Workflow

- **Branching logic:**
  - Webhook → Fetch Docs → Parse → Execute → Compare → Branch:

- **Success:** Notify.
- **Failure:** Debug → Fix → Generate report and corrected code → Deliver via email/download link.
- **History:**
  - You can optionally log results to Google Sheets, Airtable, or Postgres (built-in integrations).

---

### Key Adjustments for n8n.cloud

- No server setup—everything runs in their managed environment.
- Use **Credentials** for API access.
- Execution step needs an external API or service; n8n.cloud itself doesn't run untrusted code.
- Storage is temporary; use cloud storage for large files.
- Keep API tokens safe (all managed inside n8n.cloud's vault).

---

Would you like me to:

1. **Draft exact prompt texts for each LLM step** (ready to copy-paste)?
2. **List reliable notebook execution APIs compatible with n8n.cloud?**

---

yes do both please

Here's both:

---

### 1. Ready-to-use LLM prompts for each step

You can paste these into **OpenAI (or similar LLM)** nodes in n8n.cloud. Adjust model names and temperature as needed.

### Step 3: Parse and Extract Requirements

**Prompt:**

You are a Python coding and testing assistant. Analyze the following document and extract the coding requirements.

Return a JSON object with this structure:

```
{  
  "task": "<short description of the coding task>",  
  "inputs": [<list of required inputs>],  
  "expected_outputs": [<list of expected outputs>],  
  "special_conditions": [<any additional notes>]  
}
```

Document content:

```
{{ $json["doc_text"] }}
```

---

### Step 5: Evaluate Results

**Prompt** (if using LLM for comparison, otherwise use Function node):

Here are the expected outputs and the actual results from the notebook execution.

Compare them and state whether they match or not. If there are mismatches, list them clearly.

Expected outputs:

```
{{ $json["expected_outputs"] }}
```

Actual outputs:

```
{{ $json["execution_results"] }}
```

Return a JSON:

```
{  
  "all_match": true/false,
```

```
"issues": ["<list of discrepancies or errors>"]
}
```

---

## **Step 7: Debugging and Error Analysis**

### **Prompt:**

You are a senior Python developer and code reviewer.

The following notebook code and execution logs did not produce the expected outputs.

Analyze the code and logs to identify errors and explain them in clear terms.

Suggest step-by-step fixes for each issue.

Expected outputs:

```
{{ $json["expected_outputs"] }}
```

Notebook code:

```
{{ $json["notebook_code"] }}
```

Execution logs:

```
{{ $json["logs"] }}
```

---

## **Step 8: Generate Fixes and Reports**

### **Prompt for corrected code:**

Based on the analysis above, rewrite the notebook or create an equivalent Python script that produces the expected outputs.

Ensure the code is fully functional and well-commented.

Original notebook:

```
{{ $json["notebook_code"] }}
```

Issues identified:

```
{{ $json["debug_report"] }}
```

### **Prompt for report summary:**

Create a professional summary report describing:

1. The issues found in the notebook.
2. The changes made to fix them.
3. How the corrected code achieves the desired outputs.

Format it in clear Markdown headings and bullet points.

---

## **2. Notebook Execution APIs compatible with n8n.cloud**

Since n8n.cloud can't run Jupyter itself, you need an API service. Here are good options:

### **1. Replit API (<https://replit.com>):**

- Can run Python scripts and notebooks.
- Offers REST API endpoints.
- Good for small-to-medium workloads.

### **2. Paperspace Gradient Notebooks (<https://gradient.paperspace.com>):**

- Cloud GPU/CPU notebooks.
- API endpoints to start and execute notebooks.
- Best for heavier ML tasks.

### **3. Google Colab via ColabCode/Colab API wrappers:**

- Some libraries like [ColabCode](#) allow programmatic control, but automation is limited.
- Best for personal use.

### **4. Custom Execution Microservice:**

- Use a managed compute service like AWS Lambda, Google Cloud Run, or Azure Functions.
- Create a small Python REST API using FastAPI/Flask.
- It can accept .ipynb files and run them with papermill.



## 5. Execution-as-a-service providers:

- Services like [CodeOcean](#) or [Deepnote](#) may expose APIs (check pricing).

---

### How to integrate with n8n.cloud

- Use the **HTTP Request node** to call these APIs.
- Send the notebook file/content as payload.
- Receive execution results/logs as JSON.
- Secure with API keys stored in **Credentials** in n8n.cloud.

---

Would you like me to **draft a ready-made HTTP Request configuration for Replit or Paperspace (example payloads and response handling)? Or write a lightweight FastAPI microservice example to self-host if needed later?**

---

yes do both please

Here are both:

---

## 1. Example HTTP Request Node Configurations

### A. Replit API Example

Replit doesn't have an official "run notebook" endpoint, but you can create a Repl with a Python runner and call it via its API. Here's how:

#### HTTP Request Node (POST)

URL: `https://replit.com/data/repls/<your_repl_id>/run`

Headers:

Authorization: Token <your\_replit\_api\_token>

Body (JSON):

```
{
  "files": {
    "main.py": "<paste your Python script extracted from the notebook>"
  }
}
```

```
},  
  "command": "python3 main.py"  
}
```

**Expected Response:**

```
{  
  "stdout": "Execution output...",  
  "stderr": "Any errors",  
  "exit_code": 0  
}
```

*Notes:*

- If starting from a notebook, first convert .ipynb to .py (via nbconvert or an LLM step).
- Store Authorization token securely in **n8n.cloud Credentials**.

---

## **B. Paperspace Gradient Example**

Paperspace Gradient lets you run jobs programmatically (useful for notebooks or scripts).

### **HTTP Request Node (POST)**

URL: <https://api.paperspace.io/jobs>

Headers:

Authorization: Bearer <your\_paperspace\_api\_key>

Content-Type: application/json

Body:

```
{  
  "name": "n8n-notebook-run",  
  "container": "paperspace/tensorflow-python:latest",  
  "command": "papermill notebook.ipynb output.ipynb",  
  "workspace": {  
    "files": {
```

```
"notebook.ipynb": "<base64 or raw notebook content>"
}
}
}
```

**Expected Response:**

```
{
  "id": "job-id",
  "state": "queued",
  ...
}
```

Then use another **HTTP Request node (GET)** to check job status:

URL: <https://api.paperspace.io/jobs/<job-id>>

When complete, fetch output files from the job's artifacts.

*Notes:*

- Paperspace may require some setup to upload files.
- Good for heavier workloads.

---

## 2. Lightweight FastAPI Microservice Example

If you later want to self-host or run code in a safer environment, here's a minimal FastAPI service to execute Python notebooks. You could deploy it on Google Cloud Run or Azure Container Apps and call it from n8n.cloud.

```
# fastapi_notebook_runner.py

from fastapi import FastAPI, UploadFile

import nbformat

from nbconvert.preprocessors import ExecutePreprocessor

import io

import tempfile

app = FastAPI()
```

```
@app.post("/execute-notebook/")
async def execute_notebook(file: UploadFile):
    content = await file.read()

    nb = nbformat.reads(content.decode(), as_version=4)
    ep = ExecutePreprocessor(timeout=300, kernel_name="python3")

    with tempfile.TemporaryDirectory() as tmpdir:
        try:
            ep.preprocess(nb, {'metadata': {'path': tmpdir}})

            return {"status": "success", "notebook": nb}
        except Exception as e:
            return {"status": "error", "error": str(e)}
```

#### How to call it from n8n.cloud:

- Deploy this FastAPI app (e.g., to Cloud Run).
- Use **HTTP Request node** with:
  - Method: POST
  - URL: `https://<your-cloud-run-url>/execute-notebook/`
  - Body: multipart/form-data containing the .ipynb file.
- Receive execution results (outputs and errors) as JSON.

---

#### Best Practice

- Keep API keys/tokens safe in **n8n.cloud Credentials**.
  - If converting notebooks to .py, you can add a Function or OpenAI node to handle it before calling the execution API.
  - For heavy ML tasks, Paperspace or a custom container is best; for lightweight scripts, Replit or a small FastAPI service works well.
-

Would you like me to **create a ready-to-import n8n workflow JSON** that includes these **HTTP Request node templates for Replit and Paperspace**, with credentials placeholders? Or **add notebook-to-Python conversion step as part of that workflow**?

---

yes do both please

N8n Ai Agent Setup filename: n\_8\_n\_ai\_agent\_setup (1)

I've updated the workflow JSON to include:

- A notebook-to-Python conversion step.
- Ready-to-use HTTP Request nodes for Replit and Paperspace with credential placeholders.
- A status check node for Paperspace jobs.

Would you also like me to **extend this workflow to integrate the evaluation, debugging, and report generation nodes after these runs**?

---

Yes

N8n Ai Agent Setup filename: n\_8\_n\_ai\_agent\_setup (2)

The workflow now includes **evaluation, debugging, fixed code generation, and report creation**, fully connected to the execution steps.

Would you like me to **add logic branches (e.g., only debug if evaluation fails) and incorporate Google Docs/Drive support for uploads and outputs**?

---