

Memoria Práctica 6

Jaime Parra Jiménez

INTEGRACIÓN DE TECNOLOGÍAS Y SERVICIOS INFORMÁTICOS

Universidad de Almería

Correo: jj451@inlumine.ual.es

27 de noviembre de 2025

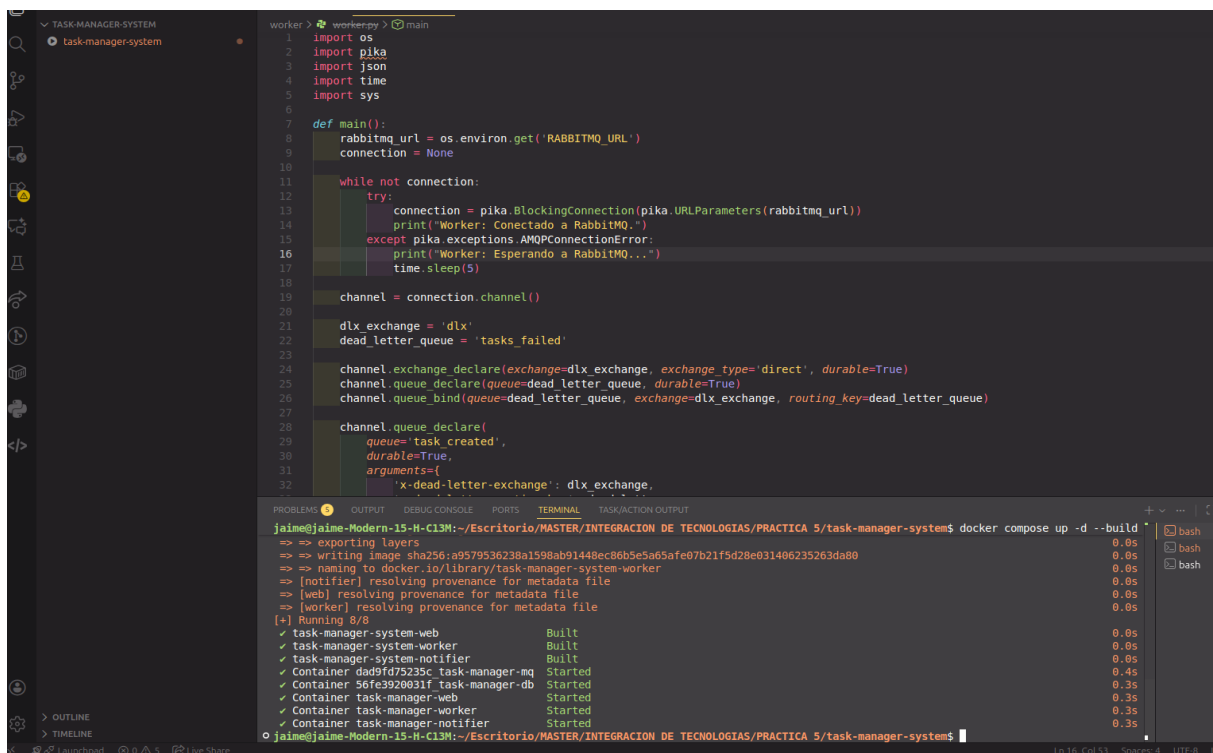
Índice general

1. Ejercicio Guiado 1	2
2. Ejercicio 1	7
3. Ejercicio 2	9
4. Ejercicio 3	11

Capítulo 1

Ejercicio Guiado 1

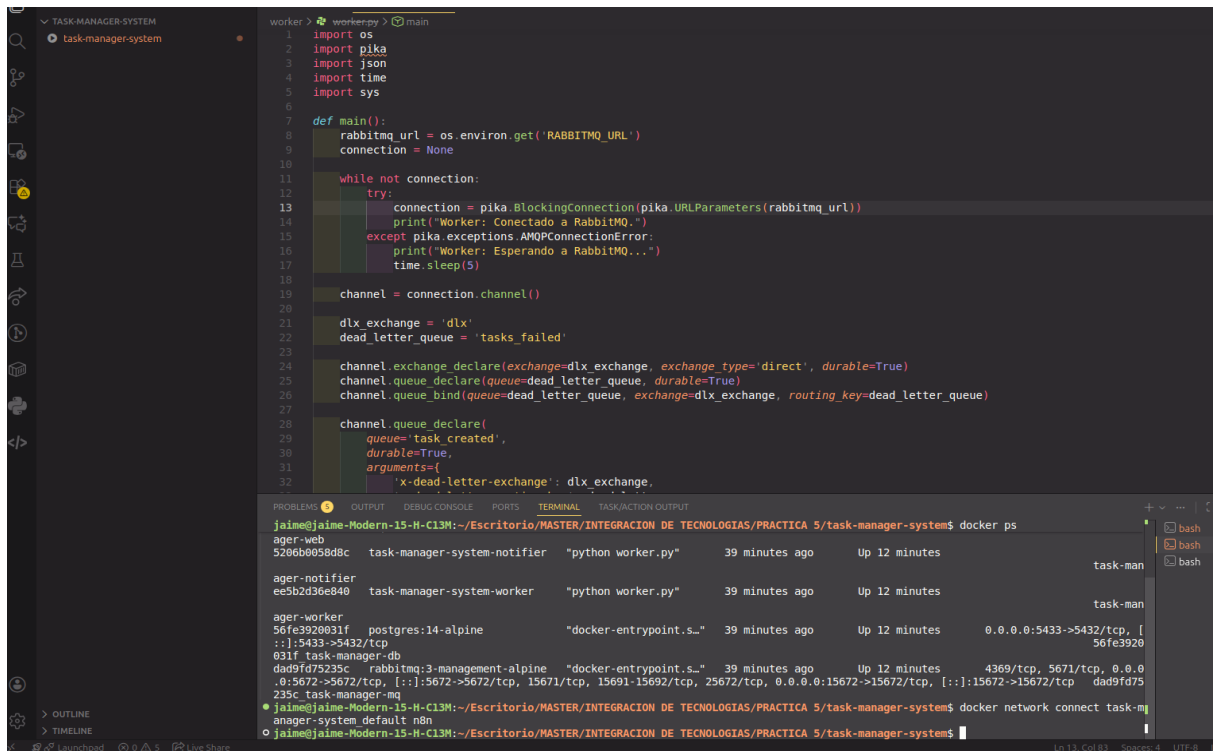
El objetivo del ejercicio guiado es conectar n8n el sistema de la práctica 5. n8n leerá datos de la base de datos PostgreSQL, y luego actuará como productor y consumidor de mensajes de RabbitMQ. El primer paso será preparar el entorno de microservicios. Para ello se debe iniciar la pila de servicios en segundo plano usando el comando `docker-compose up -d --build`.



```
1 import os
2 import pika
3 import json
4 import time
5 import sys
6
7 def main():
8     rabbitmq_url = os.environ.get('RABBITMQ_URL')
9     connection = None
10
11     while not connection:
12         try:
13             connection = pika.BlockingConnection(pika.URLParameters(rabbitmq_url))
14             print('Worker: Conectado a RabbitMQ.')
15         except pika.exceptions.AMQPConnectionError:
16             print('Worker: Esperando a RabbitMQ...')
17             time.sleep(5)
18
19     channel = connection.channel()
20
21     dlx_exchange = 'dlx'
22     dead_letter_queue = 'tasks_failed'
23
24     channel.exchange_declare(exchange=dlx_exchange, exchange_type='direct', durable=True)
25     channel.queue_declare(queue=dead_letter_queue, durable=True)
26     channel.queue_bind(queue=dead_letter_queue, exchange=dlx_exchange, routing_key=dead_letter_queue)
27
28     channel.queue_declare(
29         queue='task_created',
30         durable=True,
31         arguments={
32             'x-dead-letter-exchange': dlx_exchange,
```

```
jaim@jaim-Modern-15-H-C13M:~/Escritorio/MASTER/INTEGRACION DE TECNOLOGIAS/PRACTICA 5/task-manager-system$ docker compose up -d --build
=> exporting layers
=> writing image sha256:a9579536238a1598ab91448ec86b5e5a65afe07b21f5d28e031466235263da80
=> naming to docker.io/library/task-manager-system-worker
=> [notifier] resolving provenance for metadata file
=> [web] resolving provenance for metadata file
=> [worker] resolving provenance for metadata file
[+] Running 8/8
 ✓ task-manager-system-web          Built      0.0s
 ✓ task-manager-system-worker       Built      0.0s
 ✓ task-manager-system-notifier     Built      0.0s
 ✓ Container d4d9f475235c task-manager-mq Started   0.4s
 ✓ Container 56fe3920031f task-manager-db Started   0.3s
 ✓ Container task-manager-web       Started   0.3s
 ✓ Container task-manager-worker    Started   0.3s
 ✓ Container task-manager-notifier   Started   0.3s
jaim@jaim-Modern-15-H-C13M:~/Escritorio/MASTER/INTEGRACION DE TECNOLOGIAS/PRACTICA 5/task-manager-system$
```

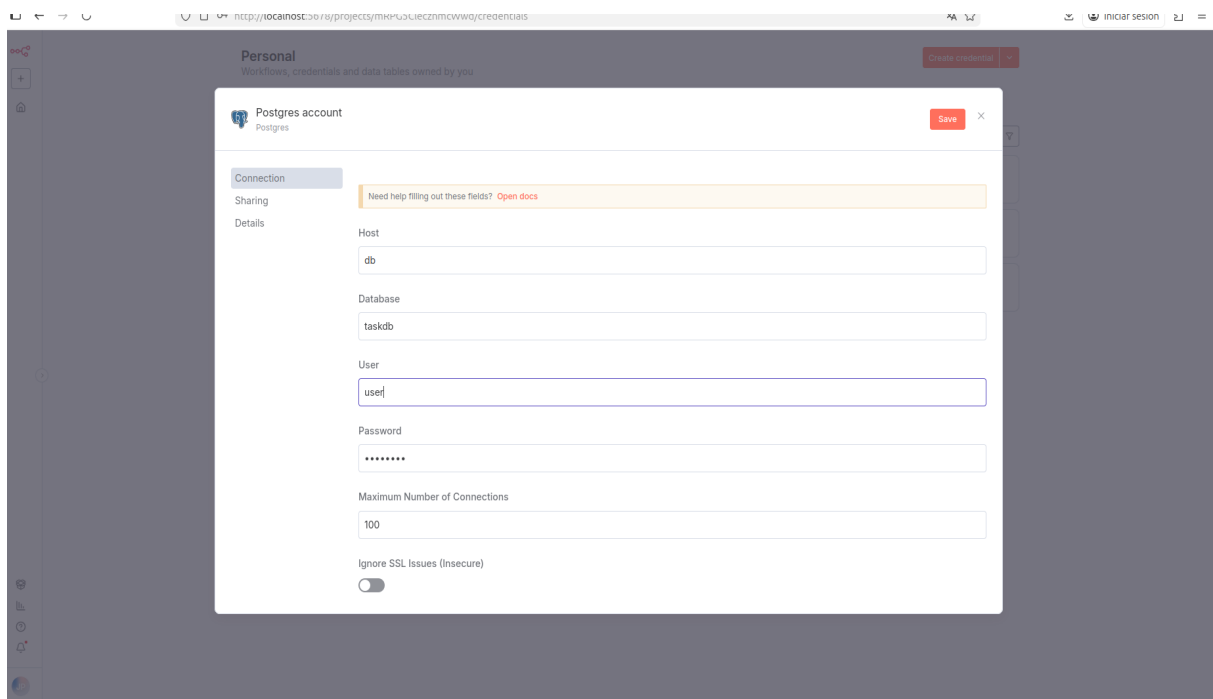
A continuación, es necesario conectar n8n a la red de microservicios. Para conectarlo se usará el siguiente comando.



```
worker > worker.py > main
1 import os
2 import pika
3 import json
4 import time
5 import sys
6
7 def main():
8     rabbitmq_url = os.environ.get('RABBITMQ_URL')
9     connection = None
10
11     while not connection:
12         try:
13             connection = pika.BlockingConnection(pika.URLParameters(rabbitmq_url))
14             print('Worker: Conectado a RabbitMQ.')
15         except pika.exceptions.AMQPConnectionError:
16             print('Worker: Esperando a RabbitMQ...')
17             time.sleep(5)
18
19     channel = connection.channel()
20
21     dlx_exchange = 'dlx'
22     dead_letter_queue = 'tasks_failed'
23
24     channel.exchange_declare(exchange=dlx_exchange, exchange_type='direct', durable=True)
25     channel.queue_declare(queue=dead_letter_queue, durable=True)
26     channel.queue_bind(queue=dead_letter_queue, exchange=dlx_exchange, routing_key=dead_letter_queue)
27
28     channel.queue_declare(
29         queue='task_created',
30         durable=True,
31         arguments={
32             'x-dead-letter-exchange': dlx_exchange,
```

```
jaime@jaime-Modern-15-H-C13M:~/Escritorio/MASTER/INTEGRACION DE TECNOLOGIAS/PRACTICA 5/task-manager-system$ docker ps
jaime@jaime-Modern-15-H-C13M:~/Escritorio/MASTER/INTEGRACION DE TECNOLOGIAS/PRACTICA 5/task-manager-system$ docker network connect task-man
jaime@jaime-Modern-15-H-C13M:~/Escritorio/MASTER/INTEGRACION DE TECNOLOGIAS/PRACTICA 5/task-manager-system$
```

Una vez conectado n8n a la red es necesario configurar las credenciales en n8n. La primera credencial es PostgreSQL. Le asignaremos el host, database, user, password y port.



Personal

Workflows, credentials and data tables owned by you

Create credential

Postgres account

Postgres

Save

Connection

Sharing

Details

Need help filling out these fields? [Open docs](#)

Host

db

Database

taskdb

User

user

Password

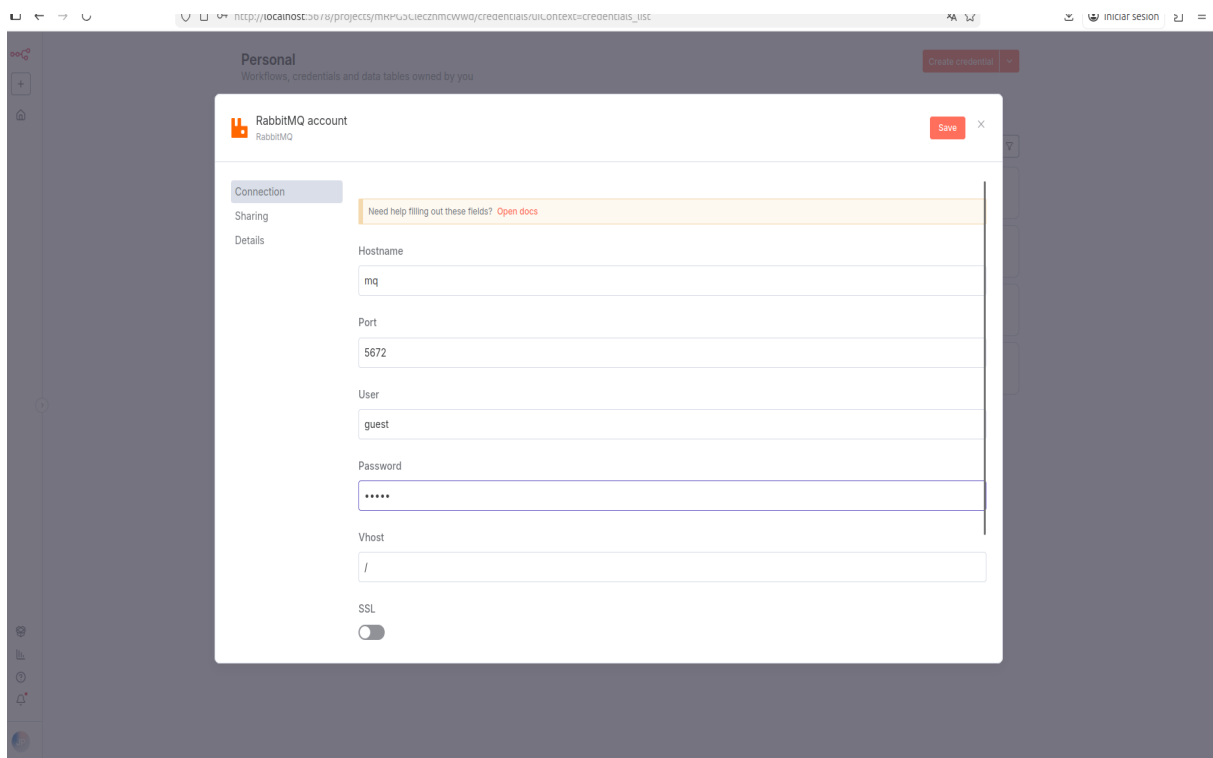
Maximum Number of Connections

100

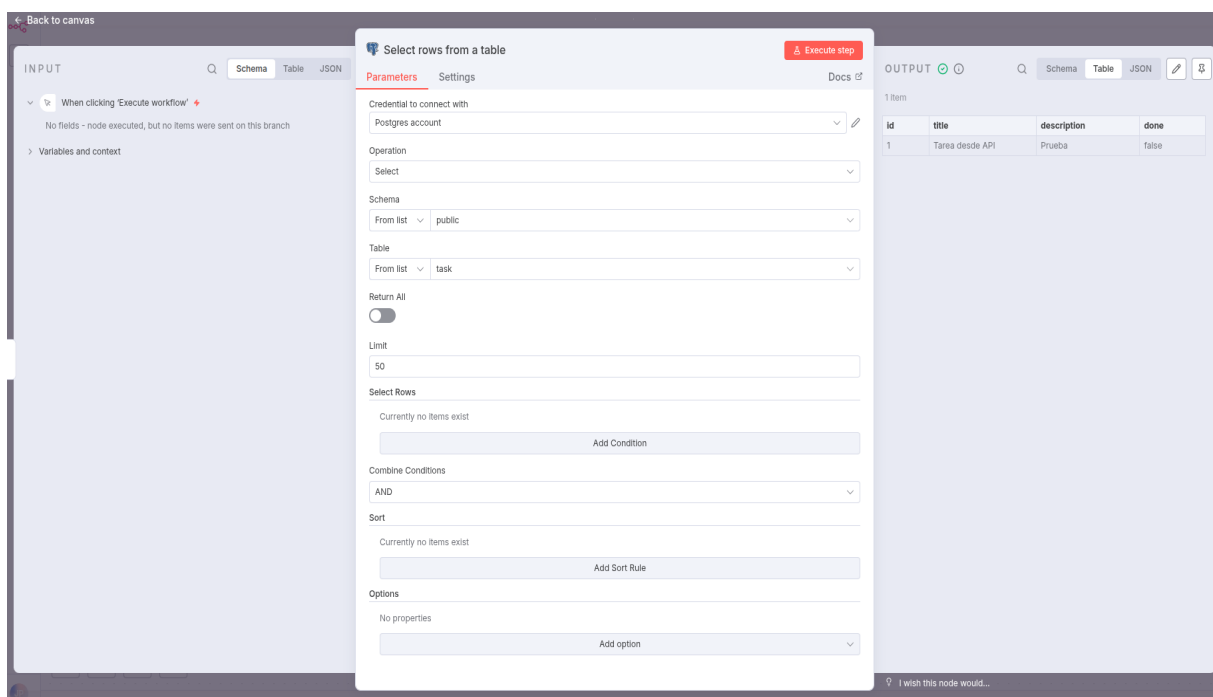
Ignore SSL Issues (Insecure)

☐

La segunda credencial es RabbitMQ donde se le asignará el host, user, password y port.



El siguiente paso es crear un flujo que lee todas las tareas de la base de datos. Para ello, se añade un nodo Manual Trigger y a continuación un nodo PostgreSQL. Se debe seleccionar la tabla task, la misma que en la práctica 5, y la operación de select.



Además, también se crearán dos flujos para demostrar el patrón asíncrono. El primer flujo será productor, que recibirá una petición web y enviará la tarea a RabbitMQ. El nombre del flujo es Productor de Tareas y comenzará con un nodo Webhook Trigger.

Webhook Listen for test event

Parameters Settings Docs

Webhook URLs

Test URL **Production URL**

POST http://localhost:5678/webhook-test/crear_tarea

HTTP Method
POST

Path
crear_tarea

Authentication
None

Respond
Immediately

If you are sending back a response, add a "Content-Type" response header with the appropriate value to avoid unexpected behavior

Options
No properties
[Add option](#)

Pull in events from Webhook
[Listen for test event](#)

Once you've finished building your workflow, run it without having to click this button by using the production webhook URL. [More info](#)

When will this node trigger my flow? [v](#)

OUTPUT Schema Table JSON v

1 item

headers	params	query	body	webhookUri	executionMode
host : localhost:5678 user-agent : curl/8.5.0 accept : */* content-type : application/json content-length : 73	(empty object)	(empty object)	title : Tarea enviada por Webhook n8n description : A cola RabbitMQ	http://localhost:5678/webhook-test/crear_tarea	test

[I wish this node would...](#) Node executed successfully

A continuación se añade un nodo RabbitMQ para que use la cola task-created y guardar los datos en esta cola.

RabbitMQ Execute step

Parameters Settings Docs

Credential to connect with
RabbitMQ account

Operation
Send a Message to RabbitMQ

Mode
Queue

Queue / Topic
task_created

Send Input Data
☒

Options
No properties
[Add option](#)

Wire me up
This node can only receive input data if you connect it to another node. [Learn more](#)

OUTPUT Schema Table JSON v

1 item

success true

[I wish this node would...](#) Node executed successfully

Para crear el flujo consumidor, llamado Consumidor de Tareas, se pondrá un RabbitMQ Trigger como primer nodo para que escuche la cola task-created.

The screenshot shows the configuration of a 'RabbitMQ Trigger' node. The 'Parameters' tab is active, showing the following settings:

- Credential to connect with:** RabbitMQ account
- Queue / Topic:** task_created
- Options:** No properties

The 'OUTPUT' panel on the right displays the message structure:

fields	properties	content
<code>consumerTag : amq.ctag-cCJ7UTT5MbFXU3KTxOCemQ</code> <code>deliveryTag : 1</code> <code>redelivered : false</code> <code>exchange : empty</code> <code>routingKey : task_created</code>	<code>headers : (empty object)</code>	<code>{}</code>

Por último, se añadirá un nodo Edit Fields(Set) para simular un procesamiento. Se añadirá un campo con el nombre de procesado-por y el valor de n8n-consumidor.

The screenshot shows the configuration of an 'Edit Fields' node. The 'Parameters' tab is active, showing the following settings:

- Mode:** Manual Mapping
- Fields to Set:** procesado_por (String), n8n_consumidor
- Include Other Input Fields:** Disabled
- Options:** No properties

The 'INPUT' panel on the left shows the message structure from the previous node:

- fields:**
 - `consumerTag`: amq.ctag-cCJ7UTT5MbFXU3KTxOCemQ
 - `deliveryTag`: 1
 - `redelivered`: false
 - `exchange`: empty
 - `routingKey`: task_created
- properties:**
 - `headers`: (empty object)
 - `content`: {}

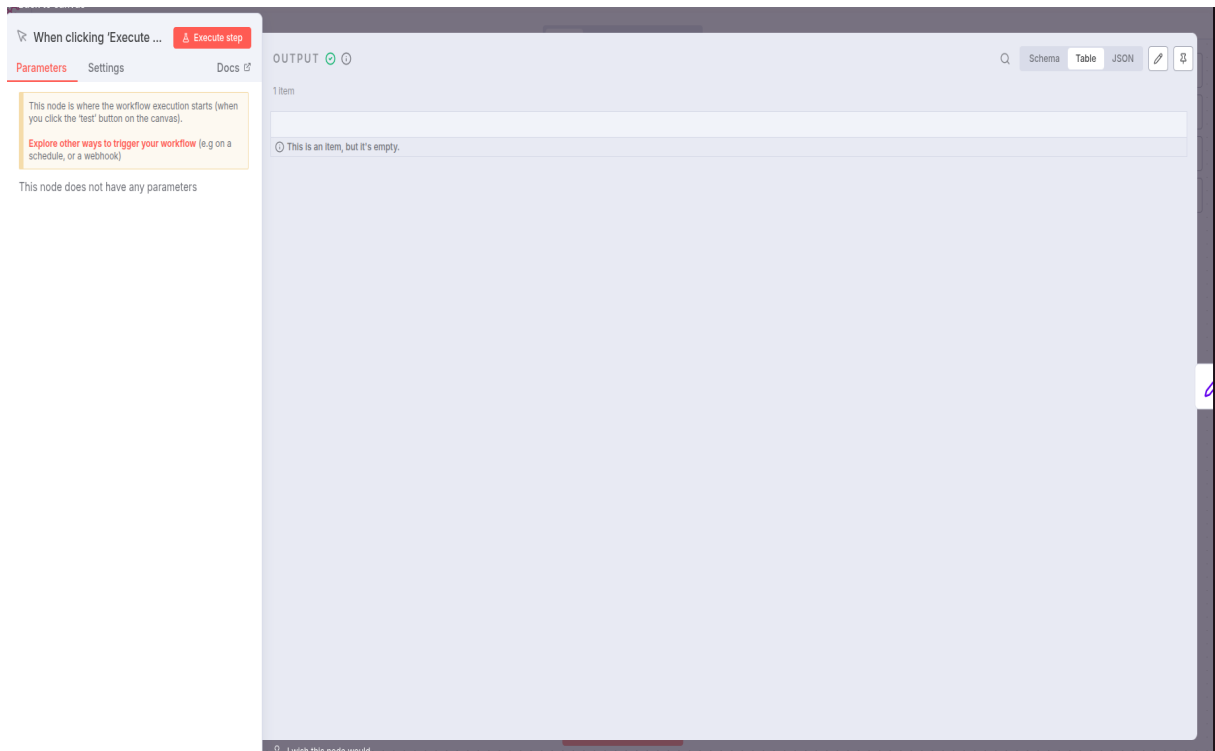
The 'OUTPUT' panel on the right shows the message structure after the edit:

procesado_por
n8n_consumidor

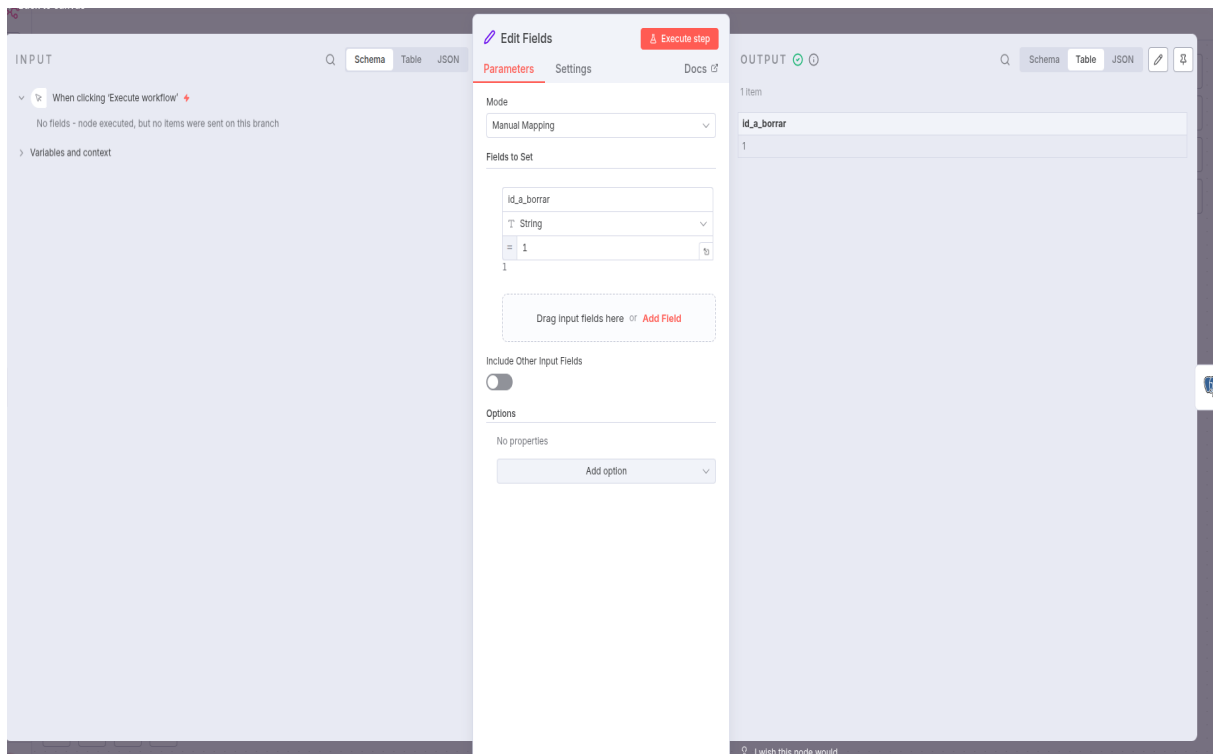
Capítulo 2

Ejercicio 1

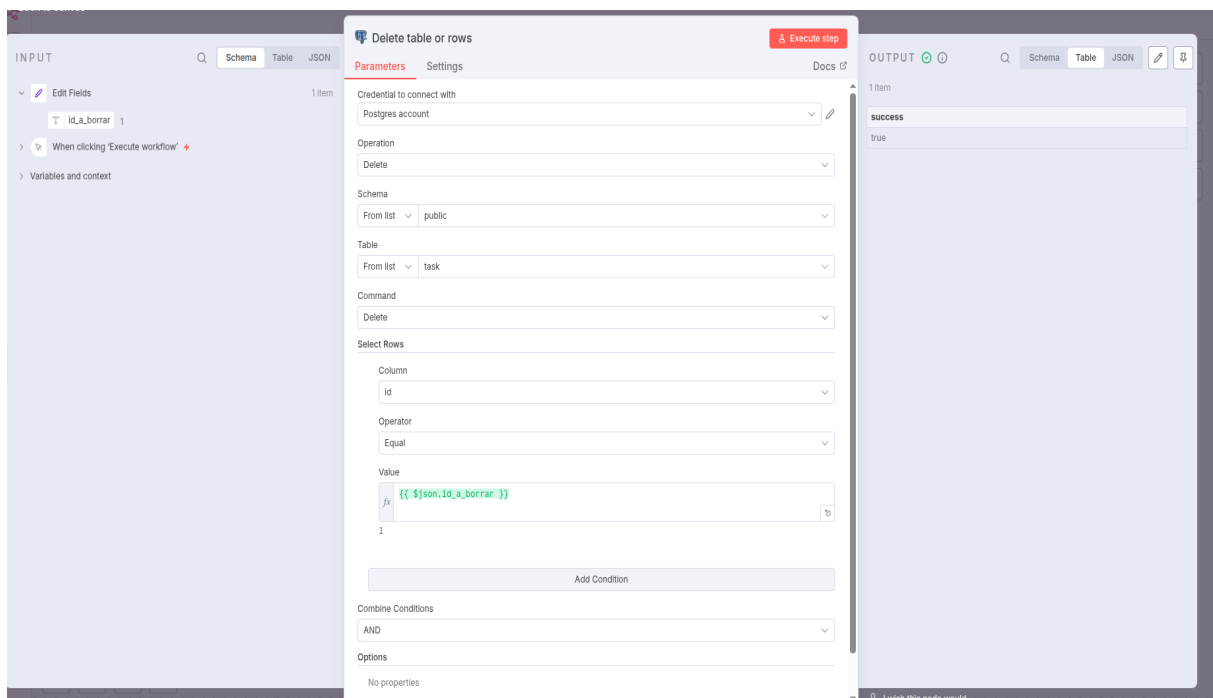
El objetivo del ejercicio 1 es crear un flujo de trabajo que elimine una tarea de la base de datos PostgreSQL. Para conseguir el objetivo lo primero que se debe hacer es añadir un Manual Trigger para dar comienzo al flujo de trabajo.



A continuación, se añade un Edit Fields(Set) para simular que tarea eliminar. Para definir la tarea se usará su ID.



El último paso será añadir un no PostgreSQL con la operación delete. Para eliminar la tarea se configura la condición WHERE para que elimine la fila donde la id sea igual a la id pasada.



Capítulo 3

Ejercicio 2

El objetivo del ejercicio 2 es crear un endpoint para actualizar un tarea para que pase de la cola creada a completada. Para conseguir el objetivo debemos añadir el endpoint de PUT en el archivo app.py.

```
@app.route("/tasks/<int:task_id>/complete", methods=["PUT"])
def complete_task(task_id):
    task = Task.query.get(task_id)

    if not task:
        return jsonify({"error": "task not found"}), 404

    task.done = True
    db.session.commit()

    event = {
        "id": task.id,
        "title": task.title,
        "status": "completed"
    }

    publish_message("task_completed", event)

    return jsonify(task.to_dict()), 200
```

El siguiente paso será crear un flujo de trabajo que comience con un nodo RabbitMQ Trigger para que escuche la cola task-completed.

The screenshot shows the configuration for a 'RabbitMQ Trigger' node. The 'Parameters' tab is active, showing the following settings:

- Credential to connect with:** RabbitMQ account
- Queue / Topic:** task_completed
- Options:** No properties

The 'OUTPUT' tab shows a single item with the following fields:

fields	properties	content
<code>deliveryTag : 1</code> <code>redelivered : false</code> <code>exchange : empty</code> <code>routingKey : task_created</code> <code>messageCount : 3</code>	<code>deliveryMode : 2</code>	<code>{ "id": 5, "title": "Tarea de prueba", "description": null, "done": false }</code>

Cuando reciba un mensaje, el flujo debe usar un nodo Send Email para notificar que la tarea ha sido completada. En el email se pondrá el título y ID de la tarea.

The screenshot shows the configuration for a 'Send email' node. The 'Parameters' tab is active, showing the following settings:

- Credential to connect with:** SMTP account
- Operation:** Send
- From Email:** jpj451@inlumine.ual.es
- To Email:** jpj451@inlumine.ual.es
- Subject:** Tarea completada: {{JSON.parse(\$json.content).title}}
- Email Format:** HTML
- HTML:** La tarea "{{JSON.parse(\$json.content).title}}" (ID: {{JSON.parse(\$json.content).id}}) ha sido marcada como completada.

The 'OUTPUT' tab shows a single item with the following fields:

fields	properties	content
<code>"accepted": [</code> <code> "jpj451@inlumine.ual.es"</code> <code>],</code> <code>"rejected": [</code> <code>],</code> <code>"ehlo": [</code> <code> "SIZE 35882577",</code> <code> "8BITMIME",</code> <code> "AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH",</code> <code> "ENHANCEDSTATUSCODES",</code> <code> "PIPELINING",</code> <code> "CHUNKING",</code> <code> "SMTPUTF8"</code> <code>],</code> <code>"envelopeTime": 276,</code> <code>"messageTime": 504,</code> <code>"messageSize": 711,</code> <code>"response": "250 2.0.0 OK 1764191713 5b1f17b1804b1-4798b0c3a28e64914665e9.9 - gsntp",</code> <code>"envelope": {</code> <code> "from": "jpj451@inlumine.ual.es",</code> <code> "to": [</code> <code> "jpj451@inlumine.ual.es"</code> <code>],</code> <code>"messageId": "<56c2aceb-7528-ff47-7059-cc145ae7ee82@inlumine.ual.es>"</code> <code>}</code>		

Capítulo 4

Ejercicio 3

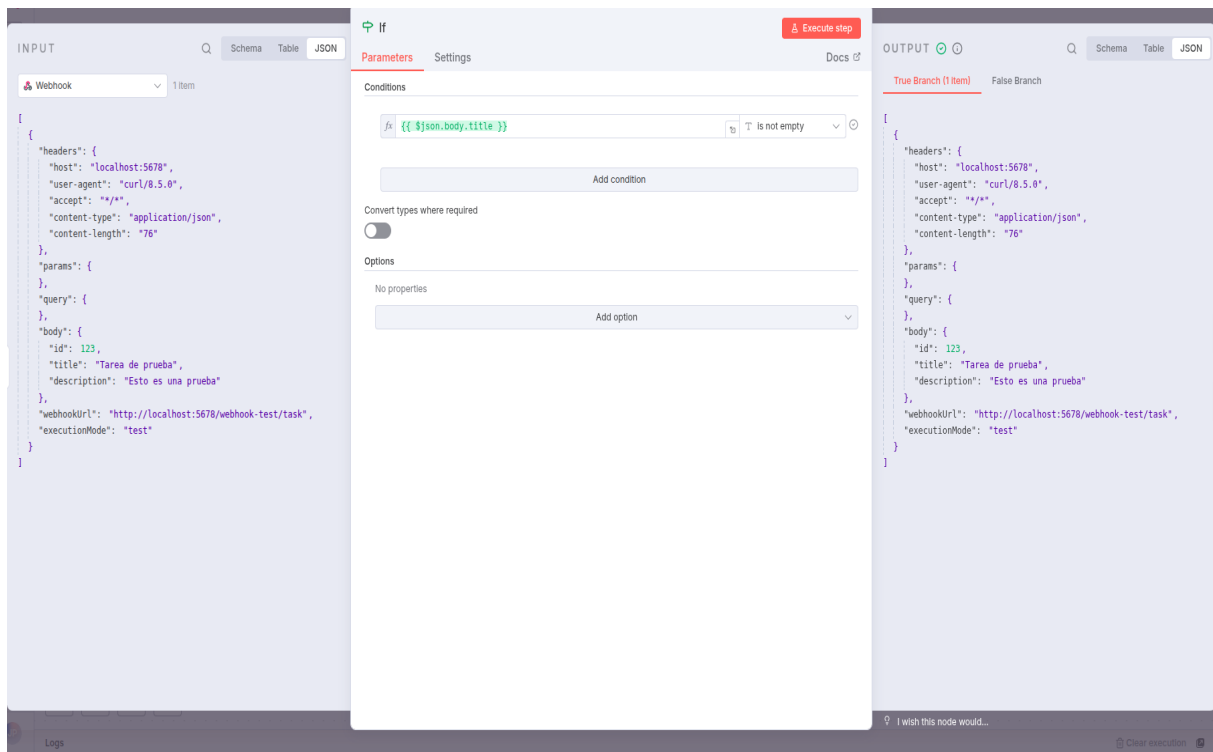
El objetivo del ejercicio 3 es reemplazar completamente la lógica del endpoint POST /tasks de la API de Flask con un flujo de trabajo de n8n. Para ello es necesario crear un nuevo flujo de trabajo que comience con un Webhook Trigger.

The screenshot displays the n8n Webhook node configuration interface. On the left, a sidebar prompts the user to 'Pull in events from Webhook' with a 'Listen for test event' button. The main configuration panel, titled 'Webhook', has tabs for 'Parameters', 'Settings', and 'Docs'. The 'Parameters' tab is active, showing the 'Webhook URLs' section with a 'Test URL' of 'http://localhost:5678/webhook-test/task' and a 'Production URL' field. The 'HTTP Method' is set to 'POST', the 'Path' is 'task', and 'Authentication' is 'None'. The 'Respond' dropdown is set to 'Immediately'. A yellow warning box states: 'If you are sending back a response, add a "Content-Type" response header with the appropriate value to avoid unexpected behavior'. The 'Options' section shows 'No properties' and an 'Add option' button. The right panel, titled 'OUTPUT', displays a JSON response:

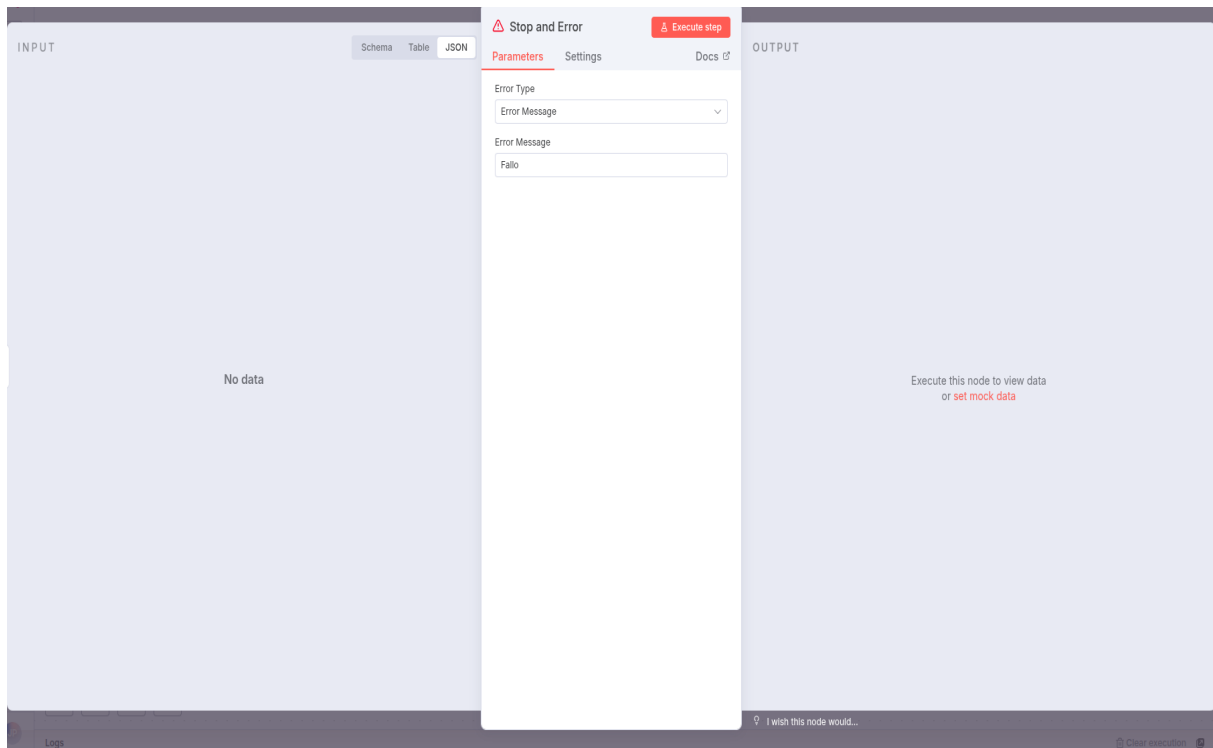
```
{
  "headers": {
    "host": "localhost:5678",
    "user-agent": "curl/8.5.0",
    "accept": "*/",
    "content-type": "application/json",
    "content-length": "76"
  },
  "params": {},
  "query": {},
  "body": {
    "id": 123,
    "title": "Tarea de prueba",
    "description": "Esto es una prueba"
  },
  "webhookUrl": "http://localhost:5678/webhook-test/task",
  "executionNode": "test"
}
```

. The bottom status bar shows 'Logs' and a 'Clear execution' button.

A continuación, se añadirá un nodo IF para validar que el campo de title existe y no esta vacio.



Si el campo esta vacio, se usará un nodo Stop and Error para comunicar el error al usuario.



Si la entrada de datos es correcta se dirigirá a un nodo PostgreSQL para insertar una nueva tarea a la tabla task.

The screenshot displays a workflow editor interface. On the left, the 'INPUT' section shows a JSON object with headers, parameters, and a body containing task details. The central panel is titled 'Insert rows in a table' and is configured with the following settings:

- Credential to connect with:** Postgres account
- Operation:** Insert
- Schema:** From list, public
- Table:** From list, task
- Mapping Column Mode:** Map Each Column Manually
- Values to Send:**
 - title:** Expression: `{{ $json.body.title }}` (Value: Tarea de prueba)
 - description:** Expression: `{{ $json.body.description }}` (Value: Esto es una prueba)
 - done:** Toggle switch (ON)
- Options:** No properties

On the right, the 'OUTPUT' section shows the resulting JSON object:

```
{  "id": 45,  "title": "Tarea de prueba",  "description": "Esto es una prueba",  "done": true}
```

El último paso será publicar los datos de la tarea en la cola de task-created de Rabbit. Para ello, se usará un nodo RabbitMQ.

The screenshot displays a workflow editor interface. On the left, the 'INPUT' section shows a JSON object with task details and a 'done' status. The central panel is titled 'RabbitMQ' and is configured with the following settings:

- Credential to connect with:** RabbitMQ account
- Operation:** Send a Message to RabbitMQ
- Mode:** Queue
- Queue / Topic:** task_created
- Send Input Data:** Expression: `{{ $JSON.stringify($json) }}` (Value: `{ "id": 45, "title": "Tarea de prueba", "description": "Esto es una prueba", "done": true }`)
- Options:** No properties

On the right, the 'OUTPUT' section shows the resulting JSON object:

```
{  "success": true}
```