

Universidad de Las Américas
Facultad de Ingenierías y Ciencias Agropecuarias
Ingeniería De Software
Progreso 3

Nombres: Enrique Merizalde, Jossue Játiva, Doménica Escobar y Juan Aristizábal.

Fecha: 02/07/2025.

Taller Pub/Sub con APIs y RabbitMQ

1. Objetivo General

Desarrollar una solución distribuida basada en el patrón Publicador/Suscriptor (Pub/Sub), utilizando RabbitMQ como middleware de mensajería, donde las aplicaciones que publican y consumen mensajes están expuestas como servicios REST independientes.

2. Contexto del Caso

La empresa StreamEdu está implementando un sistema de procesamiento de tareas estudiantiles. Cada vez que un estudiante sube una tarea, se deben ejecutar varias acciones en paralelo:

- Notificación al profesor
- Análisis de plagio
- Generación de resumen automático del documento
- Registro del evento en un log audit Trail

La empresa ha decidido utilizar RabbitMQ para desacoplar estos procesos y mejorar la escalabilidad.

Actividad

Parte 1: Preparación

- Levantar RabbitMQ usando Docker o instalarlo localmente

- Crear un Exchange tipo fanout llamado “entregas.tareas”

Parte 2: Publicador

Desarrollar una API REST (Spring Boot, Flask, FastAPI, etc.) que:

- Exponga un endpoint POST “/subir-tarea”
 - Simule la subida de una tarea (no es necesario guardar los archivos)
 - Publique un mensaje JSON a “entregas.tareas” con los datos
- ```
{
 "estudiante": "nombre_estudiante",
 "curso": "nombre_curso",
 "archivo": "nombre_archivo",
 "fechaEnvio": "fecha_envio_tarea"
}
```

## Parte 3: Suscriptores (mínimo 2)

Desarrollar al menos dos servicios independientes (REST o consola) que se conecten a RabbitMQ y estén suscritos al exchange “entregas.tareas”

Ejemplos:

- Servicio de notificaciones: recibe el mensaje y simula envío de notificación.
- Servicio de análisis de plagio: simula una validación.
- Servicio de logs: guarda en consola o archivo.
- Servicio de resumen: simula generación de resumen.

Cada servicio debe:

- Conectarse a una queue exclusiva y anónima o nominal.
- Loguear la recepción del mensaje.

## Entregables:

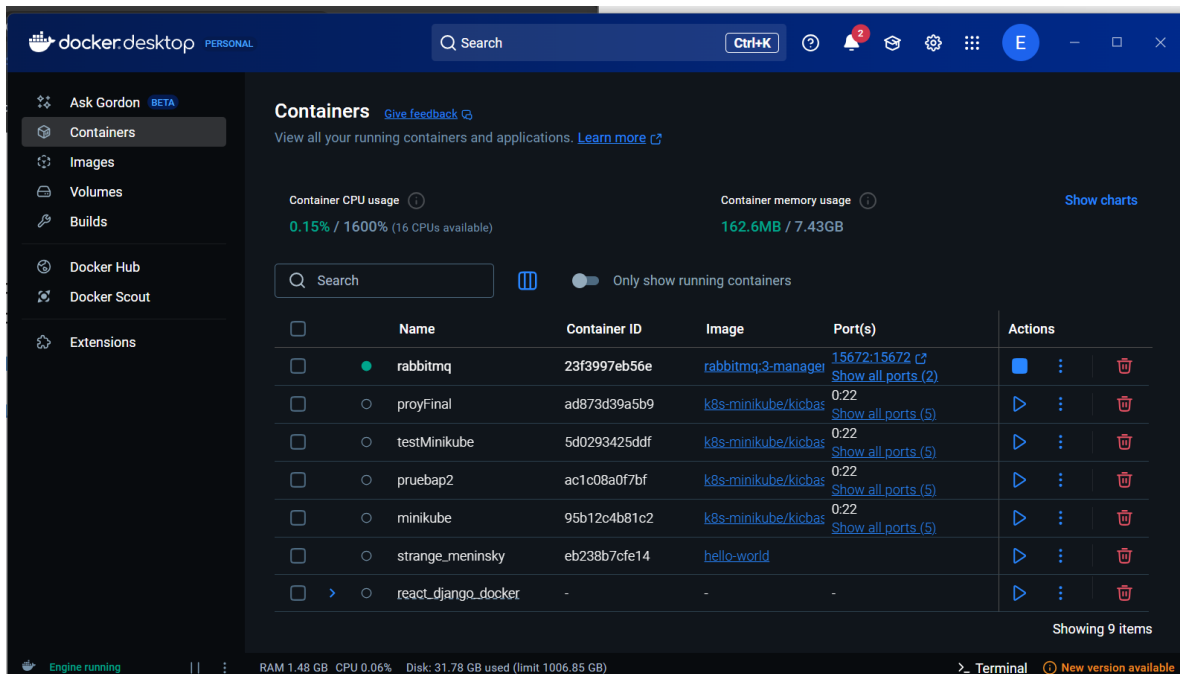
1. Link a repositorio de Código fuente del publicador y los suscriptores.
2. Evidencia (capturas o video corto) del sistema en funcionamiento:
  - Publicador enviando mensaje.
  - Suscriptores recibiendo y procesando el mensaje.
3. Diagrama de arquitectura funcional.
4. Instrucciones para ejecutar los componentes (README).

## Ejecución de la actividad:

1. Link al repositorio: <https://github.com/CHACHO617/PubSubApi>

## 2. Funcionamiento:

- a. Se ha levantado RabbitMQ utilizando un contenedor Docker.



- b. Desarrollo de una API REST con Flask que actúa como un punto de entrada para la actividad de subir tareas.

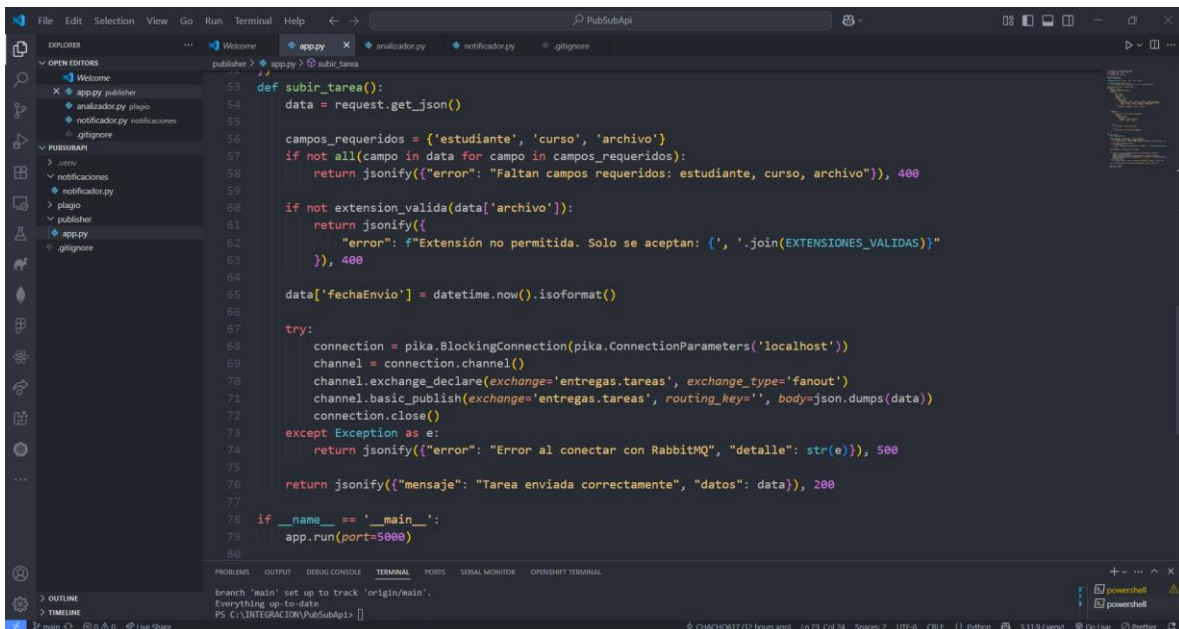
Este API inicia un servidor en el puerto 5000, exponiendo el endpoint “POST /subir-tarea”. Este recibe un JSON con la siguiente estructura

```
{
 "estudiante": "nombre_estudiante",
 "curso": "nombre_curso",
 "archivo": "nombre_archivo",
 "fechaEnvio": "fecha_envio_tarea"
}
```

Este endpoint valida que:

- Se incluyan todos los campos (estudiante, curso y archivo) tomando en cuenta que la fecha de envío se incluye automáticamente.
- El archivo tenga una extensión valida que sea .docx, .txt, .csv o .pdf

Tras estas validaciones el sistema publica el mensaje como JSON en el exchange “entrega.tareas” tipo fanout en RabbitMQ devolviendo un mensaje de respuesta de éxito u error. Además se utiliza Swagger (Flasgger) para generar automáticamente la documentación interactiva del api en localhost:5000/apidocs



```
def subir_tarea():
 data = request.get_json()

 campos_requeridos = {'estudiante', 'curso', 'archivo'}
 if not all(campo in data for campo in campos_requeridos):
 return jsonify({"error": "Faltan campos requeridos: estudiante, curso, archivo"}), 400

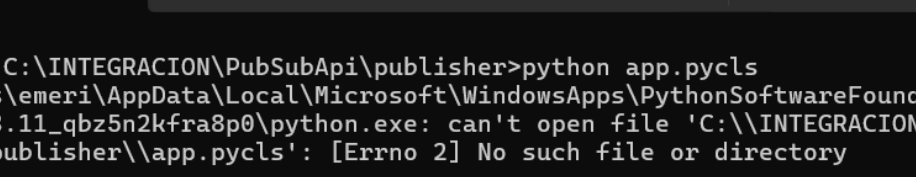
 if not extension_valida(data['archivo']):
 return jsonify({"error": f"Extensión no permitida. Solo se aceptan: {', '.join(EXTENSIONES_VALIDAS)}"}, 400)

 data['fechaEnvio'] = datetime.now().isoformat()

 try:
 connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
 channel = connection.channel()
 channel.exchange_declare(exchange='entregas.tareas', exchange_type='fanout')
 channel.basic_publish(exchange='entregas.tareas', routing_key='', body=json.dumps(data))
 connection.close()
 except Exception as e:
 return jsonify({"error": "Error al conectar con RabbitMQ", "detalle": str(e)}), 500

 return jsonify({"mensaje": "Tarea enviada correctamente", "datos": data}), 200

if __name__ == '__main__':
 app.run(port=5000)
```



The screenshot shows a Windows PowerShell terminal window with the following content:

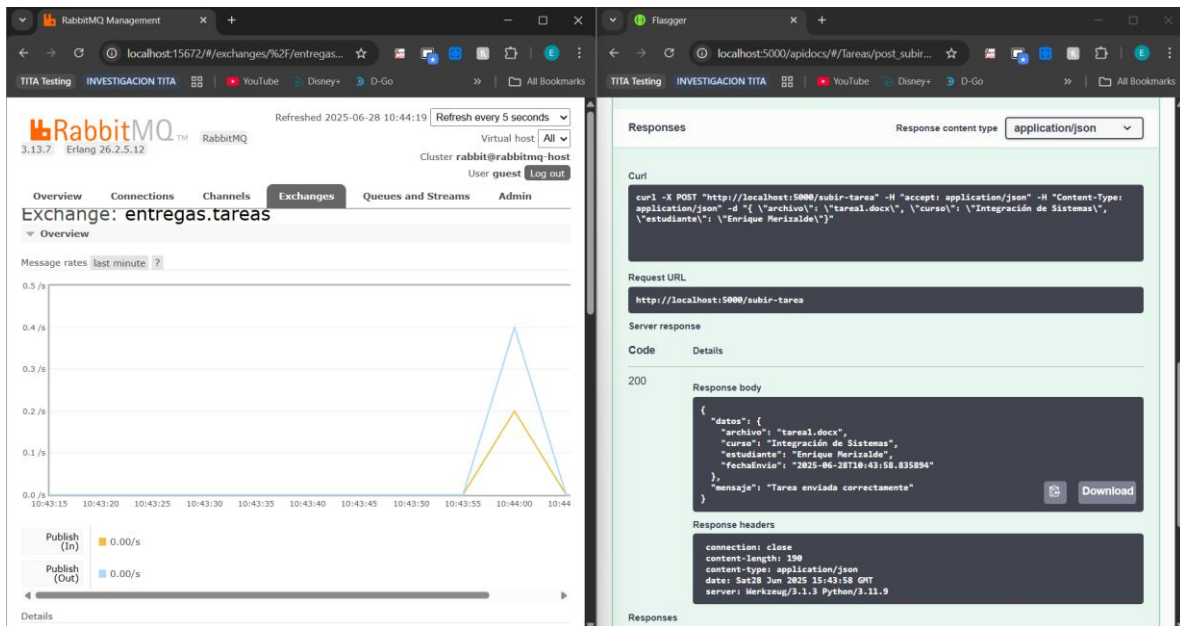
```
(.venv) C:\INTEGRACION\PubSubApi\publisher>python app.pycls
C:\Users\emeri\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.
Python.3.11_qbz5n2kfra8p0\python.exe: can't open file 'C:\\INTEGRACION\\PubS
ubApi\\publisher\\app.pycls': [Errno 2] No such file or directory

(.venv) C:\INTEGRACION\PubSubApi\publisher>python app.py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deploy
ment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

The screenshot shows the RabbitMQ Management web interface at localhost:15672/#/exchanges. The page title is "RabbitMQ Management". The breadcrumb navigation shows "Overview > Connections > Channels > Exchanges". The main heading is "All exchanges (11)". Below this, there's a pagination section showing "Page 1 of 1 - Filter:" and "Displaying 11 items , page size up to: 100".

| Virtual host | Name               | Type    | Features | Message rate in | Message rate out | +/- |
|--------------|--------------------|---------|----------|-----------------|------------------|-----|
| /            | (AMQP default)     | direct  | D        |                 |                  |     |
| /            | alert-exchange     | fanout  | D        |                 |                  |     |
| /            | alert-exchanger    | fanout  | D        |                 |                  |     |
| /            | amq.direct         | direct  | D        |                 |                  |     |
| /            | amq.fanout         | fanout  | D        |                 |                  |     |
| /            | amq.headers        | headers | D        |                 |                  |     |
| /            | amq.match          | headers | D        |                 |                  |     |
| /            | amq.rabbitmq.trace | topic   | D I      |                 |                  |     |
| /            | amq.topic          | topic   | D        |                 |                  |     |
| /            | email-exchange     | fanout  | D        |                 |                  |     |
| /            | entregas.tareas    | fanout  |          | 0.00/s          | 0.00/s           |     |

At the bottom of the table, there is a link: "Add a new exchange".



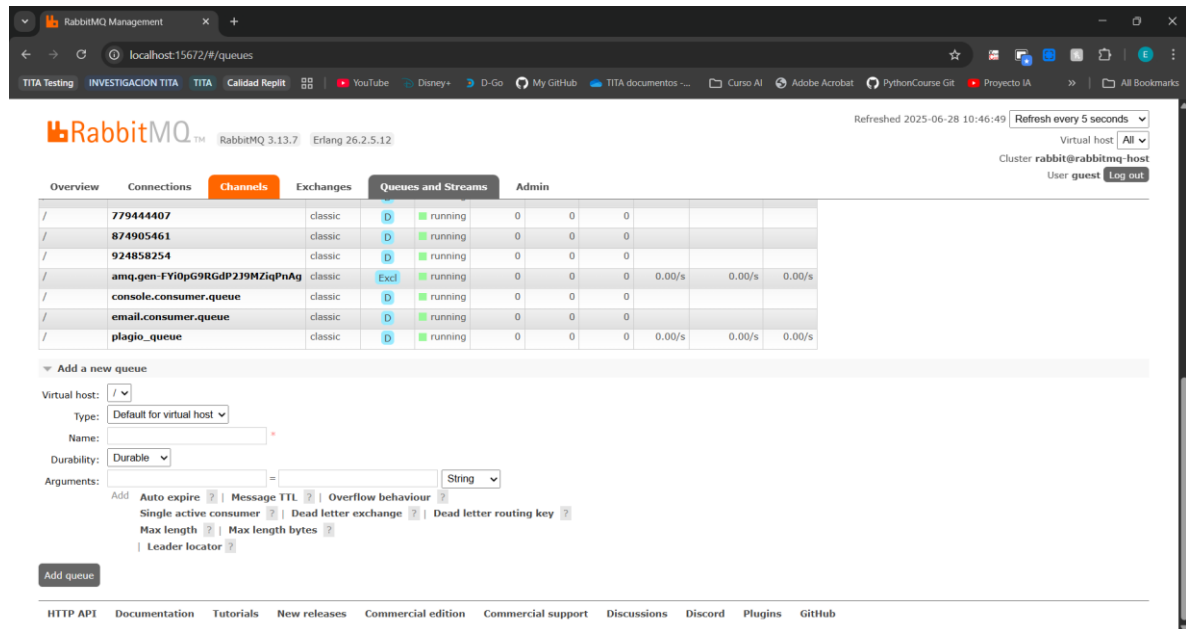
```
(.venv) C:\INTEGRACION\PubSubApi\publisher>python app.pyc
C:\Users\emeri\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\python.exe: can't open file 'C:\\INTEGRACION\\PubSubApi\\publisher\\app.pyc': [Errno 2] No such file or directory

(.venv) C:\INTEGRACION\PubSubApi\publisher>python app.py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [28/Jun/2025 10:42:41] "POST /subir-tarea HTTP/1.1" 200 -
127.0.0.1 - - [28/Jun/2025 10:43:58] "POST /subir-tarea HTTP/1.1" 200 -
```

c. Desarrollo del primer suscriptor en este caso que simula un servicio de análisis de plagio. Este API igualmente fue desarrollado utilizando Flask y este se conecta RabbitMQ que está corriendo en localhost.

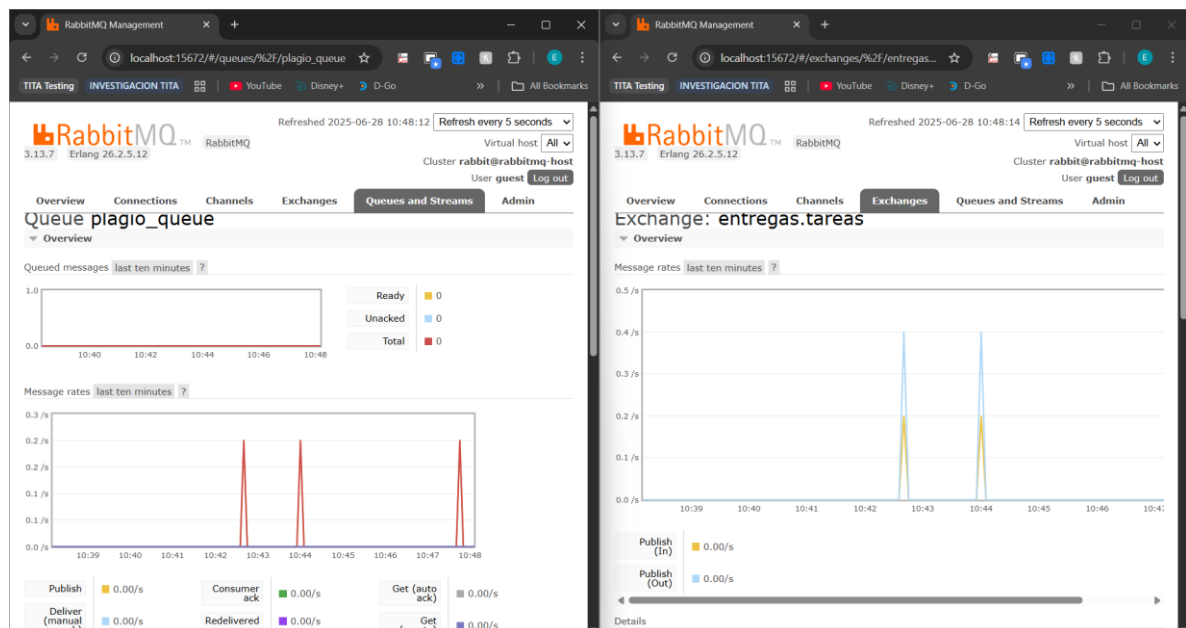
- En primera instancia se valida que el exchange “entregas.tareas” exista y sea de tipo fanout.
- Luego se declara y se conecta a una cola llamada “plagio\_queue”, la cual es persistente es decir que se mantiene si el servicio se detiene.
- Se suscribe a esta cola y se queda escuchando por mensajes.

- Cada vez que recibe un mensaje además ve mostrarlo en la interfaz de RabbitMQ lo convierte desde JSON a un diccionario de Python y muestra un mensaje en consola indicando: [Plagio] Analizando archivo “nombre\_archivo” enviado por “nombre\_estudiante”



RabbitMQ Management interface showing the 'Queues and Streams' tab. The table lists several queues, including 'plagio\_queue', which is in a 'running' state. The interface also shows options to add a new queue and various links for documentation and support.

| Queue Name | Message Count                 | Unacked Count | Ready Count | State   |
|------------|-------------------------------|---------------|-------------|---------|
| /          | 77944407                      | 0             | 0           | running |
| /          | 874905461                     | 0             | 0           | running |
| /          | 924858254                     | 0             | 0           | running |
| /          | amq.gen-FY0pG9RGdP2J9MZigPnAg | 0             | 0           | running |
| /          | console.consumer.queue        | 0             | 0           | running |
| /          | email.consumer.queue          | 0             | 0           | running |
| /          | plagio_queue                  | 0             | 0           | running |



Two side-by-side screenshots of the RabbitMQ Management interface. The left screenshot shows the 'Queue: plagio\_queue' overview with message rates and status. The right screenshot shows the 'Exchange: entregas.tareas' overview with message rates and status.

**Queue: plagio\_queue Overview**

Queued messages last ten minutes: 0.0

Message rates last ten minutes: 0.0/s

Ready: 0, Unacked: 0, Total: 0

Consumer ack: 0.00/s, Redelivered: 0.00/s, Get (auto ack): 0.00/s, Get (empty): 0.00/s

**Exchange: entregas.tareas Overview**

Message rates last ten minutes: 0.0/s

Publish (In): 0.00/s, Publish (Out): 0.00/s

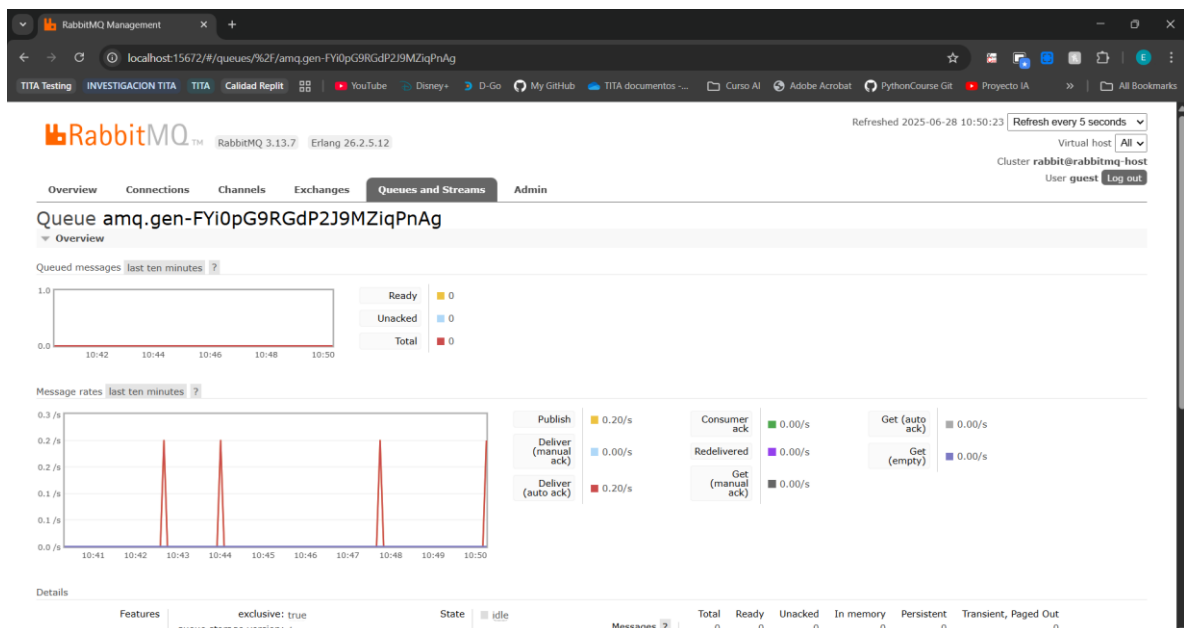
```

Command Promp x Windows PowerS x Windows PowerS x + v - □ x
PS C:\INTEGRACION\PubSubApi\plagio> python analizador.py
[*] Servicio de Análisis de Plagio esperando mensajes...
[Plagio] Analizando archivo 'tarea1.docx' enviado por Enrique Merizalde...
[Plagio] Analizando archivo 'tarea1.docx' enviado por Enrique Merizalde...
[Plagio] Analizando archivo 'tarea1.docx' enviado por Enrique Merizalde...

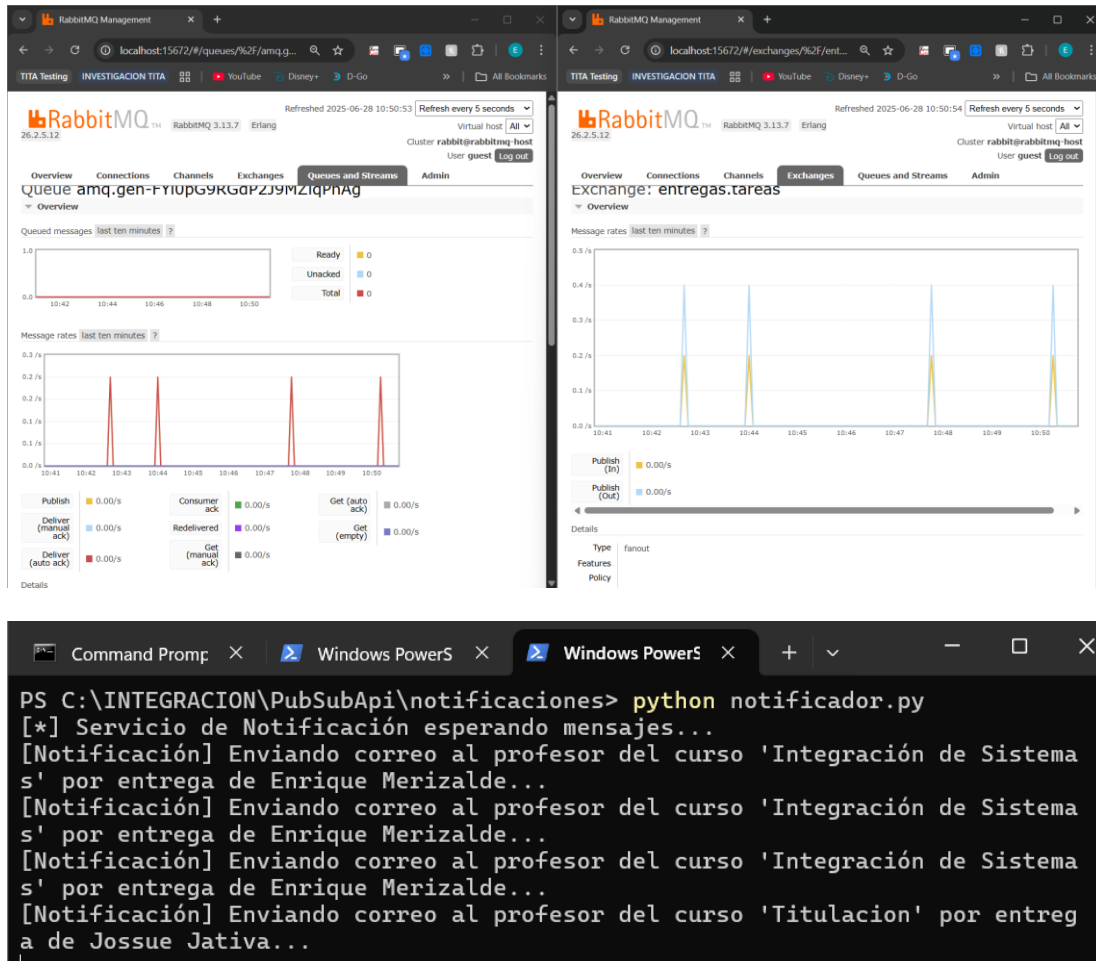
```

d. Desarrollo del segundo suscriptor, en este caso que simula un servicio de notificaciones a profesor. Este API igualmente fue desarrollado utilizando Flask y se conecta a RabbitMQ que está corriendo en el localhost.

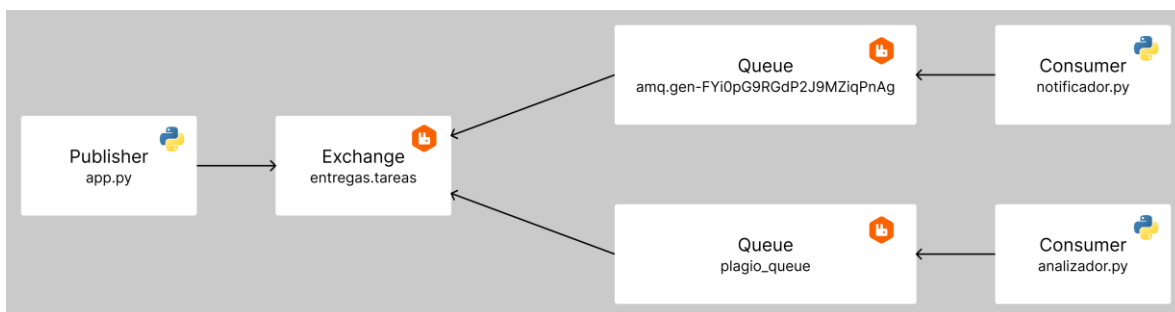
- Este declara el exchange entregas.tareas si aún no existe
- Declara una cola anónima y exclusiva, es decir que se genera automáticamente y desaparece al cerrar el programa. En este caso es ideal ya que no se necesita conservar el estado
- Se suscribe a la cola creada
- Cada vez que recibe un mensaje lo convierte desde JSON a un mensaje y muestra: [Notificación] Enviando correo al profesor del curso “nombre\_curso” por entrega de “nombre\_estudiante”







### 3. Diagrama de Arquitectura funcional



### 4. Link a Readme.md:

<https://github.com/CHACHO617/PubSubApi/tree/main#readme>