

Memoria Práctica 5

Jaime Parra Jiménez

INTEGRACIÓN DE TECNOLOGÍAS Y SERVICIOS INFORMÁTICOS

Universidad de Almería

Correo: jj451@inlumine.ual.es

19 de noviembre de 2025

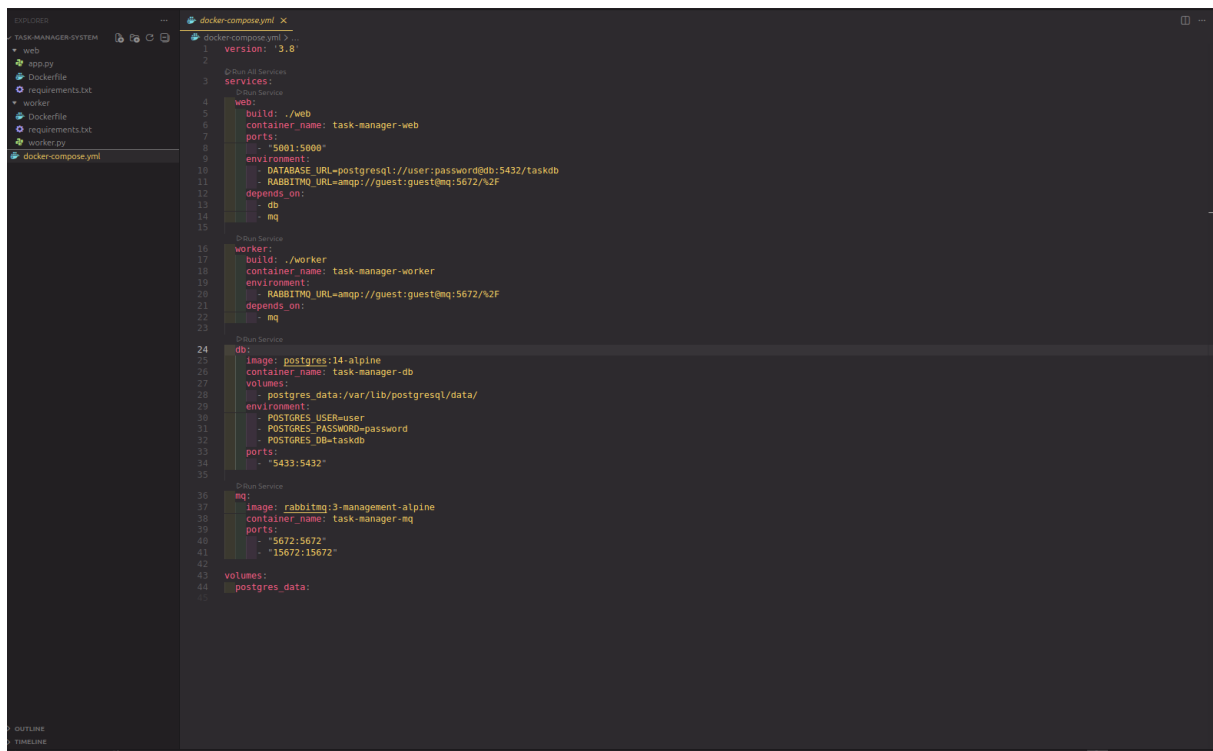
Índice general

1. Ejercicio Guiado 1	2
2. Ejercicio 1	8
3. Ejercicio 2	10
4. Ejercicio 3	14

Capítulo 1

Ejercicio Guiado 1

El objetivo del ejercicio guiado es transformar un microservicio único en una aplicación de tres componentes: una API web, una base de datos PostgreSQL y un trabajador asíncrono, todos orquestados por Docker Compose. Para ello será necesario crear varios ficheros. Primero, se crea el fichero `docker-compose.yml` en la raíz de nuestro proyecto. Este fichero será el centro de control de nuestra aplicación.



```
1 version: '3.8'
2
3 services:
4   web:
5     build: ./web
6     container_name: task-manager-web
7     ports:
8       - "5001:5000"
9     environment:
10      - DATABASE_URL=postgresql://user:password@db:5432/taskdb
11      - RABBITMQ_URL=amqp://guest:guest@mq:5672/?2F
12     depends_on:
13       - db
14       - mq
15
16   worker:
17     build: ./worker
18     container_name: task-manager-worker
19     environment:
20      - RABBITMQ_URL=amqp://guest:guest@mq:5672/?2F
21     depends_on:
22       - mq
23
24   db:
25     image: postgres:14-alpine
26     container_name: task-manager-db
27     volumes:
28       - postgres_data:/var/lib/postgresql/data/
29     environment:
30      - POSTGRES_USER=user
31      - POSTGRES_PASSWORD=password
32      - POSTGRES_DB=taskdb
33     ports:
34       - "5433:5432"
35
36   mq:
37     image: rabbitmq:3-management-alpine
38     container_name: task-manager-mq
39     ports:
40       - "5672:5672"
41       - "15672:15672"
42
43 volumes:
44   postgres_data:
```

Una vez creado el fichero, el siguiente paso es crear una carpeta llamada web. Dentro de ella tendremos los ficheros requirements.txt, Dockerfile y app.py. Primero configuraremos app.py, que es el archivo principal de la aplicación.

```

1  # Configuración de la Base de Datos ---
2  app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL')
3  app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
4  db = SQLAlchemy(app)
5
6  # Modelo de la Base de Datos ---
7  class Task(db.Model):
8      id = db.Column(db.Integer, primary_key=True)
9      title = db.Column(db.String(120), nullable=False)
10     description = db.Column(db.String(255), nullable=True)
11     done = db.Column(db.Boolean, default=False)
12
13     def to_dict(self):
14         return {
15             'id': self.id,
16             'title': self.title,
17             'description': self.description,
18             'done': self.done
19         }
20
21 # Configuración de RabbitMQ
22 RABBITMQ_URL = os.environ.get('RABBITMQ_URL')
23
24 def publish_message(queue_name, message):
25     try:
26         connection = pika.BlockingConnection(pika.URLParameters(RABBITMQ_URL))
27         channel = connection.channel()
28         channel.queue_declare(queue=queue_name, durable=True)
29         channel.basic_publish(
30             exchange='',
31             routing_key=queue_name,
32             body=json.dumps(message),
33             properties=pika.BasicProperties(delivery_mode=2) # persistente
34         )
35         connection.close()
36         print(f"[x] Sent message to queue '{queue_name}'")
37     except Exception as e:
38         print(f"Error publishing message: {e}")
39
40 # Endpoints de la API ---
41 @app.route('/tasks', methods=[GET])
42 def get_tasks():
43     tasks = Task.query.all()
44     return jsonify([task.to_dict() for task in tasks])
45
46 # POST /tasks - crear una nueva tarea
47 @app.route('/tasks', methods=[POST])
48 def create_task():
49     if not request.json or 'title' not in request.json:
50         return jsonify({'error': 'Bad request: title is required'}), 400
51
52     new_task = Task(
53         title=request.json['title'],
54         description=request.json.get('description', '')
55     )
56     db.session.add(new_task)
57     db.session.commit()
58
59     # Publicar mensaje a RabbitMQ
60     publish_message('task_created', new_task.to_dict())
61
62     return jsonify({'task': new_task.to_dict()}), 201
63
64 #... puedes añadir mas endpoints como GET /tasks/<id>, DELETE, PUT, etc.
65
66 if __name__ == '__main__':
67     with app.app_context():
68         db.create_all()
69     app.run(host='0.0.0.0', port=5000, debug=True)

```

A continuación, creamos el Dockerfile que es el encargado de crear la imagen de Docker del contenedor.

```

1 FROM python:3.9-slim-buster (last pushed 2 years ago)
2 WORKDIR /app
3
4 # Copy requirements and install
5 COPY requirements.txt .
6 RUN pip install --no-cache-dir -r requirements.txt
7
8 # Copy app code
9 COPY . .
10
11 # Expose port and start with gunicorn (or flask run for dev)
12 EXPOSE 5000
13
14 CMD ["python", "app.py"]

```

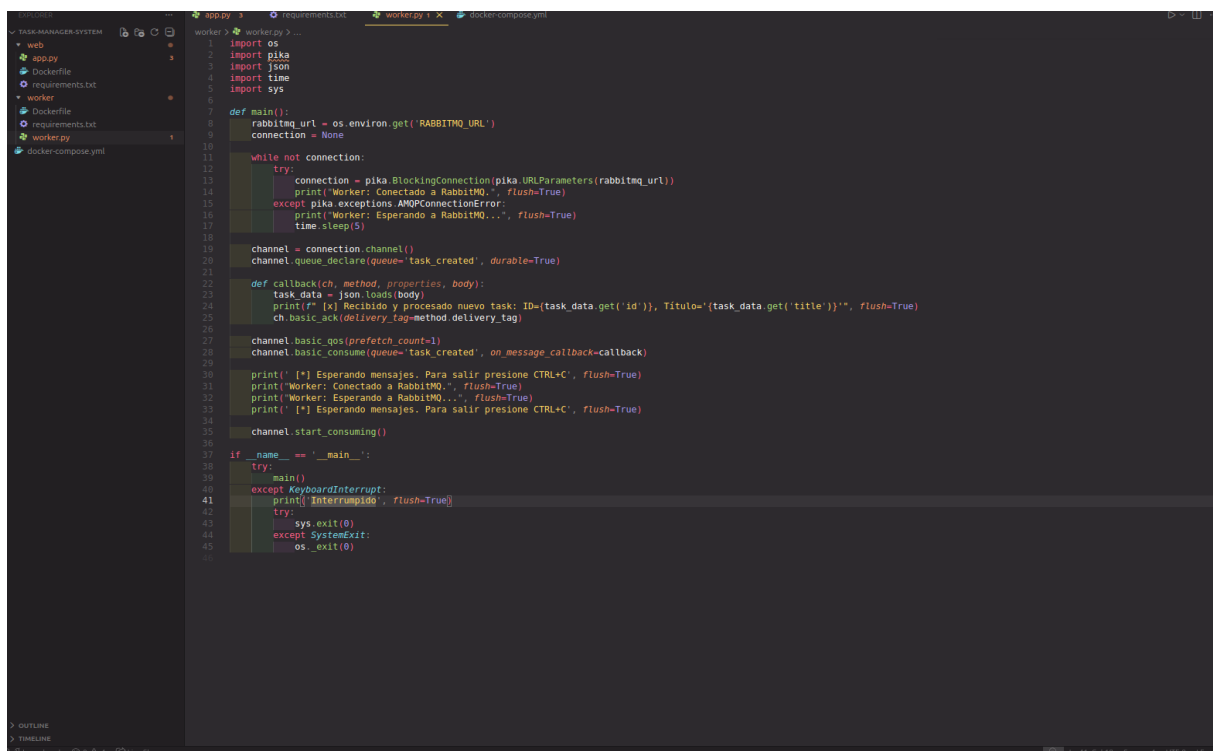
Por último, creamos el archivo requirements.txt donde se definiran las dependencias de Python que se instalaran en el contenedor.



The screenshot shows the VS Code interface with the Explorer sidebar on the left displaying the project structure. The 'web' directory is selected, and the 'requirements.txt' file is open in the editor. The file contains the following dependencies:

```
1 flask==2.2.2
2 Flask-SQLAlchemy==3.0.3
3 psycopg2-binary==2.9.5
4 pika==1.3.1
5 Werkzeug==2.2.2
```

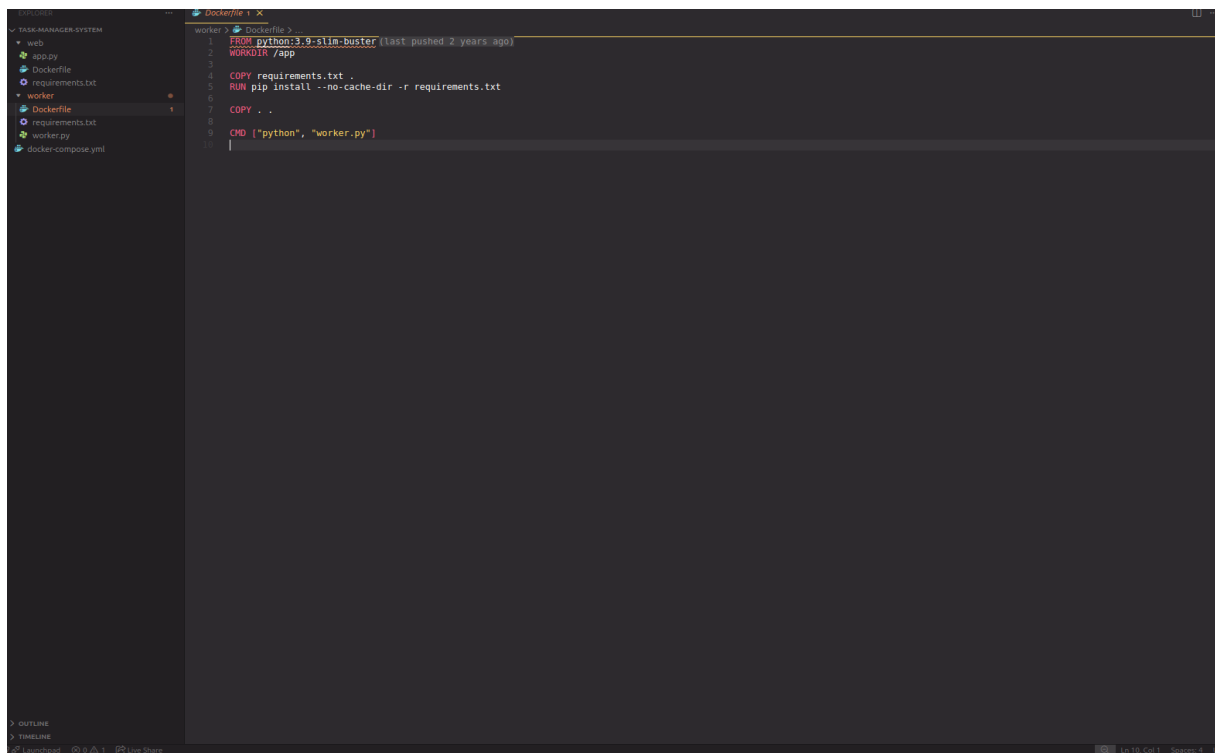
Una vez creada la web es paso de crear el worker, nuestro consumidor de mensajes. Para ello creamos en la raíz del proyecto una carpeta worker, la cual tendrá los mismos ficheros que web. Primero configuraremos worker.py, donde se conectará a RabbitMQ, escuchará mensajes en la cola y los procesará.



The screenshot shows the VS Code interface with the Explorer sidebar on the left displaying the project structure. The 'worker' directory is selected, and the 'worker.py' file is open in the editor. The file contains the following code:

```
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.', flush=True)
15         except pika.exceptions.AMQPConnectionError:
16             print('Worker: Esperando a RabbitMQ...', flush=True)
17             time.sleep(5)
18
19     channel = connection.channel()
20     channel.queue_declare(queue='task_created', durable=True)
21
22     def callback(ch, method, properties, body):
23         task_data = json.loads(body)
24         print(f'[X] Recibido y procesado nuevo task: ID={task_data.get('id')}, Titulo={task_data.get('title')}', flush=True)
25         ch.basic_ack(delivery_tag=method.delivery_tag)
26
27     channel.basic_qos(prefetch_count=1)
28     channel.basic_consume(queue='task_created', on_message_callback=callback)
29
30     print([' Esperando mensajes. Para salir presione CTRL+C', flush=True)
31     print('Worker: Conectado a RabbitMQ.', flush=True)
32     print('Worker: Esperando a RabbitMQ.', flush=True)
33     print([' Esperando mensajes. Para salir presione CTRL+C', flush=True)
34
35     channel.start_consuming()
36
37 if __name__ == '__main__':
38     try:
39         main()
40     except KeyboardInterrupt:
41         print('Interrumpido', flush=True)
42         try:
43             sys.exit(0)
44         except SystemExit:
45             os._exit(0)
```

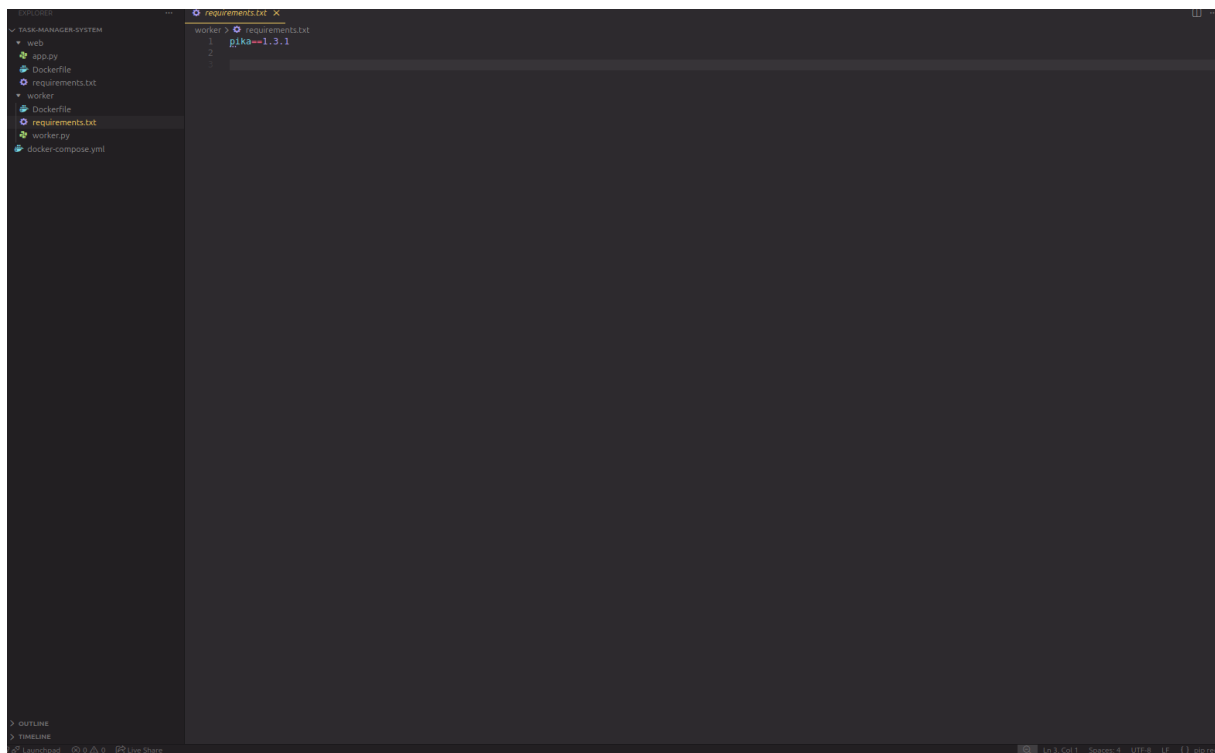
El dockerfile será igual que el de web.



The screenshot shows a VS Code editor with a file explorer on the left containing files: web, app.py, Dockerfile, requirements.txt, worker, Dockerfile, requirements.txt, worker.py, and docker-compose.yml. The main editor window is open to a file named 'Dockerfile' in the 'worker' directory. The code in the Dockerfile is as follows:

```
1 FROM python:3.9-slim-buster (last pushed 2 years ago)
2 WORKDIR /app
3
4 COPY requirements.txt .
5 RUN pip install --no-cache-dir -r requirements.txt
6
7 COPY . .
8
9 CMD ["python", "worker.py"]
```

Y por último, configuramos requirements.txt para poner las dependencias.



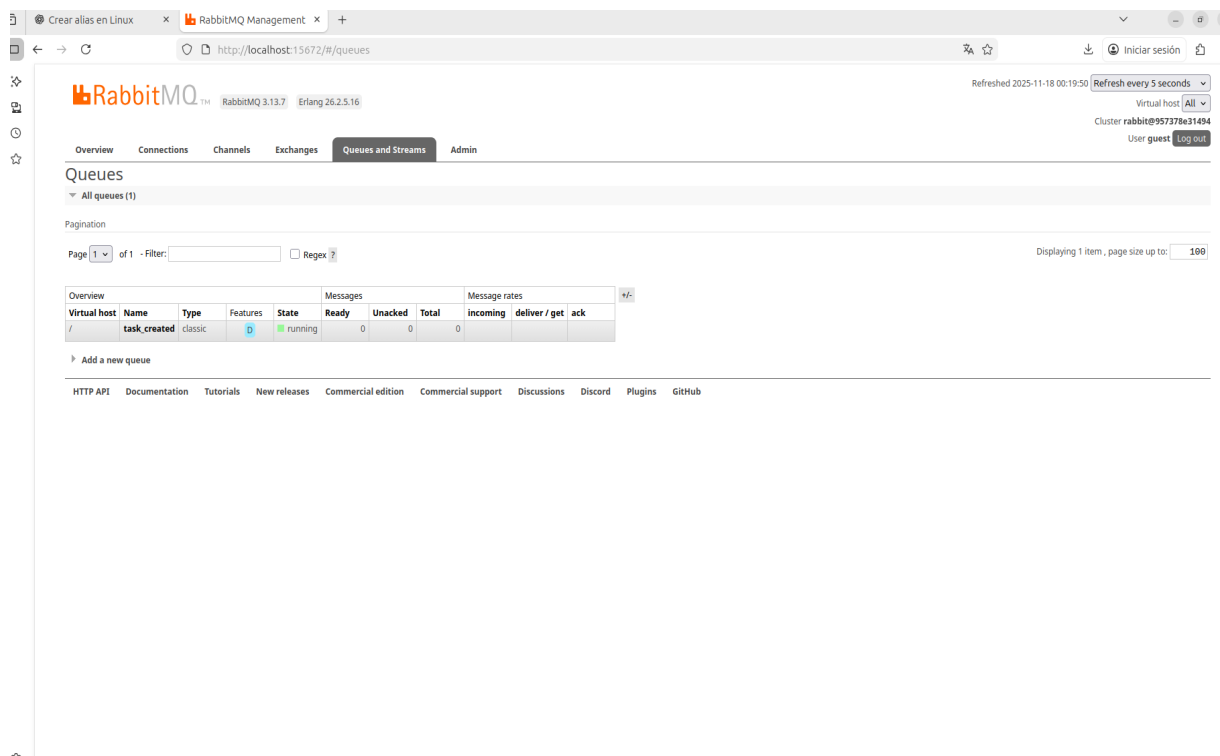
The screenshot shows the same VS Code editor with the file explorer on the left. The main editor window is now open to a file named 'requirements.txt' in the 'worker' directory. The code in the file is:

```
1 pika==1.3.1
```

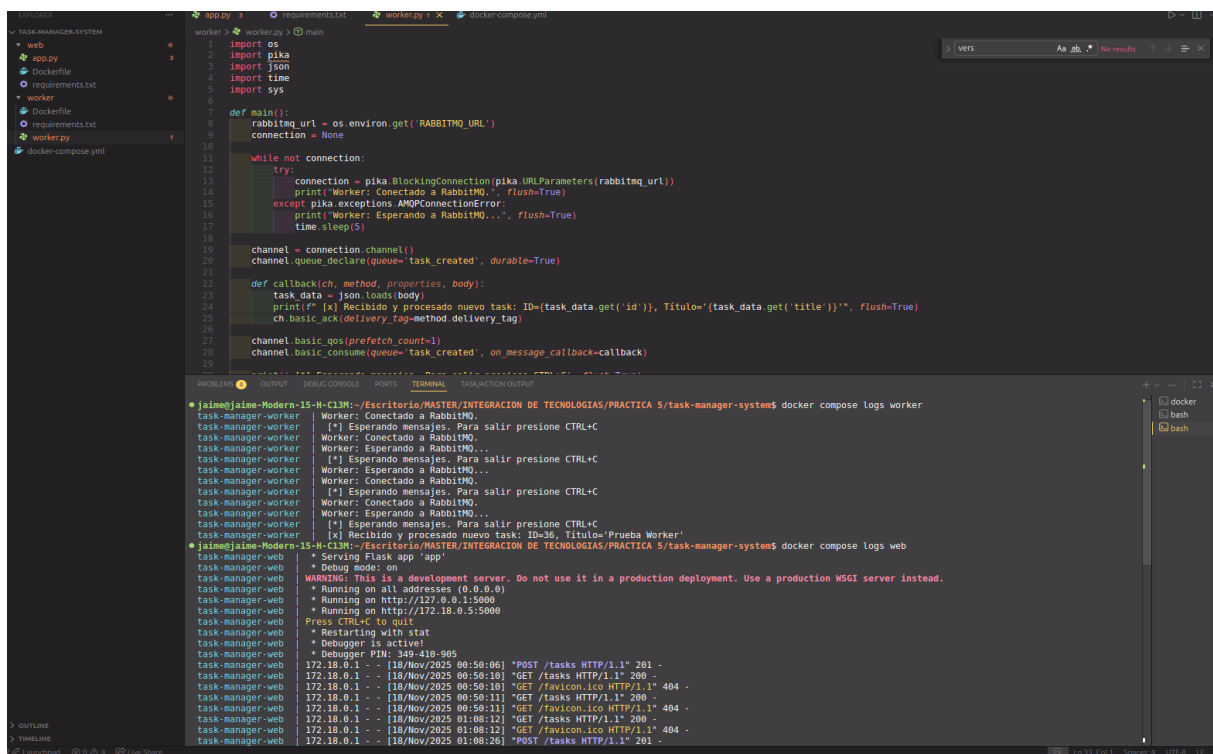
Ya configurados todos los archivos es hora de levantar la aplicación. Para ello debemos poner el siguiente comando docker-compose up --build en la terminal desde la raíz del proyecto.

```
jaime@jaime-Modern-15-H-C13M:~/Escritorio/MASTER/INTEGRACION DE TECNOLOGIAS/PRACTICA 5/task-manager-system$ docker-compose up --build
task-manager-mq | 2025-11-17 23:17:19.442870+00:00 [warning] <0.733.0> Its use will not be permitted by default in a future minor RabbitMQ version and the feature will be removed from a future major RabbitMQ version; actual versions to be determined.
task-manager-mq | 2025-11-17 23:17:19.442870+00:00 [warning] <0.733.0> To continue using this feature when it is not permitted by default, set the following parameter in your configuration:
task-manager-mq | 2025-11-17 23:17:19.442870+00:00 [warning] <0.733.0> "deprecated_features.permit.management_metrics_collection = true"
task-manager-mq | 2025-11-17 23:17:19.442870+00:00 [warning] <0.733.0> To test RabbitMQ as if the feature was removed, set this in your configuration:
task-manager-mq | 2025-11-17 23:17:19.442870+00:00 [warning] <0.733.0> "deprecated_features.permit.management_metrics_collection = false"
task-manager-mq | 2025-11-17 23:17:20.509493+00:00 [info] <0.770.0> Management plugin: HTTP (non-TLS) listener started on port 15672
task-manager-mq | 2025-11-17 23:17:20.509579+00:00 [info] <0.800.0> Statistics database started.
task-manager-mq | 2025-11-17 23:17:20.509622+00:00 [info] <0.799.0> Starting worker pool 'management_worker_pool' with 3 processes in it
task-manager-mq | 2025-11-17 23:17:20.514925+00:00 [info] <0.818.0> Prometheus metrics: HTTP (non-TLS) listener started on port 15692
task-manager-mq | 2025-11-17 23:17:20.514997+00:00 [info] <0.704.0> Ready to start client connection listeners
task-manager-mq | 2025-11-17 23:17:20.515706+00:00 [info] <0.862.0> started TCP listener on [::]:5672
task-manager-mq | completed with 5 plugins.
task-manager-mq | 2025-11-17 23:17:20.552708+00:00 [info] <0.704.0> Server startup complete; 5 plugins started.
task-manager-mq | 2025-11-17 23:17:20.552708+00:00 [info] <0.704.0> * rabbitmq_prometheus
task-manager-mq | 2025-11-17 23:17:20.552708+00:00 [info] <0.704.0> * rabbitmq_federation
task-manager-mq | 2025-11-17 23:17:20.552708+00:00 [info] <0.704.0> * rabbitmq_management
task-manager-mq | 2025-11-17 23:17:20.552708+00:00 [info] <0.704.0> * rabbitmq_management_agent
task-manager-mq | 2025-11-17 23:17:20.552708+00:00 [info] <0.704.0> * rabbitmq_web_dispatch
task-manager-mq | 2025-11-17 23:17:20.562542+00:00 [info] <0.9.0> Time to start RabbitMQ: 3655 ms
task-manager-mq | 2025-11-17 23:17:21.834064+00:00 [info] <0.867.0> accepting AMQP connection <0.867.0> (172.18.0.4:55094 -> 172.18.0.2:5672)
task-manager-mq | 2025-11-17 23:17:21.837681+00:00 [info] <0.867.0> connection <0.867.0> (172.18.0.4:55094 -> 172.18.0.2:5672): user 'guest' authenticated and granted access to vhost '/'
```

También comprobaremos RabbitMQ desde el navegador para ver la cola creada.



Por último, probaremos el flujo completo haciendo un curl para crear una nueva tarea en la API, si es correcta en el log de worker se podrá ver el mensaje de que se ha recibido la nueva tarea.



The screenshot shows a VS Code editor with a project named 'TASK-MANAGER-SYSTEM'. The file explorer on the left shows files: web, app.py, Dockerfile, requirements.txt, worker.py, and docker-compose.yml. The 'worker.py' file is open in the editor, showing a Python script that connects to a RabbitMQ queue named 'task_created'. The script uses the pika library and includes a callback function to process tasks. The terminal at the bottom shows the output of 'docker compose logs worker', displaying the worker's logs. The logs show the worker connecting to RabbitMQ, waiting for messages, and receiving a task with ID=36 and title='Prueba Worker'.

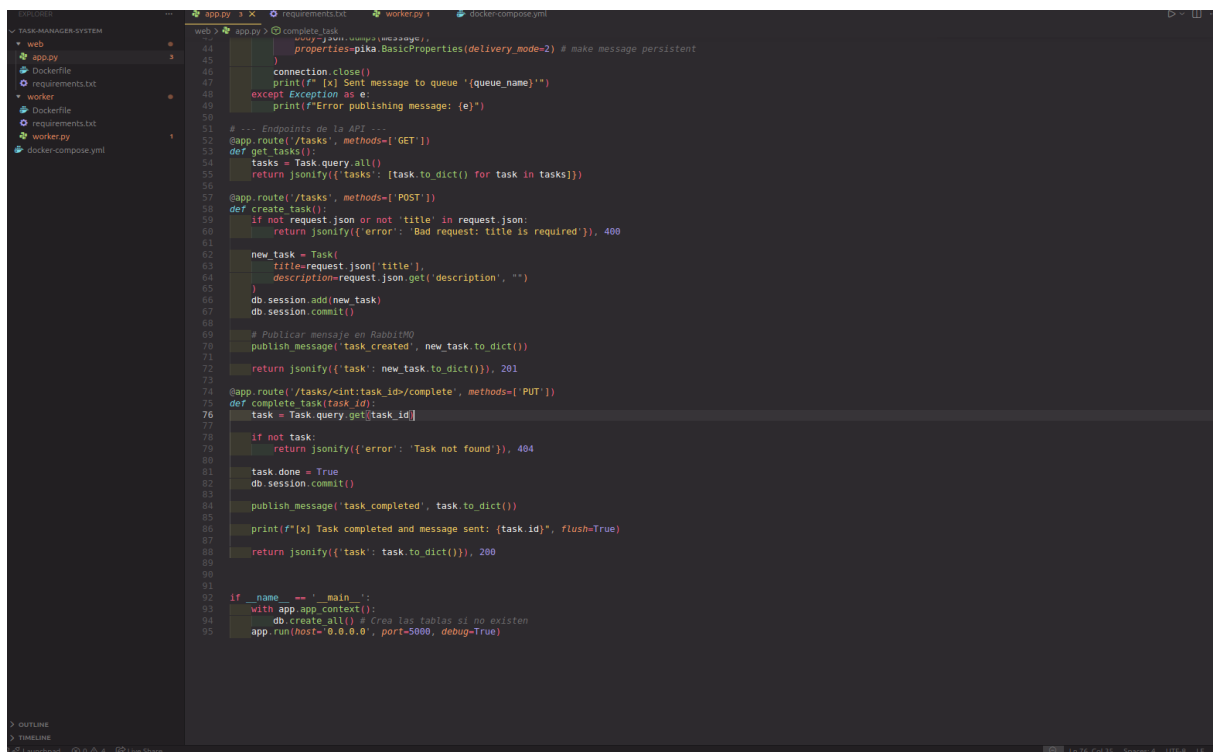
```
1 import os
2 import pika
3 import json
4 import time
5 import sys
6
7 def main():
8     rabbitmq_url = os.environ.get('RABBITMQ_URL')
9     connection = None
10
11     while not connection:
12         try:
13             connection = pika.BlockingConnection(pika.URLParameters(rabbitmq_url))
14             print('Worker: Conectado a RabbitMQ.', flush=True)
15         except pika.exceptions.AMQPConnectionError:
16             print('Worker: Esperando a RabbitMQ...', flush=True)
17             time.sleep(5)
18
19     channel = connection.channel()
20     channel.queue_declare(queue='task_created', durable=True)
21
22     def callback(ch, method, properties, body):
23         task_data = json.loads(body)
24         print(f'[x] Recibido y procesado nuevo task: ID={task_data.get('id')}, Titulo='{task_data.get('title')}', flush=True)
25         ch.basic_ack(delivery_tag=method.delivery_tag)
26
27     channel.basic_qos(prefetch_count=1)
28     channel.basic_consume(queue='task_created', on_message_callback=callback)
```

```
jaime@jaime-Modern-15-H-CL3M:~/Escritorio/MASTER/INTEGRACION DE TECNOLOGIAS/PRACTICA 5/task-manager-system$ docker compose logs worker
task-manager-worker | Worker: Conectado a RabbitMQ.
task-manager-worker | [x] Esperando mensajes. Para salir presione CTRL+C
task-manager-worker | Worker: Conectado a RabbitMQ.
task-manager-worker | [x] Esperando mensajes. Para salir presione CTRL+C
task-manager-worker | Worker: Esperando a RabbitMQ...
task-manager-worker | [x] Esperando mensajes. Para salir presione CTRL+C
task-manager-worker | Worker: Conectado a RabbitMQ.
task-manager-worker | [x] Esperando mensajes. Para salir presione CTRL+C
task-manager-worker | Worker: Conectado a RabbitMQ.
task-manager-worker | [x] Esperando mensajes. Para salir presione CTRL+C
task-manager-worker | [x] Recibido y procesado nuevo task: ID=36, Titulo='Prueba Worker'
jaime@jaime-Modern-15-H-CL3M:~/Escritorio/MASTER/INTEGRACION DE TECNOLOGIAS/PRACTICA 5/task-manager-system$ docker compose logs web
task-manager-web | * Serving Flask app 'app'
task-manager-web | * Debug mode: on
task-manager-web | WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
task-manager-web | * Running on all addresses (0.0.0.0)
task-manager-web | * Running on http://127.0.0.1:5000
task-manager-web | * Running on http://172.18.0.5:5000
task-manager-web | Press CTRL+C to quit
task-manager-web | * Restarting with stat
task-manager-web | * Debugger is active!
task-manager-web | * Debugger PIN: 349-418-905
task-manager-web | 172.18.0.1 - - [18/Nov/2025 00:58:06] "POST /tasks HTTP/1.1" 201 -
task-manager-web | 172.18.0.1 - - [18/Nov/2025 00:58:10] "GET /tasks HTTP/1.1" 200 -
task-manager-web | 172.18.0.1 - - [18/Nov/2025 00:58:10] "GET /favicon.ico HTTP/1.1" 404 -
task-manager-web | 172.18.0.1 - - [18/Nov/2025 00:58:11] "GET /tasks HTTP/1.1" 200 -
task-manager-web | 172.18.0.1 - - [18/Nov/2025 00:58:11] "GET /favicon.ico HTTP/1.1" 404 -
task-manager-web | 172.18.0.1 - - [18/Nov/2025 01:08:12] "GET /tasks HTTP/1.1" 200 -
task-manager-web | 172.18.0.1 - - [18/Nov/2025 01:08:12] "GET /favicon.ico HTTP/1.1" 404 -
task-manager-web | 172.18.0.1 - - [18/Nov/2025 01:08:26] "POST /tasks HTTP/1.1" 201 -
```


Capítulo 2

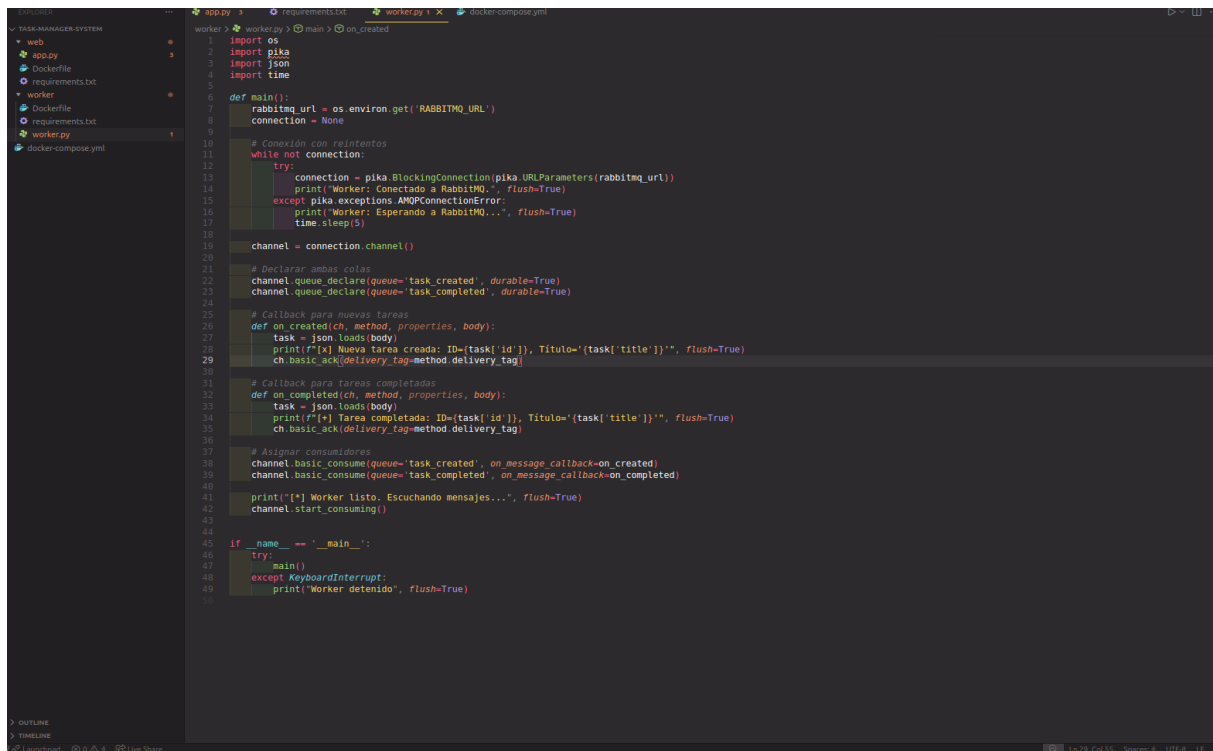
Ejercicio 1

El objetivo del ejercicio 1 es implementar la funcionalidad para marcar una tarea como completada y notificarlo a través de la cola de mensajes. Lo primero que se debe hacer es añadir un endpoint PUT en app.py para poder marcar una tarea como completada.



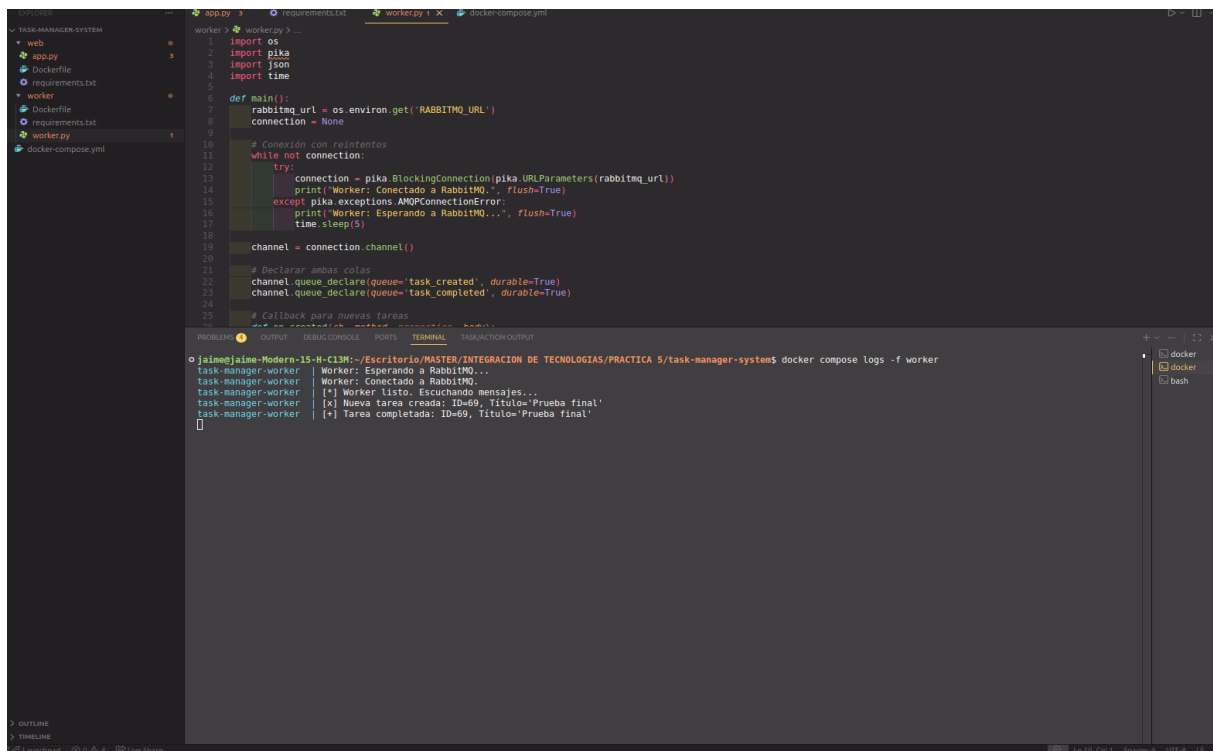
```
44     connection.close()
45     print(f"[x] Sent message to queue '{queue_name}'")
46     except Exception as e:
47         print(f"Error publishing message: {e}")
48
49 # --- Endpoints de la API ---
50
51 @app.route('/tasks', methods=['GET'])
52 def get_tasks():
53     tasks = Task.query.all()
54     return jsonify([task.to_dict() for task in tasks])
55
56 @app.route('/tasks', methods=['POST'])
57 def create_task():
58     if not request.json or not 'title' in request.json:
59         return jsonify({'error': 'Bad request: title is required'}), 400
60
61     new_task = Task(
62         title=request.json['title'],
63         description=request.json.get('description', '')
64     )
65     db.session.add(new_task)
66     db.session.commit()
67
68     # Publicar mensaje en RabbitMQ
69     publish_message('task_created', new_task.to_dict())
70
71     return jsonify({'task': new_task.to_dict()}), 201
72
73 @app.route('/tasks/<int:task_id>/complete', methods=['PUT'])
74 def complete_task(task_id):
75     task = Task.query.get(task_id)
76
77     if not task:
78         return jsonify({'error': 'Task not found'}), 404
79
80     task.done = True
81     db.session.commit()
82
83     publish_message('task_completed', task.to_dict())
84
85     print(f"[x] Task completed and message sent: {task_id}")
86
87     return jsonify({'task': task.to_dict()}), 200
88
89
90
91 if __name__ == '__main__':
92     with app.app_context():
93         db.create_all() # Crea las tablas si no existen
94     app.run(host='0.0.0.0', port=5000, debug=True)
```

Además, será necesario modificar worker.py para crear una nueva cola llamada task-completed donde se publicará un mensaje con los datos de la tarea actualizada.



```
1 import os
2 import pika
3 import json
4 import time
5
6 def main():
7     rabbitmq_url = os.environ.get('RABBITMQ_URL')
8     connection = None
9
10    # Conexión con reintentos
11    while not connection:
12        try:
13            connection = pika.BlockingConnection(pika.URLParameters(rabbitmq_url))
14            print('Worker: Conectado a RabbitMQ.', flush=True)
15        except pika.exceptions.AMQPConnectionError:
16            print('Worker: Esperando a RabbitMQ...', flush=True)
17            time.sleep(5)
18
19    channel = connection.channel()
20
21    # Declarar ambas colas
22    channel.queue_declare(queue='task_created', durable=True)
23    channel.queue_declare(queue='task_completed', durable=True)
24
25    # Callback para nuevas tareas
26    def on_created(ch, method, properties, body):
27        task = json.loads(body)
28        print(f'[x] Nueva tarea creada: ID={task['id']}, Título='{task['title']}'", flush=True)
29        ch.basic_ack(delivery_tag=method.delivery_tag)
30
31    # Callback para tareas completadas
32    def on_completed(ch, method, properties, body):
33        task = json.loads(body)
34        print(f'[*] Tarea completada: ID={task['id']}, Título='{task['title']}'", flush=True)
35        ch.basic_ack(delivery_tag=method.delivery_tag)
36
37    # Asignar consumidores
38    channel.basic_consume(queue='task_created', on_message_callback=on_created)
39    channel.basic_consume(queue='task_completed', on_message_callback=on_completed)
40
41    print('[*] Worker listo. Escuchando mensajes...', flush=True)
42    channel.start_consuming()
43
44
45 if __name__ == '__main__':
46     try:
47         main()
48     except KeyboardInterrupt:
49         print('Worker detenido', flush=True)
```

Una vez modificado worker.py para poder ver el mensaje de la cola creada se podrá ver por terminal el mensaje.

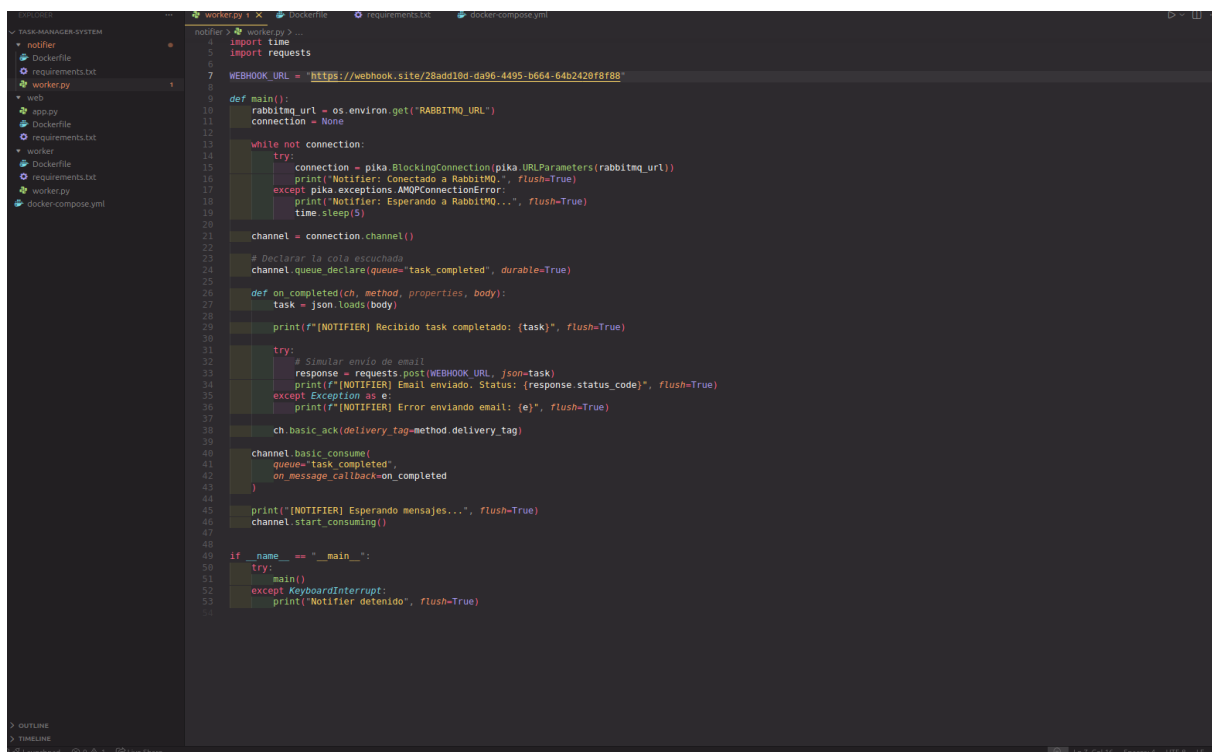


```
o jaime@jaime-MacBook-Pro: ~/Escritorio/MASTER/INTEGRACION DE TECNOLOGIAS/PRACTICA 5/task-manager-system$ docker compose logs -f worker
task-manager-worker | Worker: Esperando a RabbitMQ...
task-manager-worker | Worker: Conectado a RabbitMQ.
task-manager-worker | [*] Worker listo. Escuchando mensajes...
task-manager-worker | [x] Nueva tarea creada: ID=69, Título='Prueba final'
task-manager-worker | [*] Tarea completada: ID=69, Título='Prueba final'
```

Capítulo 3

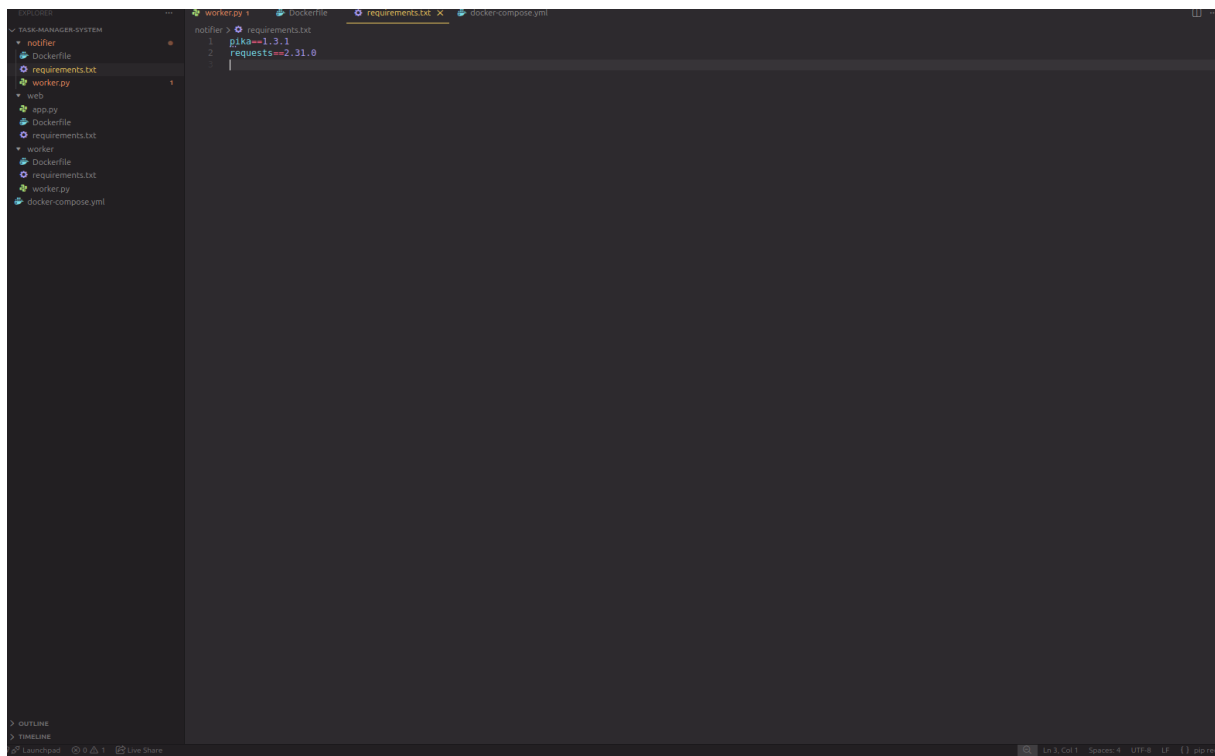
Ejercicio 2

El objetivo del ejercicio 2 es crear un tercer servicio dedicado exclusivamente a enviar notificaciones, desacoplando aún más la lógica. Para ello, lo primero es crear un nuevo directorio llamado `notifier` con su propio `worker.py`, `requirements.txt` y `Dockerfile`. En `worker.py` se desarrollará la lógica para enviar las notificaciones.



```
1 import time
2 import requests
3
4 WEBHOOK_URL = "https://webhook.site/28add10d-da96-4495-b664-64b2420f8f88"
5
6 def main():
7     rabbitmq_url = os.environ.get("RABBITMQ_URL")
8     connection = None
9
10     while not connection:
11         try:
12             connection = pika.BlockingConnection(pika.URLParameters(rabbitmq_url))
13             print("[NOTIFIER] Conectado a RabbitMQ.", flush=True)
14         except pika.exceptions.AMQPConnectionError:
15             print("[NOTIFIER] Esperando a RabbitMQ...", flush=True)
16             time.sleep(5)
17
18     channel = connection.channel()
19
20     # Declarar la cola escuchada
21     channel.queue_declare(queue="task_completed", durable=True)
22
23     def on_completed(ch, method, properties, body):
24         task = json.loads(body)
25         print(f"[NOTIFIER] Recibido task completado: {task}", flush=True)
26
27         # Enviar email
28         try:
29             response = requests.post(WEBHOOK_URL, json=task)
30             print(f"[NOTIFIER] Email enviado. Status: {response.status_code}", flush=True)
31         except Exception as e:
32             print(f"[NOTIFIER] Error enviando email: {e}", flush=True)
33
34     ch.basic_ack(delivery_tag=method.delivery_tag)
35
36     channel.basic_consume(
37         queue="task_completed",
38         on_message_callback=on_completed,
39     )
40
41     print("[NOTIFIER] Esperando mensajes...", flush=True)
42     channel.start_consuming()
43
44 if __name__ == "__main__":
45     try:
46         main()
47     except KeyboardInterrupt:
48         print("Notifier detenido", flush=True)
```

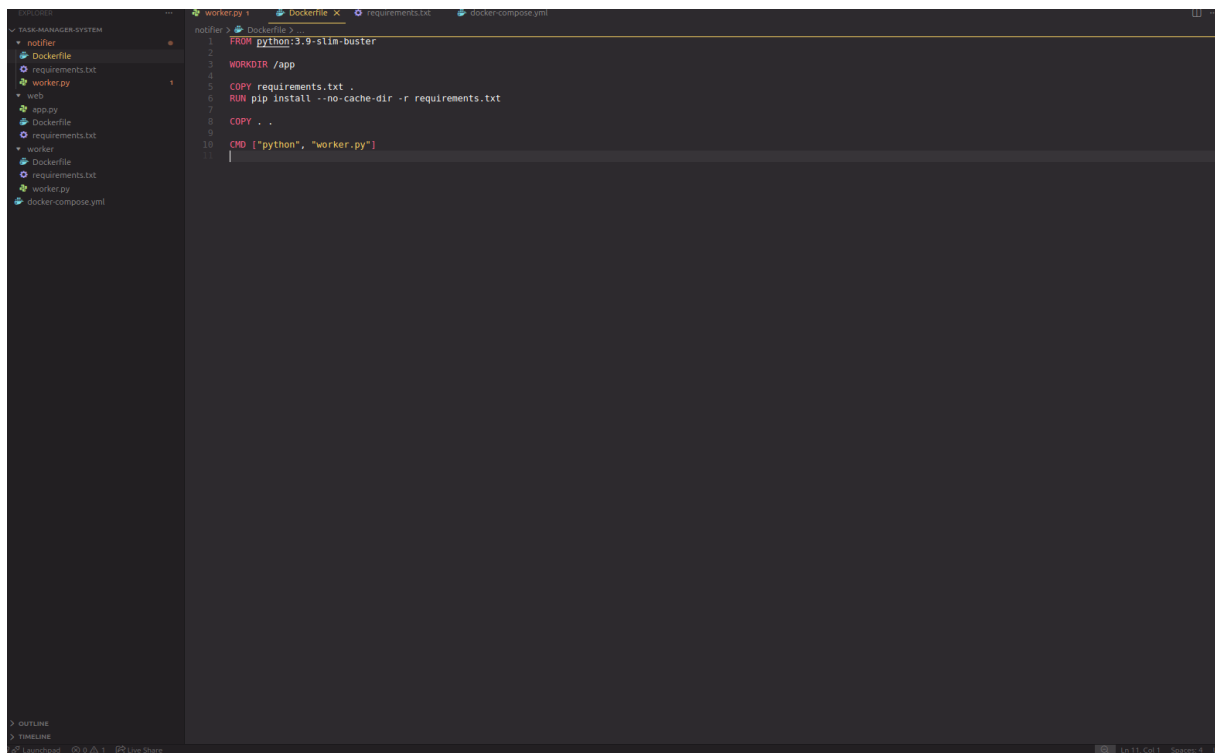
En requirements.txt se pondrán las dependencias de python.



The screenshot shows a code editor with a sidebar on the left displaying a project structure. The main editor area shows the contents of the `requirements.txt` file. The file contains two lines of text: `pika==1.3.1` and `requests==2.31.0`. The sidebar shows a tree view with folders and files, including `notifier`, `requirements.txt`, `worker.py`, `web`, `app.py`, `requirements.txt`, `worker`, `requirements.txt`, `worker.py`, and `docker-compose.yml`. The bottom status bar indicates the file is at line 3, column 1, with a space character, in UTF-8 encoding, with LF line endings.

```
notifier > requirements.txt
1 pika==1.3.1
2 requests==2.31.0
3
```

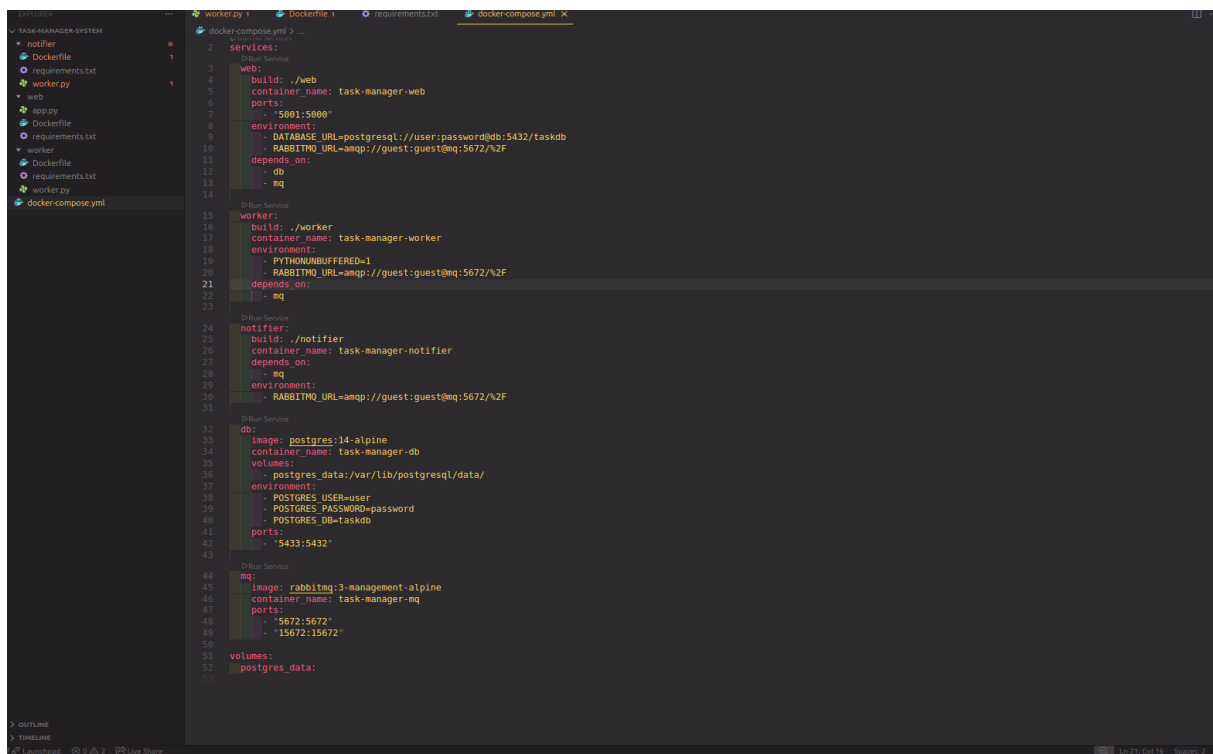
Por último, el dockerfile será el encargado de crear la imagen del docker.



The screenshot shows the same code editor with the `Dockerfile` file open. The file contains the following instructions: `FROM python:3.9-slim-buster`, `WORKDIR /app`, `COPY requirements.txt .`, `RUN pip install --no-cache-dir -r requirements.txt`, `COPY . .`, and `CMD ["python", "worker.py"]`. The sidebar shows the same project structure as the previous screenshot. The bottom status bar indicates the file is at line 11, column 1, with a space character, in UTF-8 encoding, with LF line endings.

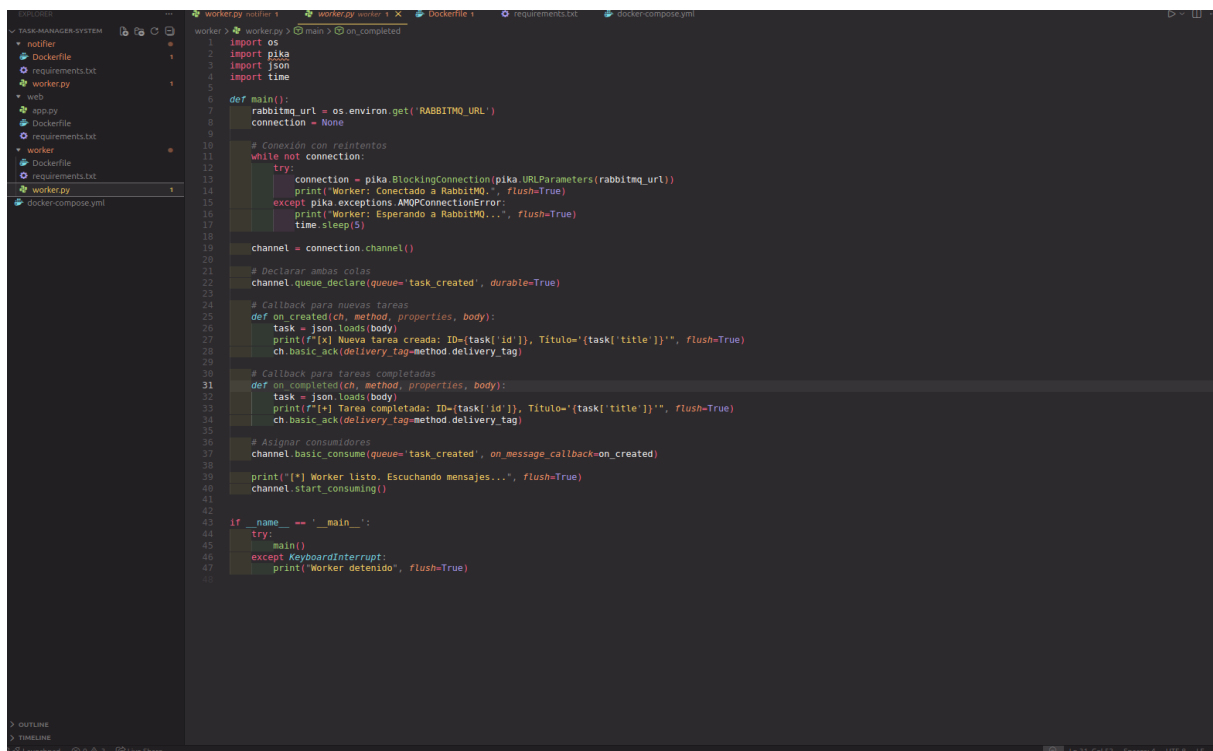
```
notifier > Dockerfile
1 FROM python:3.9-slim-buster
2
3 WORKDIR /app
4
5 COPY requirements.txt .
6 RUN pip install --no-cache-dir -r requirements.txt
7
8 COPY . .
9
10 CMD ["python", "worker.py"]
11
```

Una vez configurado el notifier es necesario añadir a docker-compose.yml el nuevo servicio.



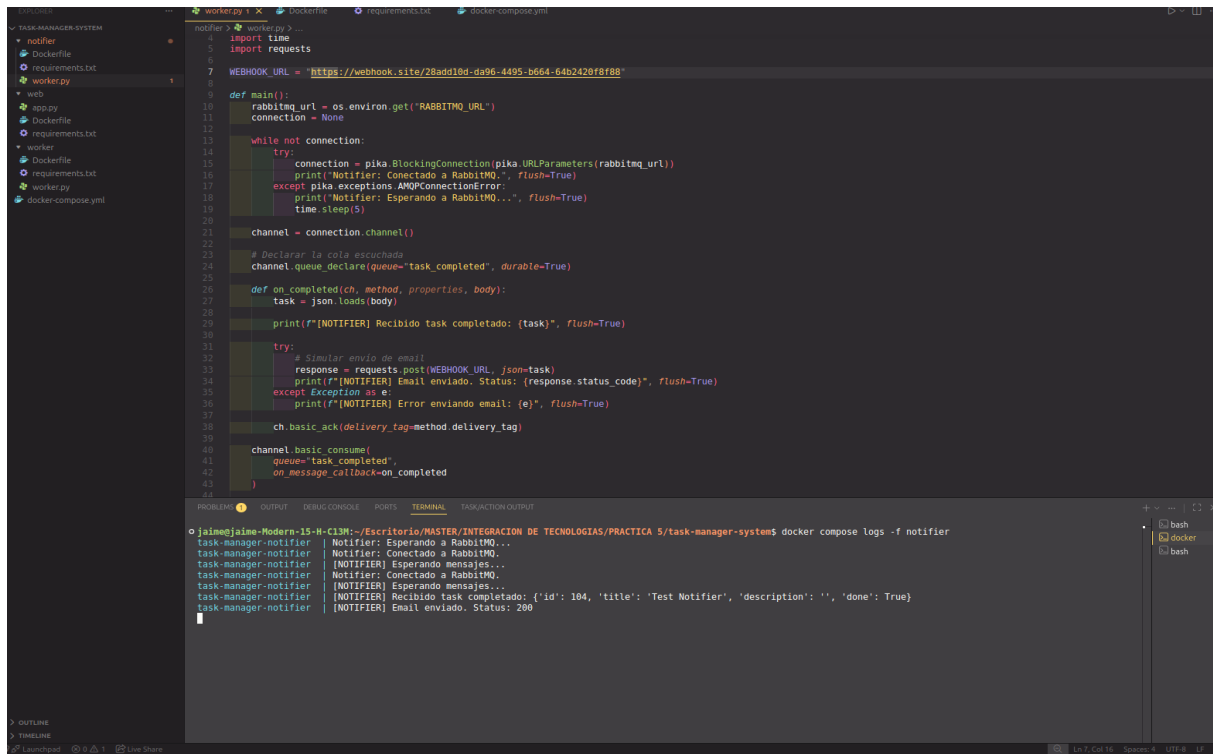
```
1 services:
2   web:
3     build: ./web
4     container_name: task-manager-web
5     ports:
6       - "5001:5000"
7     environment:
8       - DATABASE_URL=postgresql://user:password@db:5432/taskdb
9       - RABBITMQ_URL=amqp://guest:guest@mq:5672/%2F
10    depends_on:
11      - db
12      - mq
13
14   worker:
15     build: ./worker
16     container_name: task-manager-worker
17     environment:
18       - PYTHONUNBUFFERED=1
19       - RABBITMQ_URL=amqp://guest:guest@mq:5672/%2F
20    depends_on:
21      - mq
22
23   notifier:
24     build: ./notifier
25     container_name: task-manager-notifier
26     depends_on:
27       - mq
28     environment:
29       - RABBITMQ_URL=amqp://guest:guest@mq:5672/%2F
30
31   db:
32     image: postgres:14-alpine
33     container_name: task-manager-db
34     volumes:
35       - postgres_data:/var/lib/postgresql/data/
36     environment:
37       - POSTGRES_USER=user
38       - POSTGRES_PASSWORD=password
39       - POSTGRES_DB=taskdb
40     ports:
41       - "5433:5432"
42
43   mq:
44     image: rabbitmq:3-management-alpine
45     container_name: task-manager-mq
46     ports:
47       - "5672:5672"
48       - "15672:15672"
49
50 volumes:
51   postgres_data:
```

El siguiente paso es modificar el worker.py de worker para que no escuche la cola task-completed.



```
1 import os
2 import pika
3 import json
4 import time
5
6 def main():
7     rabbitmq_url = os.environ.get('RABBITMQ_URL')
8     connection = None
9
10    # Conexión con reintentos
11    while not connection:
12        try:
13            connection = pika.BlockingConnection(pika.URLParameters(rabbitmq_url))
14            print("Worker: Conectado a RabbitMQ.", flush=True)
15        except pika.exceptions.AMQPConnectionError:
16            print("Worker: Esperando a RabbitMQ...", flush=True)
17            time.sleep(5)
18
19    channel = connection.channel()
20
21    # Declarar ambas colas
22    channel.queue_declare(queue='task_created', durable=True)
23
24    # Callback para nuevas tareas
25    def on_created(ch, method, properties, body):
26        task = json.loads(body)
27        print(f"[x] Nueva tarea creada: ID={task['id']}, Título='{task['title']}'", flush=True)
28        ch.basic_ack(delivery_tag=method.delivery_tag)
29
30    # Callback para tareas completadas
31    def on_completed(ch, method, properties, body):
32        task = json.loads(body)
33        print(f"[x] Tarea completada: ID={task['id']}, Título='{task['title']}'", flush=True)
34        ch.basic_ack(delivery_tag=method.delivery_tag)
35
36    # Asignar consumidores
37    channel.basic_consume(queue='task_created', on_message_callback=on_created)
38
39    print("[*] Worker listo. Escuchando mensajes...", flush=True)
40    channel.start_consuming()
41
42
43 if __name__ == '__main__':
44     try:
45         main()
46     except KeyboardInterrupt:
47         print("Worker detenido", flush=True)
```

Una vez realizados todos estos cambios, podemos ver como notifier es el encargado de consumir la cola de mensajes.



The image shows a VS Code editor with a Python script named `notifier.py` and its terminal output. The script is designed to connect to a RabbitMQ queue and receive messages via a webhook.

```
notifier.py
1  #!/usr/bin/env python
2  import time
3  import requests
4
5  WEBHOOK_URL = "https://webhook.site/28add18d-da96-4495-b664-64b2428f8f88"
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("Notifier: Conectado a RabbitMQ.", flush=True)
15          except pika.exceptions.AMQPConnectionError:
16              print("Notifier: Esperando a RabbitMQ...", flush=True)
17              time.sleep(5)
18
19      channel = connection.channel()
20
21      # Declarar la cola escuchada
22      channel.queue_declare(queue="task_completed", durable=True)
23
24      def on_completed(ch, method, properties, body):
25          task = json.loads(body)
26          print(f"[NOTIFIER] Recibido task completado: {task}", flush=True)
27
28          try:
29              # Simular envio de email
30              response = requests.post(WEBHOOK_URL, json=task)
31              print(f"[NOTIFIER] Email enviado. Status: {response.status_code}", flush=True)
32          except Exception as e:
33              print(f"[NOTIFIER] Error enviando email: {e}", flush=True)
34
35          ch.basic_ack(delivery_tag=method.delivery_tag)
36
37      channel.basic_consume(
38          queue="task_completed",
39          on_message_callback=on_completed,
40      )
41
42  if __name__ == '__main__':
43      main()
```

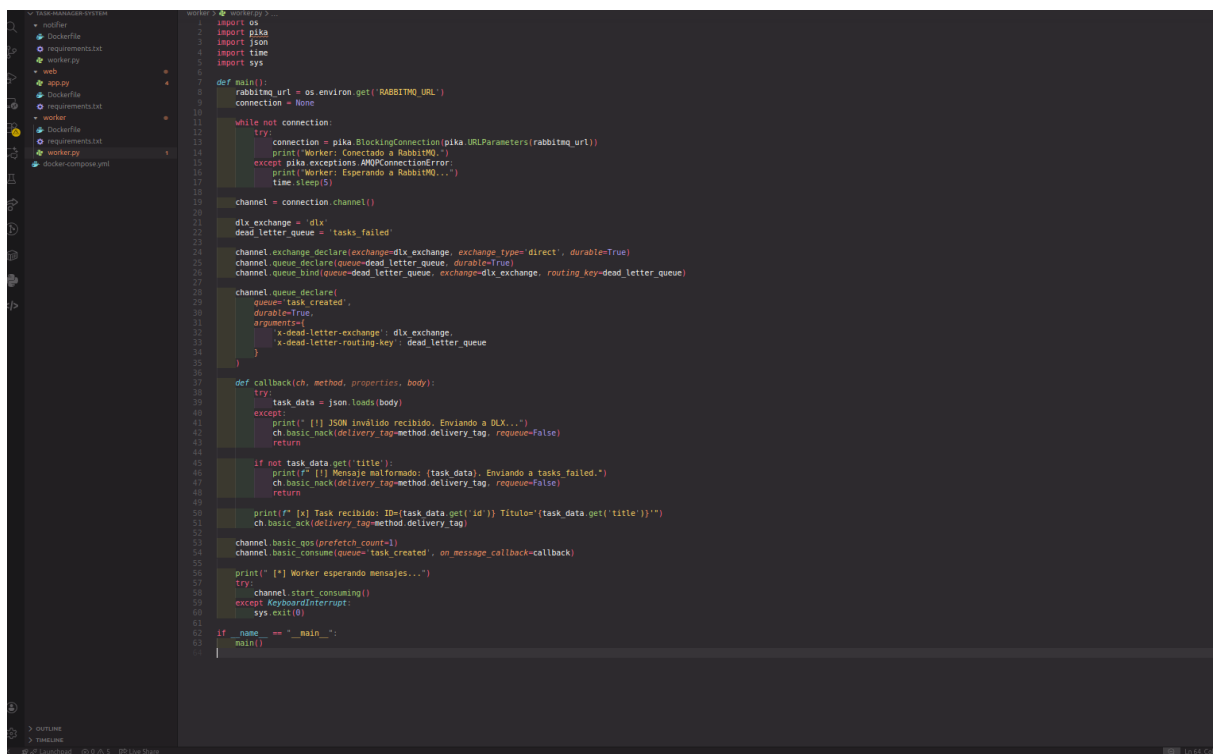
The terminal output shows the execution of the script using `docker compose logs -f notifier`. The logs indicate that the notifier successfully connected to RabbitMQ, waited for messages, and received a task completion message. It then simulated sending an email and responded with a 200 status code.

```
o jaime@jaime-Modern-15-H-CLIM:~/Escritorio/MASTER/INTEGRACION DE TECNOLOGIAS/PRACTICA 5/task-manager-system$ docker compose logs -f notifier
task-manager-notifier | Notifier: Esperando a RabbitMQ...
task-manager-notifier | Notifier: Conectado a RabbitMQ.
task-manager-notifier | [NOTIFIER] Esperando mensajes...
task-manager-notifier | Notifier: Conectado a RabbitMQ.
task-manager-notifier | [NOTIFIER] Esperando mensajes...
task-manager-notifier | [NOTIFIER] Recibido task completado: {'id': 104, 'title': 'Test Notifier', 'description': '', 'done': True}
task-manager-notifier | [NOTIFIER] Email enviado. Status: 200
```

Capítulo 4

Ejercicio 3

El objetivo del ejercicio 3 es aumentar la resiliencia del sistema de mensajería. Para ello, es necesario modificar los archivos `app.py` y `worker.py`. El primer paso ha sido adaptar `worker.py` para que al recibir un mensaje sin título rechace el mensaje sin enconlarlo.



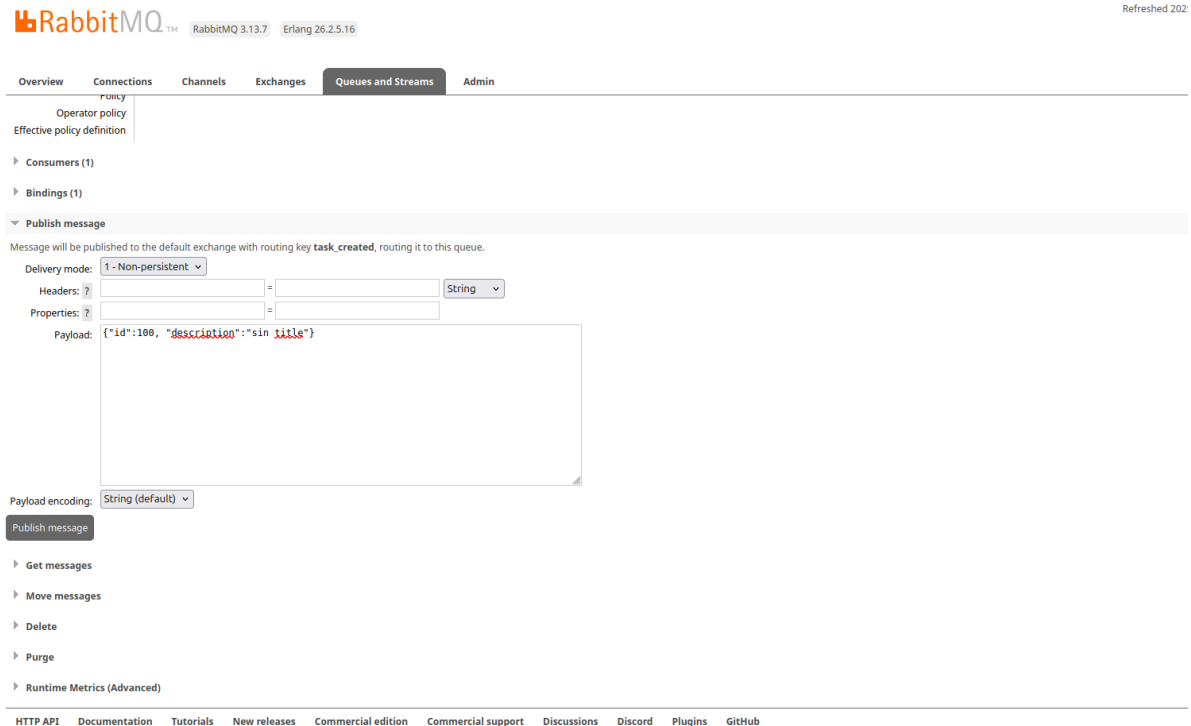
```
1 import os
2 import pika
3 import json
4 import time
5 import sys
6
7 def main():
8     rabbitmq_url = os.environ.get('RABBITMQ_URL')
9     connection = None
10
11     while not connection:
12         try:
13             connection = pika.BlockingConnection(pika.URLParameters(rabbitmq_url))
14             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,
33             'x-dead-letter-routing-key': dead_letter_queue
34         }
35     )
36
37     def callback(ch, method, properties, body):
38         try:
39             task_data = json.loads(body)
40         except:
41             print(f'[] JSON inválido recibido. Enviando a DLX...')
42             ch.basic_ack(delivery_tag=method.delivery_tag, requeue=False)
43             return
44
45         if not task_data.get('title'):
46             print(f'[] Mensaje malformado: {task_data}. Enviando a tasks failed.')
47             ch.basic_ack(delivery_tag=method.delivery_tag, requeue=False)
48             return
49
50         print(f'[] Task recibido: ID={task_data.get('id')} Titulo={task_data.get('title')}')
51         ch.basic_ack(delivery_tag=method.delivery_tag)
52
53     channel.basic_qos(prefetch_count=1)
54     channel.basic_consume(queue='task_created', on_message_callback=callback)
55
56     print('[] Worker esperando mensajes...')
57     try:
58         channel.start_consuming()
59     except KeyboardInterrupt:
60         sys.exit(0)
61
62 if __name__ == '__main__':
63     main()
```

A continuación se ha modificado app.py para mandar cualquier mensaje rechazado por worker de la cola task-created a task-failed.

```
web > app.py > create_task
1 import os
2 import json
3 import time
4 import pika
5 from flask import Flask, request, jsonify
6 from flask_sqlalchemy import SQLAlchemy
7 from sqlalchemy.exc import OperationalError
8
9 # --- Inicialización de Flask y DB ---
10 app = Flask(__name__)
11 app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL')
12 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
13 db = SQLAlchemy(app)
14
15 # --- Intentar conexión a la DB antes de iniciar ---
16 connected = False
17 while not connected:
18     try:
19         with app.app_context():
20             db.create_all()
21             connected = True
22             print("Base de datos lista.")
23     except OperationalError:
24         print("Postgres no listo, reintentando en 2 segundos...")
25         time.sleep(2)
26
27
28 # --- Modelo ---
29 class Task(db.Model):
30     id = db.Column(db.Integer, primary_key=True)
31     title = db.Column(db.String(120), nullable=False)
32     description = db.Column(db.String(255), nullable=True)
33     done = db.Column(db.Boolean, default=False)
34
35     def to_dict(self):
36         return {
37             'id': self.id,
38             'title': self.title,
39             'description': self.description,
40             'done': self.done
41         }
42
43
44 # --- Configuración de RabbitMQ ---
45 RABBITMQ_URL = os.environ.get('RABBITMQ_URL')
46
47 def publish_message(queue_name, message):
48     try:
49         connection = pika.BlockingConnection(pika.URLParameters(RABBITMQ_URL))
50         channel = connection.channel()
51
52         # Declarar DLX y cola de errores
53         dlx_exchange = "dlx"
54         dead_letter_queue = "tasks_failed"
55
56         channel.exchange_declare(exchange=dlx_exchange, exchange_type="direct", durable=True)
57         channel.queue_declare(queue=dead_letter_queue, durable=True)
58         channel.queue_bind(queue=dead_letter_queue, exchange=dlx_exchange, routing_key=dead_letter_queue)
59
60         # Declarar cola principal con DLX
```

```
web > app.py >
58 channel.queue_bind(queue=dead_letter_queue, exchange=dlx_exchange, routing_key=dead_letter_queue)
59
60 # Declarar cola principal con DLX
61 channel.queue_declare(
62     queue=queue_name,
63     durable=True,
64     arguments={
65         "x-dead-letter-exchange": dlx_exchange,
66         "x-dead-letter-routing-key": dead_letter_queue
67     }
68 )
69
70 channel.basic_publish(
71     exchange="",
72     routing_key=queue_name,
73     body=json.dumps(message),
74     properties=pika.BasicProperties(delivery_mode=2)
75 )
76 connection.close()
77
78 print(f"[X] Mensaje enviado a '{queue_name}'")
79
80 except Exception as e:
81     print(f"Error al publicar mensaje: {e}")
82
83
84 # --- Endpoints ---
85 @app.route('/tasks', methods=['GET'])
86 def get_tasks():
87     tasks = Task.query.all()
88     return jsonify({"tasks": [t.to_dict() for t in tasks]})
89
90
91 @app.route('/tasks', methods=['POST'])
92 def create_task():
93     if not request.json or "title" not in request.json:
94         return jsonify({"error": "Bad request: title is required"}), 400
95
96     new_task = Task(
97         title=request.json["title"],
98         description=request.json.get("description", "")
99     )
100     db.session.add(new_task)
101     db.session.commit()
102
103     publish_message("task_created", new_task.to_dict())
104
105     return jsonify({"task": new_task.to_dict()}), 201
106
107
108 # --- Run ---
109 if __name__ == "__main__":
110     with app.app_context():
111         db.create_all()
112
113     app.run(host="0.0.0.0", port=5000, debug=True)
114
```


Para comprobar que el funcionamiento es correcto, desde Rabbit introducirems una tarea sin título.



RabbitMQ 3.13.7 Erlang 26.2.5.16

Overview Connections Channels Exchanges **Queues and Streams** Admin

Operator policy
Effective policy definition

Consumers (1)

Bindings (1)

Publish message

Message will be published to the default exchange with routing key `task_created`, routing it to this queue.

Delivery mode: 1 - Non-persistent

Headers: ? = String

Properties: ? =

Payload: `{"id":100, "description":"sin title"}`

Payload encoding: String (default)

Publish message

Get messages

Move messages

Delete

Purge

Runtime Metrics (Advanced)

HTTP API Documentation Tutorials New releases Commercial edition Commercial support Discussions Discord Plugins GitHub

Al no tener título la tarea se deb guardar en la cola task-failed, como vemos a continuación es lo esperado por lo que el funcionamiento sería correcto.

RabbitMQ

3.13.7

Erlang 26.2.5.16

Refreshed 202

Overview

Connections

Channels

Exchanges

Queues and Streams

Admin

Queues

All queues (3)

Pagination

Page 1 of 1 - Filter: ☐ Regex ?

Overview

Messages

Message rates

Virtual host

Name

Type

Features

State

Ready

Unacked

Total

incoming

deliver / get

ack

/

task_completed

classic

D

running

0

0

0

/

task_created

classic

D DLX DLK

running

0

0

0

0.20/s

0.20/s

0.00/s

/

tasks_failed

classic

D

running

2

0

2

Add a new queue

HTTP API

Documentation

Tutorials

New releases

Commercial edition

Commercial support

Discussions

Discord

Plugins

GitHub