PFO 3: Rediseño como Sistema Distribuido (Cliente-Servidor)

Enunciado

Objetivo: Transformar el sistema en una arquitectura distribuida usando sockets.

Consignas:

1. Diseña un diagrama que incluya:
    - Clientes (móviles, web).
    - Balanceador de carga (Nginx/HAProxy).
    - Servidores workers (cada uno con su pool de hilos).
    - Cola de mensajes (RabbitMQ) para comunicación entre servidores.
    - Almacenamiento distribuido (PostgreSQL, S3).
2. Implementa en Python:
    - Un servidor que reciba tareas por socket y las distribuya a workers.
    - Un cliente que envíe tareas y reciba resultados.

Entregables:

- Diagrama del sistema.
- Código del servidor y cliente en repositorio de Github

Desarrollo:

Paso 1: Diseño de diagrama de arquitectura distribuida.

Estructura del diagrama

Componentes principales:

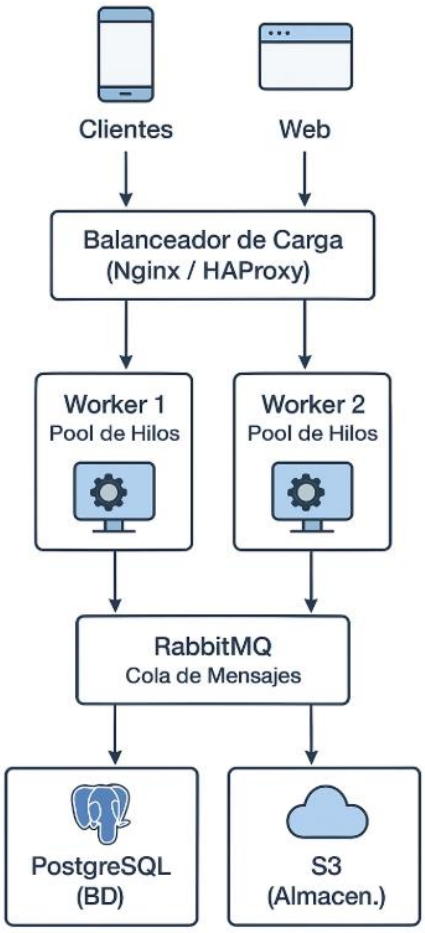
1. **Clientes**
    - Tipos: móviles y web
    - Envían tareas al sistema

2. **Balanceador de carga (Nginx / HAProxy)**
    - Distribuye las solicitudes de los clientes a los servidores workers
    - Permite escalabilidad

3. **Servidores Workers**
    - Cada servidor tiene un pool de hilos
    - Procesa las tareas concurrentemente
    - Puede enviar tareas a la cola de mensajes si necesita coordinación

4. **Cola de mensajes (RabbitMQ)**
    - Para comunicación entre servidores o workers
    - Permite procesamiento asíncrono

5. **Almacenamiento distribuido**
    - PostgreSQL: datos estructurados
    - S3: archivos o datos grandes



Estructura básica del proyecto

```
PFO3_Redes/
│
├── servidor.py      # Código del servidor con pool de hilos
├── cliente.py       # Código del cliente
└── README.md        # Explicación y cómo ejecutarlo
```

Paso 2: Implementar un servidor básico en Python que acepte conexiones por sockets.

servidor.py: será el punto central que:

1. Escucha conexiones de clientes por sockets TCP.
2. Recibe "tareas" (por ejemplo, cálculos, peticiones, etc.).
3. Usa un pool de hilos (thread pool) para procesar varias tareas al mismo tiempo.
4. Devuelve los resultados a los clientes.

Más adelante extenderlo con RabbitMQ (para simular coordinación entre varios servidores).

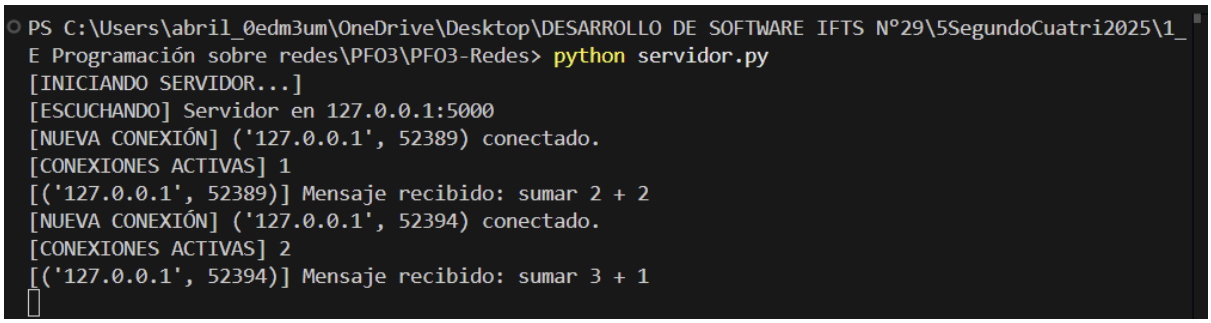
Qué hace este servidor

- Escucha en localhost:5000.
- Cada cliente que se conecta se maneja en un **hilo independiente**.
- Recibe mensajes, los procesa y devuelve una respuesta.
- Permite múltiples clientes simultáneos.

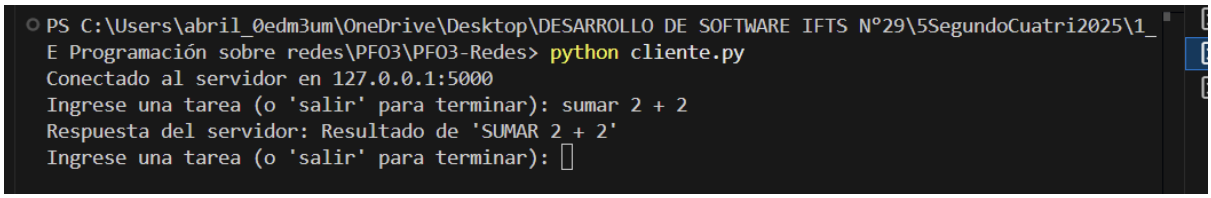
Paso 3: Implementar el cliente que envía y recibe datos por sockets.

cliente.py: se conecta al servidor, envía una tarea, espera la respuesta y la muestra por pantalla.

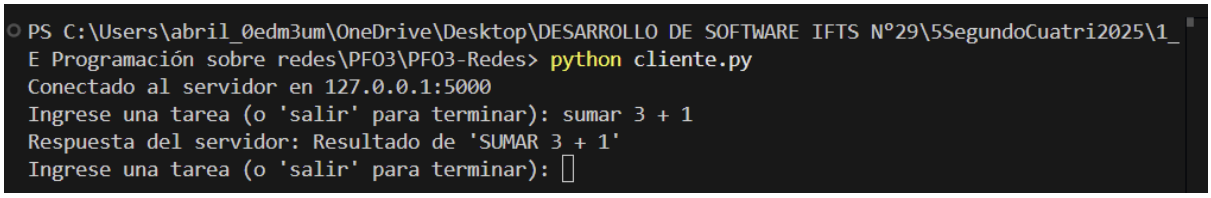
Terminal 1 - Servidor



Terminal 2 – Cliente 1



Terminal 3 – Cliente 2



Las pruebas nos confirman que tenemos:

- Servidor multihilo que acepta múltiples clientes.
- Cliente que puede enviar tareas y recibir resultados.

Paso 4: Mejorar el servidor agregando pool de hilos (ThreadPoolExecutor) para manejar múltiples tareas simultáneamente, en lugar de crear un hilo nuevo por cada conexión. Esto mejora el rendimiento y simula mejor el comportamiento de un worker server real.

servidor_threadpool.py: servidor versión mejorada.

Cambios:

Antes	**Ahora**
Se creaba un nuevo thread por cliente.	Se usa un ThreadPoolExecutor con un número fijo de hilos.
No había límite de conexiones simultáneas.	El número de workers está controlado (MAX_WORKERS).
Menos eficiente para muchas conexiones.	Más realista y estable.

Terminal 1 -Servidor

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_
E Programación sobre redes\PFO3\PFO3-Redes> python servidor_threadpool.py
>>
[INICIANDO SERVIDOR CON POOL DE HILOS...]
[ESCUCHANDO] Servidor en 127.0.0.1:5000
[POOL ACTIVO] Capacidad: 5 workers

[NUEVA CONEXIÓN] ('127.0.0.1', 51608) conectado.
[('127.0.0.1', 51608)] Tarea recibida: sumar 2 + 2
[NUEVA CONEXIÓN] ('127.0.0.1', 64186) conectado.
[('127.0.0.1', 64186)] Tarea recibida: sumar 3 + 1
```

Terminal 2 – Cliente 1

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_
E Programación sobre redes\PFO3\PFO3-Redes> python cliente.py
Conectado al servidor en 127.0.0.1:5000
Ingrese una tarea (o 'salir' para terminar): sumar 2 + 2
Respuesta del servidor: Resultado procesado por worker: 'SUMAR 2 + 2'
Ingrese una tarea (o 'salir' para terminar):
```

Terminal 3 – Cliente 2

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_
E Programación sobre redes\PFO3\PFO3-Redes> python cliente.py
Conectado al servidor en 127.0.0.1:5000
Ingrese una tarea (o 'salir' para terminar): sumar 3 + 1
Respuesta del servidor: Resultado procesado por worker: 'SUMAR 3 + 1'
Ingrese una tarea (o 'salir' para terminar):
```

El sistema ya maneja múltiples clientes concurrentes y está funcionando como un servidor worker real.

Paso 5: Integrar una cola de mensajes (RabbitMQ) que permitirá enviar tareas o resultados entre distintos *workers* o *servidores* (simulando varios nodos).

Se simula un worker secundario que escucha en RabbitMQ y procesa tareas que el servidor principal le envía.

Abrir Docker Desktop
Verificar que Docker está corriendo

```
E Programación sobre redes\PFO3\PFO3-Redes> docker --version
Docker version 28.4.0, build d8eb465
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_
E Programación sobre redes\PFO3\PFO3-Redes> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Descargar y ejecutar RabbitMQ

Descarga la imagen oficial de **RabbitMQ** con el **panel de administración web** incluido.
La opción -d lo ejecuta en segundo plano (detached mode).
Los puertos:

- 5672 → conexión del backend (Python, workers, etc.)
- 15672 → panel web de administración.

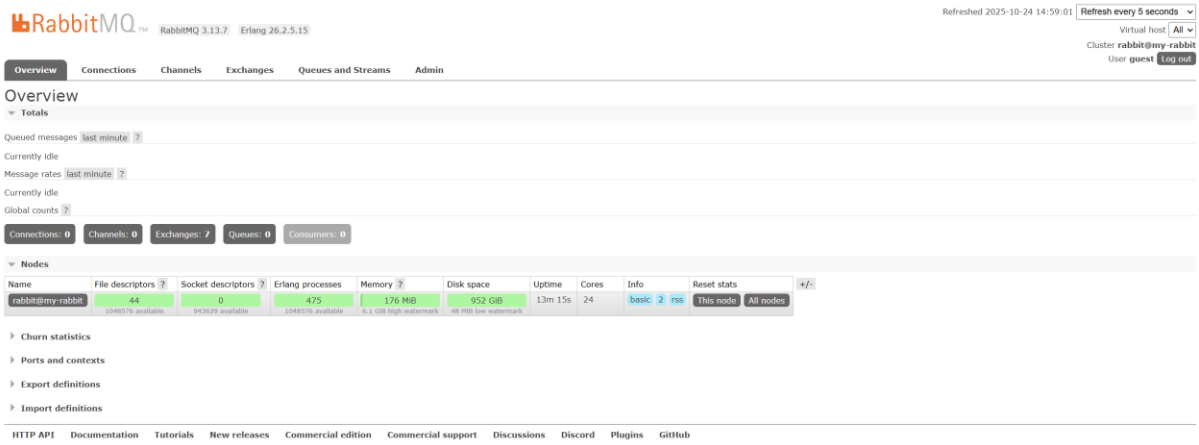
```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_
E Programación sobre redes\PFO3\PFO3-Redes> docker run -d --hostname my-rabbit --name rabbitmq -p
 5672:5672 -p 15672:15672 rabbitmq:3-management
>>
Unable to find image 'rabbitmq:3-management' locally
3-management: Pulling from library/rabbitmq
8ddbdcbe86b0: Pull complete
e32118121bc1: Pull complete
a7620a19f6fc: Pull complete
5dfddc3658b8: Pull complete
a7620a19f6fc: Pull complete
5dfddc3658b8: Pull complete
Digest: sha256:3b65f271d3e6028ef7609dadbc36e41c2fc29f67ab62103099ae44792c8dbda8
Status: Downloaded newer image for rabbitmq:3-management
ffef0c7b7708ad7655656fbbdc4bc607610cba8773ab3f9994a09879efffd0a1
```

Verificar que RabbitMQ está corriendo

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_
E Programación sobre redes\PFO3\PFO3-Redes> docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS
PORTS                                                                                  NA
MES
ffef0c7b7708   rabbitmq:3-management   "docker-entrypoint.s…"   42 seconds ago   Up 41 seconds
0.0.0.0:5672->5672/tcp, [::]:5672->5672/tcp, 0.0.0.0:15672->15672/tcp, [::]:15672->15672/tcp   ra
bbitmq
```

Abrir en el navegador http://localhost:15672/

Username: guest Password: guest



Se agregan nuevos componentes paraintegración con RabbitMQ:

worker_rabbitmq.py: este archivo actuará como worker remoto, recibe tareas desde RabbitMQ, las procesa y devuelve confirmación.

servidor_rabbit.py: servidor modificado para usar RabbitMQ, publica tareas enviadas por clientes en la cola.

Terminal 1 – Worker

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_E Programación sobre redes\PF03\PF03-Redes> python worker_rabbitmq.py
[WORKER] Esperando tareas...
[WORKER] Tarea recibida: procesar datos
[WORKER] Tarea completada: procesar datos
```

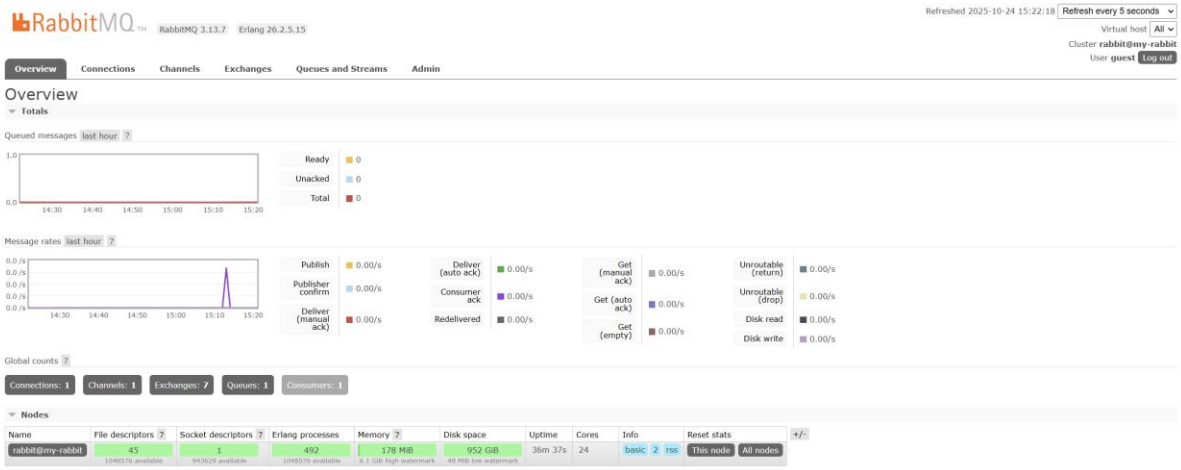
Terminal 2 – Servidor

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_E Programación sobre redes\PF03\PF03-Redes> python servidor_rabbit.py
[SERVIDOR RABBITMQ INICIADO...]
[ESCUCHANDO] Servidor en 127.0.0.1:5000
[POOL ACTIVO] Capacidad: 3 workers

[NUEVA CONEXIÓN] ('127.0.0.1', 63174) conectado.
[('127.0.0.1', 63174)] Enviando tarea a RabbitMQ: procesar datos
```

Terminal 3 – Cliente

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_E Programación sobre redes\PF03\PF03-Redes> python cliente.py
Conectado al servidor en 127.0.0.1:5000
Ingrese una tarea (o 'salir' para terminar): procesar datos
Respuesta del servidor: Tarea 'procesar datos' enviada a la cola de procesamiento.
Ingrese una tarea (o 'salir' para terminar):
```



Cliente

- Se conecta al servidor por socket.
- Envía la tarea "procesar datos".
- Recibe la confirmación de que la tarea fue enviada a la cola RabbitMQ.

Servidor

- Acepta la conexión del cliente (('127.0.0.1', 63174)).
- Publica el mensaje "procesar datos" en la cola RabbitMQ.
- Devuelve la respuesta "Tarea 'procesar datos' enviada a la cola de procesamiento."

Worker

- Está suscrito a la cola "tareas".
- Recibe el mensaje "procesar datos".
- Procesa y confirma su finalización.

La arquitectura distribuida está corriendo con:

- **Cliente:** Envia tareas.
- **Servidor:** Las enruta a RabbitMQ.
- **RabbitMQ:** Gestiona la cola.
- **Worker:** Procesa las tareas.

Paso 6: Conectar a un almacenamiento simulado (PostgreSQL/S3) para guardar resultados.

PREPARAR SERVICIOS CON DOCKER (PostgreSQL + MinIO para S3)

PostgreSQL

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_
E Programación sobre redes\PFO3\PFO3-Redes> docker run -d --name pfo3-postgres -e POSTGRES_USER=p
ostgres -e POSTGRES_PASSWORD=postgres -e POSTGRES_DB=tasksdb -p 5432:5432 postgres:15
Unable to find image 'postgres:15' locally
15: Pulling from library/postgres
8bbe1487ab04: Pull complete
5a1b3bfd0b0b: Pull complete
7cf8703a60ae: Pull complete
4b34d7a7ab2a: Pull complete
0cdc9eb2a585: Pull complete
9be150812fa2: Pull complete
f0e812253d9a: Pull complete
b12e04c2e850: Pull complete
1c3c3432cfaa: Pull complete
38513bd72563: Pull complete
70a82bc39e86: Pull complete
d5997a3d3e9c: Pull complete
9d368c9c6ba6: Pull complete
bb86ed266eaf: Pull complete
Digest: sha256:424e79b81868f5fc5cf515eaeac69d288692ebcca7db86d98f91b50d4bce64bb
Status: Downloaded newer image for postgres:15
d6f2146294031de3ab97024c3da6c17eaf160c4f9a3177419cf0d4061cd911d7
```

MinIO (S3 compatible, con consola web en 9001)

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_
E Programación sobre redes\PFO3\PFO3-Redes> docker run -d --name pfo3-minio -p 9000:9000 -p 9001:
9001 -e MINIO_ROOT_USER=minioadmin -e MINIO_ROOT_PASSWORD=minioadmin quay.io/minio/minio:latest s
erver /data --console-address ":9001"
Unable to find image 'quay.io/minio/minio:latest' locally
latest: Pulling from minio/minio
f94d28849fa3: Pull complete
b83ce1c86227: Pull complete
34013573f278: Pull complete
81260b173076: Pull complete
71e9fc939447: Pull complete
c1bc68842c41: Pull complete
0288b5a0d7e7: Pull complete
1008deaf6ec4: Pull complete
f9c0805c25ee: Pull complete
Digest: sha256:14cea493d9a34af32f524e538b8346cf79f3321eff8e708c1e2960462bd8936e
Status: Downloaded newer image for quay.io/minio/minio:latest
fea79aa8064e505430d505ca08b4bd4e683c59ee8259c1161365e384446a1385
```

MinIO: consola web en http://localhost:9001

Credenciales por defecto del ejemplo: minioadmin / minioadmin

Tenemos los 3 contenedores en Docker: pfo3-postgres, pfo-minio y rabbitmq

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1_E Programación sobre redes\PFO3\PFO3-Redes> docker ps
CONTAINER ID   IMAGE                        COMMAND                  CREATED              STATUS              PORTS
      NAMES
fea79aa8064e   quay.io/minio/minio:latest   "/usr/bin/docker-ent…"   About a minute ago   Up About a minute   0.0.0.0:9000-9001->9000-9001/tcp, [::]:9000-9001->9000-9001/tcp
      pfo3-minio
d6f214629403   postgres:15                  "docker-entrypoint.s…"   3 minutes ago        Up 3 minutes        0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp
      pfo3-postgres
ffef0c7b7708   rabbitmq:3-management        "docker-entrypoint.s…"   4 hours ago          Up 4 hours          0.0.0.0:5672->5672/tcp, [::]:5672->5672/tcp, 0.0.0.0:15672->15672/tcp, [::]:15672->15672/
tcp   rabbitmq
```

CREAR TABLA EN POSTGRESQL (usando Docker y psql)

Entrar al contenedor de PostgreSQL

Ingresar a la base de datos con psql

```
abril_0edm3um@Marysol MINGW64 ~/OneDrive/Desktop/DESARROLLO DE SOFTWARE IFTS N°29/5SegundoCuatri
2025/1_E Programación sobre redes/PFO3/PFO3-Redes
$ docker exec -it pfo3-postgres bash
root@d6f214629403:/# psql -U postgres -d tasksdb
psql (15.14 (Debian 15.14-1.pgdg13+1))
Type "help" for help.

tasksdb=#
```

Crear tabla de resultados

```
tasksdb=# CREATE TABLE IF NOT EXISTS tareas_resultados (
    id SERIAL PRIMARY KEY,
    tarea_texto TEXT NOT NULL,
    resultado_texto TEXT NOT NULL,
    archivo_s3_key TEXT,
    procesado_en TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE
```

Verificar que la tabla se creó correctamente

```
tasksdb=# \dt
              List of relations
 Schema |       Name        | Type  |  Owner
--------+-------------------+-------+----------
 public | tareas_resultados | table | postgres
(1 row)
```

Salir del cliente y del contenedor

```
tasksdb=# \q
root@d6f214629403:/# exit
exit
```

Se creó la tabla tareas_resultados dentro de la base de datos tasksdb para almacenar los resultados procesados por los workers. Se utilizó el cliente psql dentro del contenedor Docker de PostgreSQL.

Conectar el Worker con PostgreSQL (persistencia de resultados)

Creamos **worker_rabbitmq_db.py** : reemplaza al worker_rabbitmq.py, pero además de procesar la tarea, guarda el resultado en la base PostgreSQL.

Instalar el conector de PostgreSQL

```
abril_0edm3um@Marysol MINGW64 ~/OneDrive/Desktop/DESARROLLO DE SOFTWARE IFTS N°29/5SegundoCuatri
2025/1_E Programación sobre redes/PFO3/PFO3-Redes
$ pip install psycopg2-binary
Collecting psycopg2-binary
  Downloading psycopg2_binary-2.9.11-cp313-cp313-win_amd64.whl.metadata (5.1 kB)
Downloading psycopg2_binary-2.9.11-cp313-cp313-win_amd64.whl (2.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.7/2.7 MB 11.6 MB/s eta 0:00:00
Installing collected packages: psycopg2-binary
Successfully installed psycopg2-binary-2.9.11

[notice] A new release of pip is available: 24.2 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Iniciar los 3 componentes en terminales separadas:

Terminal 1 – Worker

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1
_E Programación sobre redes\PFO3\PFO3-Redes> python worker_rabbitmq_db.py
[WORKER] Esperando tareas...
```

Terminal 2 – Servidor

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1
_E Programación sobre redes\PFO3\PFO3-Redes> python servidor_rabbit.py
>>
[SERVIDOR RABBITMQ INICIADO...]
[ESCUCHANDO] Servidor en 127.0.0.1:5000
[POOL ACTIVO] Capacidad: 3 workers
```

Terminal 3 - Cliente

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1
_E Programación sobre redes\PFO3\PFO3-Redes> python cliente.py
Conectado al servidor en 127.0.0.1:5000
Ingrese una tarea (o 'salir' para terminar): procesar datos de usuario
Respuesta del servidor: Tarea 'procesar datos de usuario' enviada a la cola de procesamiento.
Ingrese una tarea (o 'salir' para terminar):
```

Verificación final en Worker

```
PS C:\Users\abril_0edm3um\OneDrive\Desktop\DESARROLLO DE SOFTWARE IFTS N°29\5SegundoCuatri2025\1
_E Programación sobre redes\PFO3\PFO3-Redes> python worker_rabbitmq_db.py
[WORKER] Esperando tareas...
[WORKER] Tarea recibida: procesar datos
[DB] Resultado guardado en la base: procesar datos -> Resultado de 'procesar datos' procesado co
rrectamente.
[WORKER] Tarea completada y guardada en BD.
```

Consulta en PostgreSQL

docker exec -it pfo3-postgres bash
psql -U postgres -d tasksdb
SELECT * FROM tareas_resultados;

```
 id |  tarea_texto   |                  resultado_texto                   | archivo_s3_key |
       procesado_en
----+----------------+----------------------------------------------------+----------------+
----------------------------
  1 | procesar datos | Resultado de 'procesar datos' procesado correctamente. |                |
 2025-10-24 22:38:40.505643
(1 row)
```

Resumen

Componente	Rol	Estado
Cliente	Envía tareas por socket	Funciona
Servidor RabbitMQ	Recibe tareas d clientes, las envía a la cola de mensajes RabbitMQ	En ejecución
RabbitMQ	Cola de mensajes que conecta servidores y workers	Conectado
Worker con PostgreSQL	Recibe tareas de la cola, las procesa y guarda resultados en BD	Procesando
PostgreSQL	Guarda los resultados de las tareas procesadas	Guardando datos

Estructura Final del proyecto

```
PFO3_Redes/
 ├── servidor.py                 # Servidor básico con sockets
 ├── servidor_threadpool.py      # Servidor mejorado con ThreadPool
 ├── servidor_rabbit.py          # Servidor que publica tareas en RabbitMQ
 ├── cliente.py                  # Cliente que envía y recibe tareas
 ├── worker_rabbitmq.py          # Worker remoto que procesa tareas desde RabbitMQ
 ├── worker_rabbitmq_persist.py  # Worker que guarda resultados en PostgreSQL
 ├── README.md
 └── diagrama.png                # Diagrama de arquitectura
```