PFO 3 – Rediseño como Sistema Distribuido (Cliente–Servidor)

Materia: Programación sobre Redes

Tema: Programación Concurrente Año: 2025 - 2° cuatrimestre consigna y entrega:

Práctica Formativa Obligatoria (PFO) 3

Alumno:

-Coria Daniel

-Mazar María

Docente: Germán Ríos

Índice

1. Resumen / Abstract

Este trabajo presenta el rediseño de una aplicación monolítica a una arquitectura distribuida basada en sockets TCP y mensajería asíncrona con RabbitMQ. Se implementaron: un cliente que envía tareas, un servidor distribuidor que recibe y encola tareas, y múltiples workers que procesan y almacenan en un sistema distribuido simulado. El repositorio con el código se encuentra en: https://github.com/santysanty/Python_DistributedSystem_PFO3.

2. Objetivos

• Transformar el sistema a una arquitectura distribuida.
• Implementar comunicación cliente → distribuidor → workers usando sockets y cola de mensajes.
• Demostrar balanceo de trabajo y procesamiento asíncrono.
• Documentar el diseño y la forma de ejecución para reproducibilidad.

3. Alcance y supuestos

Alcance: Implementación funcional básica para demostrar la arquitectura (no un producto en producción).

Supuestos: RabbitMQ corre en localhost. Python 3.10+ disponible. En la entrega se incluye alternativa sin RabbitMQ usando sockets (si se requiere).

4. Arquitectura del sistema

4.1 Diagrama

Insertar /docs/diagrama_sistema_distribuido.png aquí (centrado). Leyenda: "Diagrama lógico del sistema: Clientes → Balanceador → Distribuidor → Cola (RabbitMQ) → Workers → Almacenamiento distribuido (PostgreSQL/S3)"

4.2 Componentes y roles

• Clientes: apps móviles / web que envían tareas via TCP.
• Balanceador de carga (opcional): Nginx / HAProxy para distribuir peticiones entre varios distribuidores.
• Servidor Distribuidor: escucha en puerto TCP (ej. 5000), recibe JSON y publica en cola tareas.
• Cola de Mensajes (RabbitMQ): desacopla productores y consumidores; garantiza persistencia y reintentos.
• Workers: consumen la cola, procesan y escriben en almacenamiento.
• Almacenamiento Distribuido: simulación de PostgreSQL y S3 (describir cómo simulas escrituras — logs, archivos, etc.).

5. Implementación (detalles técnicos)

5.1 Estructura del repositorio

```
Python_DistributedSystem_PFO3/
 ├cliente_tareas.py
 ├servidor_distribuidor.py
 ├servidor_worker.py
 ├README.md
 ├docs/
 |  └diagrama_sistema_distribuido.png
```

5.2 Resumen de archivos y fragmentos clave

cliente_tareas.py: genera ID (uuid), arma JSON y conecta por socket a localhost:5000. Envía N tareas y recibe confirmación del distribuidor.

servidor_distribuidor.py: escucha conexiones TCP, crea un hilo por cliente, parsea JSON y publica en RabbitMQ (o reenvía a workers por sockets con Round-Robin).

servidor_worker.py: consumer RabbitMQ con basic_qos(prefetch_count=1), procesa tarea (sleep simulado), ack. Alternativa sin RabbitMQ: worker TCP con ThreadPoolExecutor.

5.3 Dependencias

Python 3.10+

pip packages: pika

pip install pika

6. Instrucciones de ejecución (paso a paso)

Ejecutar en este orden (abrir terminales independientes):

- Iniciar RabbitMQ (Windows: Docker)
- Iniciar workers: python servidor_worker.py
- Iniciar distribuidor: python servidor_distribuidor.py
- Ejecutar cliente: python cliente_tareas.py

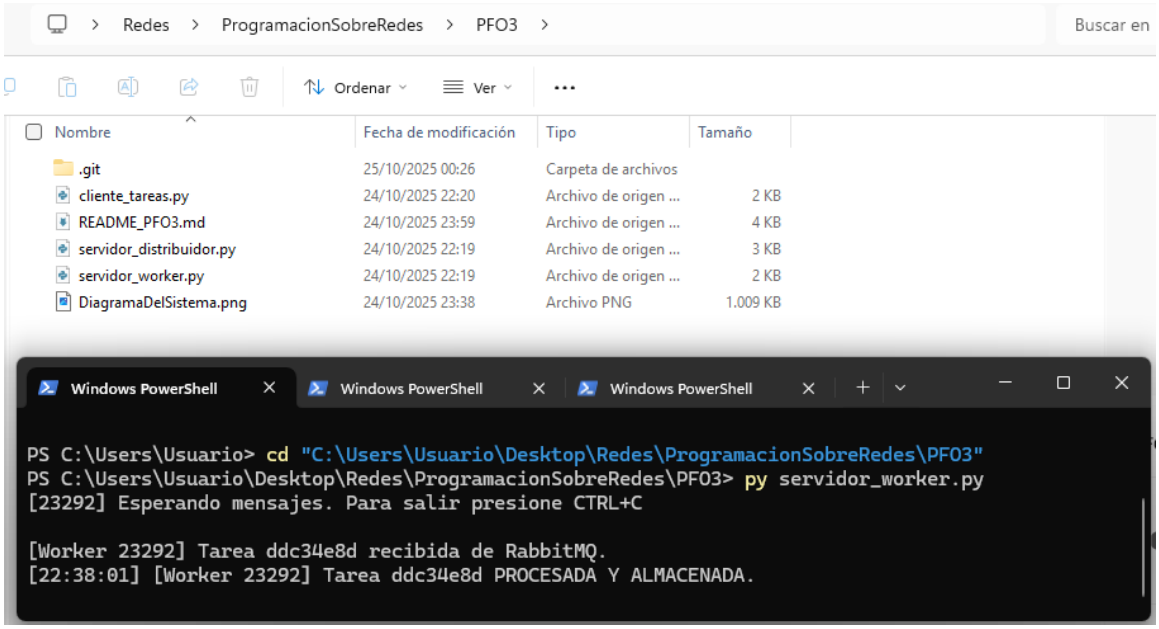
Alternativa sin RabbitMQ: servidor_worker_sin_pika.py, servidor_distribuidor_sin_pika.py, ejecutar cliente.

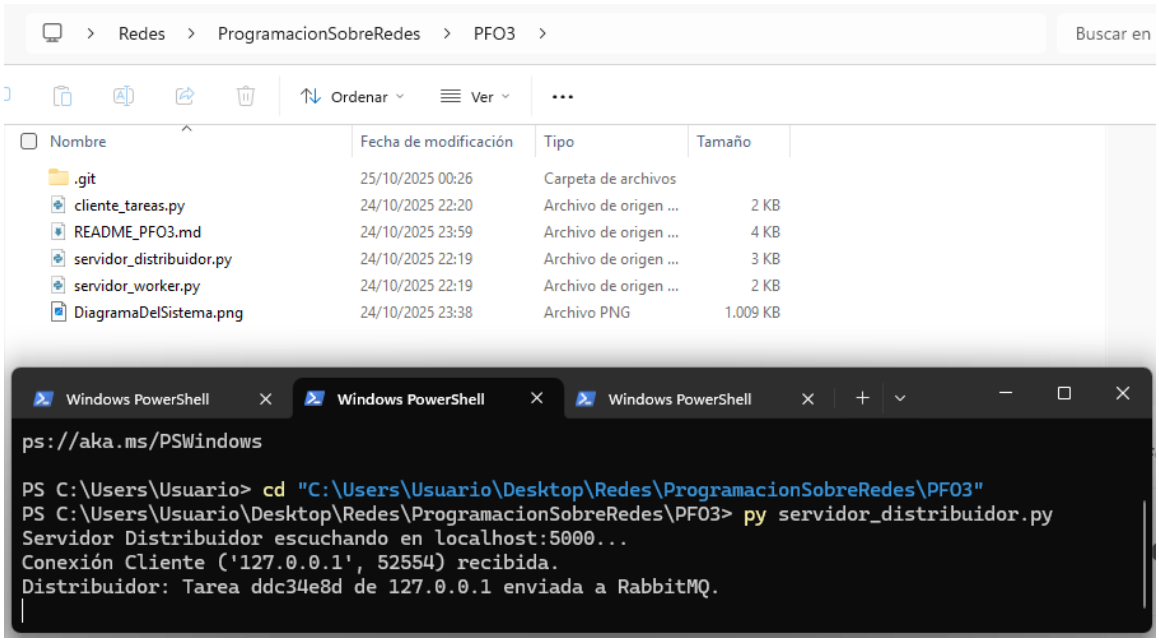
7. Pruebas realizadas y resultados

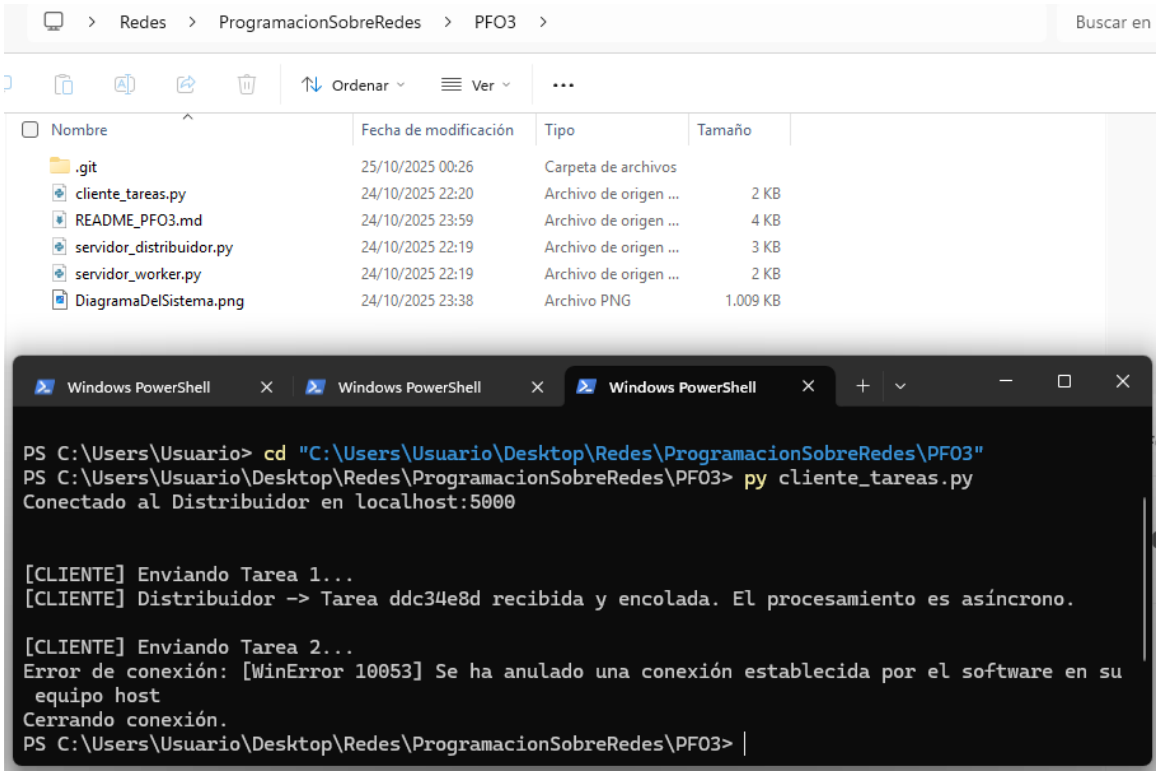
Prueba 1: 3 clientes secuenciales → confirmación inmediata del distribuidor → tasks encoladas.
Resultado: Cliente recibió confirmación; workers procesaron tareas (logs con PID y timestamps).

Prueba 2: Ejecutar 3 clientes en paralelo.
Resultado: Distribución round-robin o RabbitMQ automáticamente entre workers.

Evidencias:







8. Limitaciones y mejoras futuras

• No se implementó autenticación/seguridad TLS en sockets.
• Balanceador físico (Nginx/HAProxy) no desplegado.

• Persistencia real: conectar a PostgreSQL y S3.
• Métricas y monitoring (Prometheus / Grafana) para observabilidad.

9. Conclusión

Breve párrafo sobre lo aprendido y cómo la arquitectura cumple las consignas del PFO: desacople, escalabilidad y prueba de conceptos con sockets y mensajería.

10. Anexos

A. Código completo: ruta del repo en GitHub.
B. Diagrama: docs/diagrama_sistema_distribuido.png.
C. Comandos Git para la entrega.
D. Capturas de prueba: poner las imágenes en docs/ y enlazarlas.