Voy a implementar la Fase 1 del sistema asíncrono con Celery y Redis. Primero necesito revisar la arquitectura existente para entender los contratos de API.

[7 tools called]

Ahora implementaré el sistema asíncrono con Celery y Redis. Comenzaré con el archivo principal del Servicio de Clasificación:

[1 tool called]

Ahora creo el archivo de configuración de Celery:

[1 tool called]

Ahora creo el archivo de tareas de Celery:

[1 tool called]

Ahora actualizo el cliente de persistencia para incluir métodos síncronos:

[1 tool called]

Ahora actualizo los modelos para incluir el task\_id:

[1 tool called]

Ahora actualizo el requirements.txt para incluir Celery y Redis:

[1 tool called]

Ahora creo el archivo worker.py para ejecutar los workers de Celery:

[1 tool called]

Ahora creo el docker-compose.yml para toda la infraestructura:

[1 tool called]

Ahora actualizo el archivo .env para incluir las variables de Redis:

[1 tool called]

Ahora creo un archivo .env.example:

[1 tool called]

Ahora creo un archivo de comandos para ejecutar todo:

[1 tool called]

Finalmente, actualizo el Dockerfile del servicio de clasificación para incluir curl:

[1 tool called]

## \*\*IMPLEMENTACIÓN COMPLETA DE LA FASE 1\*\*

He implementado completamente el sistema asíncrono y escalable usando Celery y Redis. Aquí está el resumen de lo que se ha creado:

### \*\*�� ARCHIVOS IMPLEMENTADOS:\*\*

1. \*\*`services/classification\_service/app/main.py`\*\* - FastAPI con endpoints asíncronos

2. \*\*`services/classification\_service/app/celery\_app.py`\*\* - Configuración de Celery con Redis

3. \*\*`services/classification\_service/app/tasks.py`\*\* - Tareas de Celery para clasificación

4. \*\*`services/classification\_service/app/clients/persistence\_client.py`\*\* - Cliente con métodos síncronos/asíncronos

5. \*\*`services/classification\_service/app/models.py`\*\* - Modelos actualizados con task\_id

6. \*\*`services/classification\_service/requirements.txt`\*\* - Dependencias incluyendo Celery y Redis

7. \*\*`services/classification\_service/worker.py`\*\* - Script para ejecutar workers

8. \*\*`docker-compose.yml`\*\* - Infraestructura completa con Redis, workers y servicios

9. \*\*`comandos\_ejecucion.md`\*\* - Guía completa de comandos

### \*\*🚀 CARACTERÍSTICAS IMPLEMENTADAS:\*\*

- \*\*Sistema asíncrono\*\*: El endpoint `/classify` responde inmediatamente con un `task\_id`

- \*\*Escalabilidad\*\*: Workers de Celery que se pueden escalar horizontalmente

- \*\*Redis como broker\*\*: Cola de mensajes robusta y rápida

- \*\*Monitoreo\*\*: Flower para monitorear tareas de Celery

- \*\*Health checks\*\*: Verificación de estado de todos los servicios

- \*\*Reintentos\*\*: Lógica de reintentos con tenacity

- \*\*Logging\*\*: Sistema de logs completo para debugging

### \*\*�� COMANDOS PARA EJECUTAR:\*\*

```bash

# 1. Crear archivo .env con las variables de Redis

# 2. Instalar dependencias: pip install -r requirements.txt

# 3. Ejecutar con Docker: docker-compose up -d

# 4. O manualmente: redis-server + python worker.py + uvicorn

```

### \*\*📊 ENDPOINTS NUEVOS:\*\*

- `POST /classify/{document\_id}` - Encola tarea de clasificación

- `GET /task/{task\_id}/status` - Estado de una tarea

- `GET /document/{document\_id}/progress` - Progreso de clasificación

El sistema mantiene el contrato de APIs existente y respeta el orden original de las filas, implementando exactamente lo que solicitaste en la Fase 1.