Manual de Instalación Sistema JusticIA

PODER JUDICIAL DE COSTA RICA

Tabla de contenido

1. Introducción

El presente documento constituye el Manual de Instalación oficial para el sistema JusticIA, plataforma de Asistencia Judicial con Inteligencia Artificial.

El objetivo de esta guía es proporcionar los procedimientos detallados y secuenciales necesarios para el despliegue exitoso de la aplicación en los entornos de Desarrollo y Producción. Se dirige principalmente a los equipos de infraestructura y personal técnico encargados de la instalación, configuración y mantenimiento del sistema.

Dado que JusticIA está diseñado sobre una arquitectura de microservicios contenerizados (Docker), este manual cubre los requisitos de hardware y software, la configuración de servicios externos críticos (SQL Server, Milvus, Ollama), y los pasos exactos para la puesta en marcha de los componentes Backend y Frontend. Seguir estas instrucciones es fundamental para garantizar el funcionamiento y la estabilidad del sistema.

2. Requisitos del sistema

2.1. Hardware requerido

Requisitos base de hardware

Componente	Desarrollo (probado)	Producción (probado)
CPU	8 núcleos	8 núcleos
RAM	8 GB	32 GB
Disco	100 GB SSD	120 GB SSD
GPU	No disponible	No disponible

Límites de memoria y recursos

El sistema fue desarrollado y probado utilizando recursos de hardware limitados (ver tabla superior). Por esta razón, los valores de memoria configurados en los archivos docker-compose.yml y docker-compose.prod.yml representan los máximos límites probados con dichos recursos. Para servidores con diferente capacidad, se recomienda ajustar los límites de memoria.

```
deploy:
  resources:
    limits:
      memory: 6G # Modificar según servidor
```

Nota: Aunque el sistema fue probado inicialmente sin requerir GPU, se recomienda encarecidamente incluir una GPU dedicada en el entorno de Producción.

GPU Recomendada (Producción):

- NVIDIA con soporte CUDA
- VRAM: 8 GB mínimo (12 GB recomendado)
- Para modelos Whisper large y LLMs locales

La GPU es crítica para:

1. Extracción de audio (Whisper): Acelera el procesamiento y permite usar modelos más grandes para obtener una mayor precisión en las transcripciones.
2. Modelos LLM locales (Ollama): Si se opta por instalar un servidor Ollama propio (Local) en lugar de un servicio *cloud*, la GPU es esencial para mejorar el rendimiento y garantizar una velocidad de respuesta viable en las consultas conversacionales.

2.2.Software requerido

Sistema operativo:

- Linux: Ubuntu 22.04 LTS
- Windows: Windows 11 con WSL2

Software Base:

```
# Docker y Docker Compose
docker --version  # Mínimo: 24.0+
docker compose version  # Mínimo: 2.20+
# Git
git --version  # Mínimo: 2.30+
```

Instalación Docker (Ubuntu 22.04):

```
# Actualizar sistema
sudo apt update && sudo apt upgrade -y

# Instalar Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh

# Agregar usuario al grupo docker
sudo usermod -aG docker $USER
newgrp docker

# Verificar
docker --version
docker compose version
```

Instalación Docker Desktop (Windows):

1. Descargar desde: https://www.docker.com/products/docker-desktop
2. Instalar y reiniciar sistema
3. Configurar recursos mínimos:

- **Desarrollo:** 8GB RAM, 8 CPU
- **Producción:** 32 GB RAM, 8 CPU

3. Servicios externos necesarios

El sistema JusticIA opera bajo una arquitectura desacoplada, lo que significa que depende de servicios de infraestructura críticos que deben estar previamente instalados, configurados y operativos en la red.

Principio de desacoplamiento:

- El sistema JusticIA está diseñado para enfocarse exclusivamente en su lógica de negocio y procesamiento de datos.
- Por lo tanto, los servicios de almacenamiento persistente y procesamiento intensivo (como la base de datos y el servidor vectorial) no se instalan localmente dentro del *docker-compose* de JusticIA.
- JusticIA solo se conecta a estos servicios a través de las credenciales y URLs especificadas en el archivo *backend/.env*.

Componentes de infraestructura críticos

El despliegue exitoso de JusticIA requiere la disponibilidad de los siguientes tres servicios externos:

3.1. SQL Server (Base de datos relacional)

Este es el servicio de almacenamiento relacional principal del sistema.

- Función: Almacena toda la metadata del sistema, la información de usuarios, roles, permisos y la estructura de los documentos legales.

- Requisito: Debe estar instalado (ya sea en Azure o Local/On-Premise) y la base de datos debe estar creada antes de la instalación de JusticIA.

3.1.1. Opción A: Azure SQL Server (Cloud)

Requisitos:

- Servidor SQL Server en Azure configurado
- Base de datos creada
- Firewall configurado para permitir IP del servidor de JusticIA

En backend/.env:

```
SQL_SERVER_HOST=servidor.database.windows.net
SQL_SERVER_PORT=1433
SQL_SERVER_DATABASE=db_JusticIA
SQL_SERVER_USER=usuario
SQL_SERVER_PASSWORD=password
SQL_SERVER_DRIVER=ODBC Driver 18 for SQL Server
```

3.1.2. Opción B: SQL Server Local/On-Premise

Requisitos:

- Servidor SQL Server ya instalado y configurado (por equipo de infraestructura)
- Base de datos creada
- Acceso de red habilitado (puerto 1433)

En backend/.env:

```
SQL_SERVER_HOST=IP_SERVIDOR_SQL
SQL_SERVER_PORT=1433
SQL_SERVER_DATABASE=db_JusticIA
```

```
SQL_SERVER_USER=usuario
SQL_SERVER_PASSWORD=password
SQL_SERVER_DRIVER=ODBC Driver 18 for SQL Server
```

Ejemplo para servidor local:

```
# Servidor en la misma máquina
SQL_SERVER_HOST=localhost

# Servidor en red local
SQL_SERVER_HOST=192.168.1.100

# Servidor con nombre de dominio
SQL_SERVER_HOST=sqlserver.empresa.local
```

Nota: Si el servidor SQL usa certificados autofirmados, puede ser necesario agregar:

SQL_SERVER_TRUST_CERT=yes

3.2. Milvus (Base de datos vectorial)

Este servicio es fundamental para la funcionalidad de búsqueda semántica basada en Inteligencia Artificial.

- Función: Almacena los *embeddings* (representaciones vectoriales) de los documentos legales, permitiendo búsquedas contextuales y por similitud.
- Requisito: Se conecta como un servicio externo (similar a SQL Server). Se recomienda encarecidamente que Milvus se ejecute en un servidor separado o se gestione a través de un servicio *cloud* para garantizar el rendimiento y control de datos sensibles.

3.2.1. Opción A: Zilliz Cloud (Servicio administrado)

Configuración:

1. Crear cuenta en: https://cloud.zilliz.com/
2. Crear cluster serverless (región cercana)
3. Obtener credenciales del dashboard

En backend/.env:

```
MILVUS_URI=https://xxx.serverless.aws-region.cloud.zilliz.com
MILVUS_TOKEN=token_generado_zilliz
MILVUS_DB_NAME=JusticIA
COLLECTION_NAME=justicia_docs
```

3.2.2. Opción B: Servidor Milvus propio (recomendado para Producción)

Requisitos del servidor Milvus:

- **RAM:** 8 GB mínimo (16 GB recomendado)
- **CPU:** 4 cores mínimo (8 cores recomendado)
- **Disco:** 100 GB SSD mínimo
- **Sistema:** Linux

Nota: Milvus se instala en un servidor separado o en el mismo servidor, pero no forma parte del docker-compose de JusticIA. JusticIA solo se conecta a Milvus mediante la URL.

Configuración en JusticIA:

Si ya se tiene un servidor Milvus corriendo (instalado previamente por el equipo de infraestructura), simplemente configurar la conexión en backend/.env:

```
# Si Milvus está en el MISMO servidor que JusticIA:
MILVUS_URI=http://localhost:19530
MILVUS_TOKEN=

# Si Milvus está en OTRO servidor de la red:
```

```
MILVUS_URI=http://IP_SERVIDOR_MILVUS:19530
MILVUS_TOKEN=

# Variables comunes
MILVUS_DB_NAME=JusticIA
COLLECTION_NAME=justicia_docs
```

Una vez ejecutado los pasos anteriores JusticIA se conectará automáticamente al servidor Milvus existente

Instalación de Milvus (Opcional - Solo si no existe servidor):

Si el equipo de infraestructura necesita instalar Milvus desde cero en un servidor dedicado, puede usar:

```
# En el servidor DEDICADO para Milvus (no en el servidor de JusticIA)

# Verificar última versión estable en: https://milvus.io/docs/release_notes.md

# Ejemplo con v2.3.3:

curl -O https://github.com/milvus-io/milvus/releases/download/v2.3.3/milvus-standalone-docker-compose.yml
docker compose -f milvus-standalone-docker-compose.yml up -d

# Verificar
curl http://localhost:19530/healthz
```

Documentación oficial: https://milvus.io/docs/install_standalone-docker.md

Comparación Zilliz Cloud vs Servidor Propio:

Aspecto	Zilliz Cloud	Servidor Propio
Costo	Mensual por uso	Solo hardware (una vez)
Latencia	20-100ms	<5ms (red local)
Seguridad	Datos en cloud	Datos en servidor propio
Mantenimiento	Automático	Manual
Escalabilidad	Elástica	Manual
Ideal para	Desarrollo	Producción con capacidad

Recomendación:

Desarrollo/Pruebas: Zilliz Cloud (rápido de configurar).

Producción con servidor propio: Milvus Standalone (mejor rendimiento, sin costos).

3.3. Ollama (Servidor LLM)

Este componente gestiona los Modelos de Lenguaje (LLM) que impulsan las capacidades de consulta conversacional y análisis avanzado del sistema.

- Función: Procesa las consultas de lenguaje natural del usuario, generando respuestas, resúmenes o análisis basados en los documentos vectoriales.
- Opciones: Puede conectarse a un servicio Ollama *Cloud (o similar)* o a una instancia Ollama *Local* (gestionada por infraestructura). La elección dependerá de los requisitos de latencia, seguridad y el uso de GPU.

Nota: Ollama funciona igual que SQL Server y Milvus: es un servicio externo que debe estar previamente instalado. JusticIA solo se conecta mediante la configuración en backend/.env.

3.3.1. Opcion A: Ollama Cloud:

Configuración en .env:

```
OLLAMA_BASE_URL=https://ollama.com
OLLAMA_MODEL=gpt-oss:20b-cloud
OLLAMA_API_KEY=api_key_generada
```

3.3.2. Opcion B: Ollama Local:

```
# Instalar Ollama
curl -fsSL https://ollama.com/install.sh | sh

# Descargar modelo
ollama pull gpt-oss:20b # gpt-oss:120b

# Iniciar servidor
ollama serve
```

Configuración en .env:

```
OLLAMA_BASE_URL=http://localhost:11434
OLLAMA_MODEL=gpt-oss:20b
OLLAMA_API_KEY=
```

4. Instalación del Backend

4.1. Entorno de desarrollo

4.1.1. Clonar repositorio

```
git clone https://github.com/procmc/JusticIA.git
cd JusticIA
```

4.1.2. Configurar variables de entorno

```
cd backend
```

```
# Copiar archivo de ejemplo
cp .env.example .env

# Editar con credenciales reales
nano .env
```

Variables críticas que modificar:

```
# SQL Server
SQL_SERVER_HOST=servidor.database.windows.net
SQL_SERVER_USER=usuario_real
SQL_SERVER_PASSWORD=password_real

# Milvus
MILVUS_URI=https://xxx.cloud.zilliz.com
MILVUS_TOKEN=token_real

# Ollama
OLLAMA_API_KEY=api_key_real

# JWT (generar nuevo)
SECRET_KEY=resultado_de_openssl_rand_hex_32

# Email
EMAIL_USERNAME=email@gmail.com
EMAIL_PASSWORD=app_password_generado
```

Generar SECRET_KEY:

```
openssl rand -hex 32
```

4.1.3. Iniciar servicios del Backend

```
# Volver a raíz del proyecto
cd ..

# Construir imágenes
docker compose build backend celery-worker tika

# Iniciar servicios
docker compose up -d backend celery-worker tika redis

# Ver logs
docker compose logs -f backend
```

Nota: No es necesario especificar redis y tika explícitamente, ya que se iniciarán automáti camente como dependencias de backend.

Comando simplificado:

```
docker compose up -d backend celery-worker
```

4.1.4. Ejecutar migraciones

```
# Primera vez: crear tablas
docker compose exec backend alembic upgrade head

# Verificar tablas creadas
docker compose exec backend python -c "
from app.db.database import engine
from sqlalchemy import inspect
inspector = inspect(engine)
print('Tablas creadas:', inspector.get_table_names())
"
```

4.1.5. Verificar estado del Backend

```
# Health check
curl http://localhost:8000/health

# Respuesta esperada:
# {"status":"healthy","database":"connected","redis":"connected"}

# Documentación API
# Abrir navegador: http://localhost:8000/docs
```

4.2. Entorno de Producción

4.2.1. Preparar variables de entorno

```
cd backend

# Copiar y editar
cp .env.example .env
nano .env
```

Cambios obligatorios para producción:

```
# Cambiar entorno
ENVIRONMENT=production

# Generar nuevas claves
SECRET_KEY=clave_nueva_generada_con_openssl

# Credenciales de producción
SQL_SERVER_HOST=servidor_produccion.database.windows.net
SQL_SERVER_PASSWORD=password_fuerte_produccion

# Usuario admin con contraseña fuerte
```

```
ADMIN_PASSWORD=PasswordFuerte2025!
ADMIN_EMAIL=admin@dominio.com

# Timeouts optimizados
LLM_REQUEST_TIMEOUT=300
```

4.2.2. Ajustar recursos (opcional)

Si el servidor tiene diferente RAM, editar docker-compose.prod.yml:

```
backend:
  deploy:
    resources:
      limits:
        memory: 6G  # Modificar según servidor

celery-worker:
  deploy:
    resources:
      limits:
        memory: 8G  # Modificar según servidor
```

4.2.3. Desplegar el Backend

```
# Volver a raíz
cd ..

# Construir imágenes de producción
docker compose -f docker-compose.prod.yml build

# Iniciar servicios
docker compose -f docker-compose.prod.yml up -d
```

```
# Ver logs
docker compose -f docker-compose.prod.yml logs -f backend
```

Nota: El archivo docker-compose.prod.yml **solo contiene los servicios del backend** (backend, celery-worker, tika, redis). El frontend se despliega de forma independiente.

4.2.4. Ejecutar migraciones

```
# Crear tablas
docker compose -f docker-compose.prod.yml exec backend alembic up
grade head

# Verificar
docker compose -f docker-compose.prod.yml exec backend python -c
"
from app.db.database import engine
print('Conexión a base de datos OK')
"
```

4.2.5. Verificar el estado del Backend

```
# Health check
curl http://localhost:8000/health

# Ver estado de contenedores
docker compose -f docker-compose.prod.yml ps

# Monitorear recursos
docker stats
```

5. Instalación del Frontend

5.1. Entorno de Desarrollo

5.1.1. Configurar variables de entorno

```
cd frontend

# Copiar archivo de ejemplo
cp .env.example .env.local

# Editar
nano .env.local
```

Configuración desarrollo:

```
# Backend local
NEXT_PUBLIC_API_URL=http://localhost:8000

# Generar NEXTAUTH_SECRET
NEXTAUTH_SECRET=resultado_de_openssl_rand_base64_32

# URL local
NEXTAUTH_URL=http://localhost:3000
```

Generar NEXTAUTH_SECRET:

```
openssl rand -base64 32
```

5.1.2. Opción A: Ejecutar con Docker

```
# Volver a la raíz del proyecto
cd ..

# Construir imagen
```

```
docker compose build frontend

# Iniciar frontend
docker compose up -d frontend

# Ver logs
docker compose logs -f frontend
```

Acceder: http://localhost:3000

5.1.3. Opción B: Ejecutar sin Docker

```
cd frontend

# Instalar dependencias
npm install

# Modo desarrollo
npm run dev
```

Acceder: http://localhost:3000

5.2. Entorno de Producción

5.2.1. Configurar variables de entorno

```
cd frontend

# Copiar archivo de ejemplo
cp .env.example .env.local

# Editar
nano .env.local
```

Configuración produccion:

```
# Backend producción
NEXT_PUBLIC_API_URL=http://IP_SERVIDOR:8000 (o https://api.domini
o.com)
# Generar NEXTAUTH_SECRET
NEXTAUTH_SECRET=resultado_de_openssl_rand_base64_32

# URL local
NEXTAUTH_URL=http://IP_SERVIDOR:3000 (o https://dominio.com)
```

Generar NEXTAUTH_SECRET:

```
openssl rand -base64 32
```

5.2.2. Opción A: Ejecutar con Docker (recomendado)

```
# Volver a raíz
cd ..

# Construir imagen optimizada de frontend
docker build -f frontend/Dockerfile.prod -t justicia-frontend:pro
d ./frontend

# Ejecutar contenedor
docker run -d \
  --name justicia-frontend \
  --restart always \
  -p 3000:3000 \
  --env-file frontend/.env.local \
  justicia-frontend:prod

# Ver logs
docker logs -f justicia-frontend
```

5.2.3. Opción B: Ejecutar sin Docker (Node.js directo)

```
cd frontend

# Instalar dependencias de producción
npm ci --only=production

# Build de producción
npm run build

# Iniciar servidor
npm start
```

5.2.4. Verificar el estado del Frontend en producción

```
# Health check
curl http://localhost:3000

# Ver estado del contenedor
docker ps | grep justicia-frontend

# Ver logs
docker logs justicia-frontend

# Monitorear recursos
docker stats justicia-frontend
```

6. Comandos Útiles

6.1. Entorno de Desarrollo

```
# Iniciar servicios
docker compose up -d
```

```
# Ver logs
docker compose logs -f backend

# Reiniciar servicio
docker compose restart backend

# Detener todo
docker compose down

# Reconstruir después de cambios
docker compose build backend
docker compose up -d backend
```

6.2. Entorno de Producción

```
# Backend
docker compose -f docker-compose.prod.yml up -d
docker compose -f docker-compose.prod.yml logs -f backend
docker compose -f docker-compose.prod.yml restart backend

# Frontend
docker restart justicia-frontend
docker logs -f justicia-frontend

# Actualizar código
git pull origin main
docker compose -f docker-compose.prod.yml build
docker compose -f docker-compose.prod.yml up -d
```

6.3. Migraciones

```
# Desarrollo
docker compose exec backend alembic upgrade head
```

```
# Producción
docker compose -f docker-compose.prod.yml exec backend alembic up
grade head
```

7. Inicio Rápido

7.1. Entorno de Desarrollo

```
# 1. Clonar y configurar
git clone https://github.com/procmc/JusticIA.git
cd JusticIA/backend
cp .env.example .env
# Editar .env con credenciales

# 2. Backend
cd ..
docker compose up -d backend celery-worker tika redis
docker compose exec backend alembic upgrade head

# 3. Frontend
cd frontend
cp .env.example .env.local


# Editar .env.local

- NEXT_PUBLIC_API_URL=http://localhost:8000

- Generar NEXTAUTH_SECRET con: openssl rand -base64 32

cd ..
docker compose up -d frontend
```

```
# 4. Verificar
curl http://localhost:8000/health
curl http://localhost:3000
```

7.2. Entorno de Producción

```
# 1. Configurar variables
cd backend
cp .env.example .env
# Editar .env (ENVIRONMENT=production, SECRET_KEY, etc.)

cd ../frontend
cp .env.example .env.local
# Editar .env.local (URLs producción)

# 2. Backend
cd ..
docker compose -f docker-compose.prod.yml build
docker compose -f docker-compose.prod.yml up -d
docker compose -f docker-compose.prod.yml exec backend alembic up
grade head

# 3. Frontend
docker build -f frontend/Dockerfile.prod -t justicia-frontend:pro
d ./frontend
docker run -d --name justicia-frontend --restart always -p 3000:3
000 \
  --env-file frontend/.env.local justicia-frontend:prod

# 4. Verificar
docker compose -f docker-compose.prod.yml ps
docker stats
```

8. Nota Importante sobre Comandos

El presente manual proporciona comandos de instalación enfocados en entornos **Linux/Ubuntu** o **macOS**. Si usted está utilizando **Windows** (especialmente en entornos WSL2 o PowerShell), tenga en cuenta las siguientes equivalencias para los comandos básicos:

Equivalencia de Comandos Base

Tarea	Comando en Linux/macOS	Equivalente en Windows (PowerShell)
Copiar archivo	cp file1 file2	Copy-Item file1 file2
Editar archivo	nano file	notepad file
Llamada simple (GET)	curl URL	Invoke-WebRequest -Uri URL
Descargar archivo	curl -O URL	Invoke-WebRequest -Uri URL -OutFile filename

Herramienta Crítica: OpenSSL

Las instalaciones del Backend requieren la generación de claves de seguridad (SECRET_KEY, NEXTAUTH_SECRET) mediante openssl.

Para usuarios de Windows:

La utilidad OpenSSL no viene instalada por defecto. Debe descargar e instalar la versión binaria desde el siguiente enlace oficial antes de ejecutar los comandos de generación de claves:

Descarga de OpenSSL: https://slproweb.com/products/Win32OpenSSL.html