# Proceso para la creación de un RAG desde cero en contenedor Docker

## 1. Introducción

En este documento se describe el proceso detallado para crear un sistema de Retrieval-Augmented Generation (RAG) desde cero, utilizando Docker para contenerizar las aplicaciones necesarias. El proyecto se clonará desde un repositorio en GitHub haciendo uso de PowerShell y Git. Además, se explicará cómo crear las credenciales necesarias para el correcto funcionamiento del RAG.

## 2. Prerrequisitos

- Tener instalado Docker y Docker Compose.  
- Tener instalado Git en la máquina local.  
- Tener acceso a PowerShell (Windows) o una terminal compatible con comandos de PowerShell (Linux/Mac).  
- Contar con una cuenta de GitHub y acceso al repositorio que contiene el proyecto RAG.  
- Conexión a internet para clonar el repositorio y descargar imágenes Docker.

## 3. Clonado del repositorio usando PowerShell y Git

1. Abre PowerShell (o terminal con soporte PowerShell).  
2. Navega al directorio en el que deseas clonar el proyecto:  
 ```powershell  
 cd C:\ruta\a\tu\directorio  
 ```  
3. Clona el repositorio de GitHub:  
 ```powershell  
 git clone https://github.com/tu-usuario/tu-repositorio-rag.git  
 ```  
4. Accede al directorio del proyecto:  
 ```powershell  
 cd tu-repositorio-rag  
 ```

## 4. Estructura del proyecto

Una vez clonado el repositorio, la estructura típica del proyecto RAG podría ser la siguiente:  
- \*\*Dockerfile\*\*: define la imagen para la aplicación principal (por ejemplo, un servicio Python que gestione consultas RAG).  
- \*\*docker-compose.yml\*\*: orquesta los servicios necesarios (base de datos, vector store, modelo LLM, aplicación web, etc.).  
- \*\*app/\*\*: directorio que contiene el código fuente de la aplicación (scripts Python, configuración, etc.).  
- \*\*credentials/\*\*: carpeta donde se almacenarán archivos de credenciales o archivos `.env`.  
- \*\*README.md\*\*: descripción general del proyecto y comandos básicos.

## 5. Creación de credenciales

Para que el RAG funcione correctamente, se necesitarán credenciales que permitan el acceso a los servicios involucrados (por ejemplo, base de datos y API de modelo). A continuación, se describe cómo crear un archivo de variables de entorno.  
  
1. Crea una carpeta llamada `credentials` en la raíz del proyecto:  
 ```powershell  
 mkdir credentials  
 ```  
2. Dentro de `credentials`, crea un archivo `.env` con el siguiente contenido:  
 ```env  
 # Variables para la base de datos PostgreSQL  
 POSTGRES\_USER=usuario\_rag  
 POSTGRES\_PASSWORD=contraseña\_segura  
 POSTGRES\_DB=db\_rag  
  
 # Variables para el almacén de vectores (por ejemplo, Qdrant)  
 QDRANT\_HOST=qdrant  
 QDRANT\_PORT=6333  
  
 # Variables para la API del modelo de lenguaje (por ejemplo, Ollama)  
 OLLAMA\_API\_URL=http://ollama:11434  
 OLLAMA\_MODEL=llama3.1:latest  
 ```  
3. Asegúrate de no subir el archivo `.env` a repositorios públicos añadiéndolo al `.gitignore`:  
 ```gitignore  
 /credentials/.env  
 ```

## 6. Configuración del contenedor Docker

Se utilizará `docker-compose` para definir y levantar los servicios necesarios.  
Crea un archivo `docker-compose.yml` en la raíz del proyecto con el siguiente contenido de ejemplo:  
  
```yaml  
version: "3.8"  
services:  
 postgres:  
 image: postgres:15  
 container\_name: rag\_postgres  
 env\_file:  
 - credentials/.env  
 volumes:  
 - postgres\_data:/var/lib/postgresql/data  
 ports:  
 - "5432:5432"  
  
 qdrant:  
 image: qdrant/qdrant:v1.5.4  
 container\_name: rag\_qdrant  
 ports:  
 - "6333:6333"  
 volumes:  
 - qdrant\_data:/qdrant/storage  
  
 ollama:  
 image: ollama/ollama:latest  
 container\_name: rag\_ollama  
 ports:  
 - "11434:11434"  
  
 app:  
 build:  
 context: .  
 dockerfile: Dockerfile  
 container\_name: rag\_app  
 depends\_on:  
 - postgres  
 - qdrant  
 - ollama  
 env\_file:  
 - credentials/.env  
 volumes:  
 - ./:/app  
 ports:  
 - "8000:8000"  
  
volumes:  
 postgres\_data:  
 qdrant\_data:  
```

El \*\*Dockerfile\*\* para el servicio `app` podría ser:  
  
```dockerfile  
FROM python:3.10-slim  
WORKDIR /app  
COPY app/requirements.txt .  
RUN pip install --no-cache-dir -r requirements.txt  
COPY app/ .  
CMD [ "python", "main.py" ]  
```

## 7. Levantar los servicios y probar el RAG

Para construir y levantar los contenedores definidos, ejecuta el siguiente comando en PowerShell dentro del directorio del proyecto:  
```powershell  
docker-compose up --build -d  
```  
Esto hará lo siguiente:  
- Descargar las imágenes de Postgres, Qdrant y Ollama.  
- Construir la imagen del servicio `app`.  
- Levantar todos los contenedores en segundo plano.  
  
Luego puedes verificar que todos los contenedores estén en ejecución:  
```powershell  
docker-compose ps  
```  
Para revisar los logs del servicio `app`:  
```powershell  
docker-compose logs -f app  
```

## 8. Pruebas finales y verificación

1. Verifica que puedas conectarte a la base de datos PostgreSQL usando las credenciales definidas:  
 ```powershell  
 docker exec -it rag\_postgres psql -U $env:POSTGRES\_USER -d $env:POSTGRES\_DB  
 ```  
2. Verifica que Qdrant está corriendo y puedes crear colecciones de vectores:  
 ```powershell  
 curl http://localhost:6333/collections  
 ```  
3. Verifica que Ollama está disponible:  
 ```powershell  
 curl http://localhost:11434/v1/models  
 ```  
4. Finalmente, realiza una consulta de prueba al servicio `app` para asegurarte de que el RAG responde correctamente:  
 ```powershell  
 curl http://localhost:8000/query -d "{\"pregunta\": \"¿Qué es RAG?\"}" -H "Content-Type: application/json"  
 ```

## 9. Conclusión

Con estos pasos, has creado desde cero un sistema RAG containerizado en Docker. Has clonado el proyecto desde GitHub, creado las credenciales necesarias, definido los contenedores para cada servicio, y verificado que el sistema funcione correctamente. Ahora puedes ampliar esta configuración agregando más funcionalidades, optimizando el modelo o integrando nuevas fuentes de datos.