**Propuesta de Despliegue**

El objetivo de esta propuesta de despliegue es separar los componentes del chatbot para aprovechar los servicios gratuitos o recursos de laboratorio, manteniendo precisión y bajo costo. Se han elegido Railway y una computadora del laboratorio de sistemas de la UTN.

**Parte del sistema que se desplegará en Railway**

Se va a utilizar **Railway** para el despliegue de:

* Discord Bot (Python + discord.py)
* LangChain + ChromaDB (base de datos de vectores)
* PostgreSQL (base de datos relacional)
* Comunicación asíncrona con modelo LLM

Se ha seleccionado Railway dado a que es:

* gratuito hasta cierto uso.
* ideal para alojar bots sencillos.
* combinable con repositorio de GitHub, lo que permite que haya un despliegue más sencillo.
* Intuitivo, fácil de usar.

Se han descartado otras opciones:

1. Docker junto con máquina virtual en la nube ya que se trata de servicios pagados, se requiere conocimiento muy técnico en redes, puertos y seguridad. Además, hay que encargarse de actualizar y monitorear el sistema.
2. Google Cloud Platform (GCP) tiene un nivel gratuito (con límites de uso mensual que se desconecta si se pasa) llamado *Free Tier* pero es complicado de configurar. No es lo más fácil ni lo más libre. A veces parece gratis y te terminan pidiendo tarjeta.

**Parte del sistema que se desplegará en una computadora del Laboratorio de Sistemas de la UTN**

Se va a utilizar una **computadora del Laboratorio de Sistemas de la UTN para correr Mistral**, ejecutando un servidor local FastAPI para atender peticiones.

* un servidor local tipo FastAPI implica correr una API en Python (como un microservicio), en esa PC del laboratorio, que va a exponer un endpoint (por ejemplo: http://ip-del-lab:8000/consultar\_mistral) al que el bot le puede mandar preguntas y recibir respuestas.

**Requisitos de la computadora del Laboratorio de Sistemas de la UTN**

Se puede usar una computadora del Laboratorio de Sistemas de la UTN para correr Mistral siempre que cumpla estos requisitos:

* Sistema operativo (hay dos opciones):
  + Linux
  + Windows (hay dos opciones)
    - con WSL2: funcionalidad de Windows que permite correr Linux dentro de Windows, sin máquina virtual completa para programar como si estuviera en Linux.
    - con Docker: plataforma de software que permite crear, probar, implementar, actualizar y administrar aplicaciones en contenedores. Los contenedores son unidades estandarizadas que contienen todo lo necesario para ejecutar una aplicación.
* CPU: 4 núcleos reales como mínimo. Lo ideal serían 6 u 8.
* RAM: 16 GB mínimo. Lo ideal sería 32 GB.
* Almacenamiento: SSD con al menos 20-30 GB libres (solo para modelo y entorno).
* Conexión a internet para vincular Mistral con Railway.

Si se pudiera acceder a una máquina con GPU (como por ejemplo NVIDIA) sería mucho mejor porque bajaría el tiempo de respuesta y consumo de RAM.

* Las GPU (Graphics Processing Units) son placas gráficas que pueden hacer cálculos matemáticos mucho más rápido que las CPUs normales, sobre todo en tareas paralelas como el procesamiento de modelos de inteligencia artificial.
* Mistral, como cualquier LLM (Modelo de Lenguaje Largo), fue diseñado para correr en GPU, pero puede correr en CPU con un consumo alto de RAM y funcionamiento lento. Podría tardar varios segundos o incluso minutos en responder, dependiendo de los requisitos que cumpla la computadora del laboratorio de sistemas.

**Beneficios de la elección de tecnologías de despliegue**

* Herramientas de despliegue sin costo (uso de versiones gratuitas)
* No se utilizan APIs pagas
* Flexibilidad para pruebas y mejoras

**Posibles riesgos:**

* PC del laboratorio de la UTN no cumple requisitos mínimos para correr Mistral: Consultar si UTN tiene servidores o recomendación.
* Mistral muy lento en CPU: cumpliéndose los requisitos mínimos para funcionar en una computadora, se puede usar una versión más liviana llamada gguf.
  + gguf es una versión comprimida de Mistral para correr localmente (se usa con llama.cpp y sus variantes, que sirven para cargar y usar el modelo)
    - Mistral 7B Q4\_0.gguf → versión más liviana, ocupa menos RAM, pero es menos precisa.
    - Mistral 7B Q8\_0.gguf → más pesada, más precisa.
* Seguridad del Laboratorio: a continuación, se presentan alternativas sobre este tema.

**Alternativas para exponer el servidor del laboratorio de forma segura**

*Railway* se debe conectar a un servidor local (PC del laboratorio de Sistemas de la UTN donde corre Mistral), sin dejar la red abierta ni comprometer la seguridad.

Las alternativas posibles son usar alguna de estas dos herramientas:

* Ngrok: permite exponer un servidor local (FastAPI en la Computadora del Laboratorio) a internet de forma segura y temporal mediante un túnel cifrado.
  + Ejemplo: se corre FastAPI con el Modelo Mistral en la PC del laboratorio en localhost:8000 (algún puerto) y ngrok lo convierte en <https://abcd1234.ngrok.io>. Se le proporciona esa URL al bot que está en Railway y puede hacerle peticiones al modelo Mistral.
  + La dificultad en implementación y uso es baja, ya que se instala y se ejecuta.
  + Gratuito en su versión básica.
* VPN (Virtual Private Network): herramienta para crear una red privada entre la PC del laboratorio y Railway.
  + La Dificultad es alta ya que se necesita configurar un servidor VPN.

Considero que ngrok es la alternativa adecuada ya que:

* No es un chatbot para miles de usuarios en simultáneo.
* Permite levantar el servidor local (Mistral con FastAPI o Flask) y exponerlo temporalmente a Internet de forma rápida, segura y sin demasiadas complicaciones.
* Se puede configurar una API key para limitar el acceso al bot, sumando una capa de seguridad
  + Una API key es una “contraseña” que se incorpora en el código para que se puedan usar el servicio que se proporciona (acceso local a una computadora del laboratorio para utilizar Mistral e interactuar con el chatbot). Sirve para controlar quién puede acceder al servidor.
    - Ejemplo: 1234abcd-5678-efgh-9012-ijklmnop3456.
  + Si se tiene un servidor local que sirve respuestas de Mistral, y se lo expone a Internet con ngrok, cualquiera que tenga el link de ngrok podría intentar acceder. Para evitar eso:
    - En el código (FastAPI/Flask) se puede hacer que todas las solicitudes al bot revisen si el cliente está enviando una API key válida.
      * Si no la manda o es incorrecta, se rechaza la solicitud. Así, aunque alguien descubra la URL de ngrok, no va a poder usar el bot si no tiene la clave correcta.