INTRODUCCIÓN
La documentación en el enlace que proporcionaste trata sobre **Rasa Open Source** y **Rasa Pro**.

Rasa Open Source es un marco de trabajo de código abierto muy popular para construir asistentes de IA basados en chat y voz. Ha sido descargado más de 25 millones de veces y proporciona los componentes esenciales para crear asistentes virtuales o chatbots. Permite comprender y mantener conversaciones, así como conectarse a canales de mensajería y sistemas de terceros a través de un conjunto de APIs.

Rasa Pro es la oferta comercial de Rasa, diseñada para satisfacer las necesidades empresariales en términos de seguridad, capacidad de observación y escalabilidad. Es parte de la solución empresarial Rasa Platform, que también incluye Rasa X/Enterprise, una interfaz de usuario de bajo código que apoya a los equipos de IA conversacional en la revisión y mejora de los asistentes de IA a gran escala.

En la documentación, cuando un contenido es aplicable solo a Rasa Pro, se señala con la etiqueta "Rasa Pro Only", lo que indica que se necesita una licencia de Rasa Pro para acceder a esas capacidades.

Puedes encontrar más detalles en la [documentación de Rasa X/Enterprise](https://rasa.com/docs/rasa-enterprise) y la [política de lanzamiento y mantenimiento de productos de Rasa](https://rasa.com/rasa-product-release-and-maintenance-policy/).

RASA PRO
La documentación sobre **Rasa Pro** describe este producto como la oferta comercial y de código profesional de Rasa, diseñada para satisfacer las necesidades empresariales en cuanto a seguridad, observabilidad y escalabilidad. Rasa Pro se integra perfectamente con los entornos tecnológicos empresariales y es la plataforma recomendada para todos los clientes empresariales de Rasa.

Características de Rasa Pro:

1. **Analytics with Conversational Data Pipeline**: Permite visualizar las métricas de Rasa en una herramienta de terceros para medir el rendimiento del asistente. [Más información aquí](https://rasa.com/docs/rasa/monitoring/analytics/getting-started-with-analytics).

2. **Concurrent Lock Store**: Escala la implementación y maneja de manera confiable altos volúmenes de tráfico en múltiples instancias de Rasa, asegurando que no se pierdan mensajes. [Más información aquí](https://rasa.com/docs/rasa/lock-stores#concurrentredislockstore).

3. **End-to-End Testing**: Solución de pruebas de extremo a extremo diseñada para cumplir con los criterios de integración y aceptación de grado empresarial. [Más información aquí](https://rasa.com/docs/rasa/testing-your-assistant#end-to-end-testing).

4. **IVR Voice Connector**: Integra con sistemas IVR de primera categoría a través de conectores de voz preconfigurados. [Más información aquí](https://rasa.com/docs/rasa/connectors/audiocodes-voiceai-connect).

5. **Observabilidad (Tracing)**: Ayuda a resolver problemas de rendimiento rápidamente e identificar cuellos de botella en el manejo de mensajes y entrenamiento de modelos. [Más información aquí](https://rasa.com/docs/rasa/monitoring/tracing).

6. **Manejo de PII (Información Personalmente Identificable)**: Anonimiza la PII en los registros y eventos transmitidos a través del broker de eventos Kafka. [Más información aquí](https://rasa.com/docs/rasa/pii-management).

7. **Real-Time Markers**: Permite marcar puntos de interés en las conversaciones para apoyar el análisis dirigido de las experiencias de los usuarios en tiempo real. [Más información aquí](https://rasa.com/docs/rasa/monitoring/analytics/realtime-markers).

8. **Manejo de Secretos**: Mejora la seguridad con la integración de Vault, permitiendo la rotación dinámica de credenciales para las bases de datos de Rasa sin interrupciones del sistema. [Más información aquí](https://rasa.com/docs/rasa/secrets-managers).

9. **Security Scanning for Vulnerability Protection**: Realiza escaneos de seguridad nocturnos y aplica parches proactivos en la imagen de Docker para mantener las dependencias actualizadas.

10. **Spaces (Alpha Release)**: Modulariza el asistente para mejorar la escalabilidad y la colaboración en equipo. [Más información aquí](https://rasa.com/docs/rasa/spaces).

SETTING UP ENVIRONMENT
La documentación sobre la configuración del entorno para Rasa proporciona los siguientes pasos clave:

1. Configuración del entorno Python
- **Verificar si Python está configurado**: Usa los siguientes comandos para verificar si ya tienes Python configurado:
  ```bash
  python3 --version
  pip3 --version
  ```
- **Versiones compatibles con Python**: Rasa es compatible con las versiones `3.7`, `3.8`, `3.9` y `3.10` de Python. Sin embargo, Python `3.10` solo es compatible a partir de Rasa `3.4.x`, y su instalación en Apple Silicon con Python `3.10` no es funcional en `3.4.x`, pero se espera que sea compatible desde `3.5.x`.

- **Instalación de paquetes**: Si no tienes Python y pip instalados, puedes instalar los paquetes necesarios usando `apt` y configurar un entorno virtual con `pip`:

```bash
sudo apt update
sudo apt install python3-dev python3-pip
```

2. Configuración del entorno virtual
- **Recomendación**: Se recomienda encarecidamente usar entornos virtuales para aislar proyectos de Python. Herramientas como [virtualenv](https://virtualenv.pypa.io/en/latest/) y [virtualenvwrapper](https://virtualenvwrapper.readthedocs.io/en/latest/) proporcionan entornos Python aislados, lo que evita conflictos de dependencias y permite instalar paquetes sin privilegios de root.

- **Crear y activar el entorno virtual**:
```bash
source ./venv/bin/activate
```

Limitaciones para M1 / M2 (Apple Silicon)
- **Uso de GPU**: Las instalaciones de Rasa en Apple Silicon no utilizan [Apple Metal](https://developer.apple.com/metal/) por defecto. Se ha encontrado que usar la GPU en Apple Silicon aumenta significativamente el tiempo de entrenamiento para los componentes `DIETClassifier` y `TEDPolicy`.
- **Restricciones**:
  - No se puede ejecutar Duckling como contenedor Docker en Apple Silicon. Se recomienda un despliegue en la nube de Duckling si necesitas utilizar el extractor de entidades Duckling.
  - Rasa en Apple Silicon no soporta el componente `ConveRTFeaturizer` ni las canalizaciones que lo contienen, debido a la falta de soporte nativo para `tensorflow-text`.

INSTALLING RASA OPEN SOURCE
La documentación sobre la instalación de **Rasa Open Source** cubre los siguientes puntos clave:

Instalación de Rasa Open Source
1. **Actualización de `pip`**: Asegúrate de que tu versión de `pip` esté actualizada antes de comenzar la instalación.
2. **Instalación básica**: Simplemente ejecuta el comando de instalación proporcionado para instalar Rasa Open Source. Al ejecutar Rasa por primera vez, verás un mensaje sobre la recolección de datos de uso anónimo, que puedes leer más en la [documentación de telemetría](https://rasa.com/docs/rasa/telemetry/telemetry).

Crear un Nuevo Proyecto
Una vez instalada, puedes crear un nuevo proyecto de Rasa utilizando los comandos proporcionados en la documentación.

Construcción desde el Código Fuente
Para utilizar la versión de desarrollo de Rasa Open Source desde GitHub:
- Usa `git clone` para clonar el repositorio de Rasa.

- Instala las dependencias utilizando `poetry`.

Dependencias Adicionales
Algunos algoritmos de aprendizaje automático requieren paquetes adicionales que no se instalan por defecto para mantener el tamaño del paquete pequeño. Puedes instalar todas las dependencias adicionales utilizando:
```bash
pip3 install 'rasa[full]'
```

Requisitos para Python 3.10
- **Linux**: Si usas Linux, puede haber fallos al instalar `tokenizers` y `cryptography`. Para resolverlo, instala un compilador de Rust.
- **macOS**: En macOS, puede haber fallos al instalar `tokenizers`. Para resolverlo, instala un compilador de Rust mediante `brew`.

Dependencias para spaCy
Puedes instalar spaCy y su modelo de idioma en inglés con:
```bash
pip3 install 'rasa[spacy]'
python3 -m spacy download en_core_web_md
```
Es recomendable usar modelos de tamaño "medium" (`_md`) o más grandes para un mejor rendimiento.

Dependencias para MITIE
Instala MITIE utilizando:
```bash
pip3 install git+https://github.com/mit-nlp/MITIE.git
pip3 install 'rasa[mitie]'
```
Luego, descarga los modelos MITIE necesarios y guárdalos en una ubicación accesible para tu proyecto.

Actualización de Versiones
Para actualizar tu instalación de Rasa Open Source a la última versión disponible en PyPI:
```bash
pip3 install --upgrade rasa
```
Para descargar una versión específica, especifica el número de versión en el comando.

BUILDING ASSISTANTS: COMMAND LINE

Cheat Sheet

Hoja de trucos

- **`rasa init`**: Crea un nuevo proyecto con datos de entrenamiento de ejemplo, acciones y archivos de configuración.
- **`rasa train`**: Entrena un modelo utilizando tus datos NLU y las historias, guarda el modelo entrenado en `./models`.
- **`rasa interactive`**: Inicia una sesión de aprendizaje interactivo para crear nuevos datos de entrenamiento charlando con tu asistente.
- **`rasa shell`**: Carga tu modelo entrenado y te permite hablar con tu asistente en la línea de comandos.
- **`rasa run`**: Inicia un servidor con tu modelo entrenado.
- **`rasa run actions`**: Inicia un servidor de acciones utilizando el SDK de Rasa.
- **`rasa visualize`**: Genera una representación visual de tus historias.
- **`rasa test`**: Prueba un modelo de Rasa entrenado en cualquier archivo que comience con `test_`.
- **`rasa test e2e`**: Realiza pruebas de extremo a extremo completamente integradas con el servidor de acciones.
- **`rasa data split nlu`**: Realiza una división 80/20 de tus datos de entrenamiento NLU.
- **`rasa data split stories`**: Similar a lo anterior, pero para los datos de historias.
- **`rasa data convert`**: Convierte los datos de entrenamiento entre diferentes formatos.
- **`rasa data migrate`**: Migra el dominio 2.0 al formato 3.0.
- **`rasa data validate`**: Verifica inconsistencias en el dominio, NLU y datos de conversación.
- **`rasa export`**: Exporta conversaciones desde un almacén de rastreo a un intermediario de eventos.
- **`rasa evaluate markers`**: Extrae marcadores de un almacén de rastreo existente.
- **`rasa marker upload`**: Sube configuraciones de marcadores al Analytics Data Pipeline.
- **`rasa license`**: Muestra información sobre la licencia.
- **`rasa -h`**: Muestra todos los comandos disponibles.

La sección sobre **niveles de registro (Log Level)** de la interfaz de línea de comandos de Rasa proporciona información sobre cómo controlar la cantidad y tipo de mensajes que se muestran en los registros (logs). Aquí te dejo un resumen traducido al español:

Nivel de Registro (Log Level)

Rasa genera mensajes de registro en diferentes niveles (por ejemplo, advertencias, información, errores, etc.). Puedes controlar el nivel de registros que deseas ver con los argumentos de línea de comandos `--verbose` (equivalente a `-v`) o `--debug` (equivalente a `-vv`). A continuación, se explican más detalles sobre estos argumentos:

- **`--verbose (-v)`**: Muestra mensajes de registro a nivel de información (INFO).
- **`--debug (-vv)`**: Muestra muchos mensajes de depuración, estableciendo el nivel de registro en depuración (DEBUG).

Además de los argumentos de la CLI, existen varias variables de entorno que te permiten controlar la salida de los registros de manera más granular. Con estas variables, puedes configurar los niveles de registro para los mensajes creados por bibliotecas externas como Matplotlib, Pika y Kafka. Estas variables siguen los [niveles de registro estándar en Python](https://docs.python.org/3/library/logging.html#logging-levels). Las variables de entorno actualmente admitidas son:

1. **LOG_LEVEL_LIBRARIES**: Configura el nivel de registro general para las principales bibliotecas que usa Rasa, como Tensorflow, `asyncio`, APScheduler, SocketIO, Matplotlib, RabbitMQ, y Kafka.
2. **LOG_LEVEL_MATPLOTLIB**: Configura específicamente el nivel de registro solo para Matplotlib.
3. **LOG_LEVEL_RABBITMQ**: Configura específicamente el nivel de registro solo para las bibliotecas AMQP (como `aio_pika` y `aiormq`).
4. **LOG_LEVEL_KAFKA**: Configura específicamente el nivel de registro solo para Kafka.
5. **LOG_LEVEL_PRESIDIO**: Configura específicamente el nivel de registro solo para Presidio (`presidio_analyzer` y `presidio_anonymizer`).
6. **LOG_LEVEL_FAKER**: Configura específicamente el nivel de registro solo para Faker.

La configuración general (`LOG_LEVEL_LIBRARIES`) tiene menor prioridad que la configuración específica de cada biblioteca (`LOG_LEVEL_MATPLOTLIB`, `LOG_LEVEL_RABBITMQ`, etc.). Esto significa que las variables se pueden usar juntas con un resultado predecible. Por ejemplo:

```bash
LOG_LEVEL_LIBRARIES=ERROR LOG_LEVEL_MATPLOTLIB=WARNING
LOG_LEVEL_KAFKA=DEBUG rasa shell --debug
```

El comando anterior mostrará:

- Mensajes con nivel `DEBUG` y superior por defecto (debido a `--debug`).
- Mensajes con nivel `WARNING` y superior para Matplotlib.
- Mensajes con nivel `DEBUG` y superior para Kafka.
- Mensajes con nivel `ERROR` y superior para otras bibliotecas no configuradas.

Ten en cuenta que la configuración de la CLI establece el nivel más bajo de mensajes de registro que se manejarán. Por ejemplo, el siguiente comando establecerá el nivel de registro en `INFO` (debido a `--verbose`), y no se verán mensajes de depuración (la configuración a nivel de biblioteca no tendrá ningún efecto):

```bash
LOG_LEVEL_LIBRARIES=DEBUG LOG_LEVEL_MATPLOTLIB=DEBUG rasa shell
--verbose
```

La sección sobre **configuración de registro personalizada (Custom Logging Configuration)** de la interfaz de línea de comandos de Rasa describe cómo puedes configurar los registros (logs) utilizando un archivo YAML. A continuación, te ofrezco un resumen traducido al español:

Configuración de Registro Personalizada

La CLI de Rasa ahora incluye un nuevo argumento `--logging-config-file`, que acepta un archivo YAML como valor.

Puedes configurar cualquier formateador o manejador de registros en un archivo YAML separado. El archivo de configuración de registros YAML debe seguir el [esquema de diccionario incorporado de Python](https://docs.python.org/3/library/logging.config.html#dictionary-schema-details); de lo contrario, fallará la validación. Puedes pasar este archivo como argumento a la opción de la CLI `--logging-config-file` y usarlo con cualquiera de los comandos de Rasa.

Aquí tienes la traducción completa de la sección "rasa init" del documento:

Rasa Init

Este comando configura un asistente completo para ti con algunos datos de entrenamiento de ejemplo:

Crea los siguientes archivos:

```
.
├── actions
│   ├── __init__.py
│   └── actions.py
├── config.yml
├── credentials.yml
├── data
│   ├── nlu.yml
│   └── stories.yml
├── domain.yml
├── endpoints.yml
├── models
│   └── <timestamp>.tar.gz
└── tests
    └── test_stories.yml
```

Te preguntará si deseas entrenar un modelo inicial utilizando estos datos. Si respondes que no, el directorio `models` estará vacío.

Cualquiera de los comandos CLI predeterminados esperará esta configuración de proyecto, por lo que esta es la mejor manera de comenzar. Puedes ejecutar `rasa train`, `rasa shell` y `rasa test` sin necesidad de configuración adicional.

Aquí tienes la traducción completa de la sección "rasa train" del documento:

Rasa Train

El siguiente comando entrena un modelo de Rasa:

Si tienes modelos existentes en tu directorio (por defecto bajo `models/`), solo se volverán a entrenar las partes de tu modelo que hayan cambiado. Por ejemplo, si editas tus datos de entrenamiento NLU y nada más, solo se entrenará la parte de NLU.

Si deseas entrenar un modelo NLU o de diálogo individualmente, puedes ejecutar `rasa train nlu` o `rasa train core`. Si proporcionas datos de entrenamiento solo para uno de estos, `rasa train` recurrirá a uno de estos comandos por defecto.

`rasa train` guardará el modelo entrenado en el directorio definido por `--out`, por defecto en `models/`. El nombre del modelo por defecto es `<timestamp>.tar.gz`. Si deseas nombrar tu modelo de manera diferente, puedes especificar el nombre utilizando la opción `--fixed-model-name`.

Por defecto, se ejecuta una validación antes de entrenar el modelo. Si deseas omitir la validación, puedes usar la opción `--skip-validation`. Si deseas que se produzca un error en caso de advertencias durante la validación, puedes usar la opción `--fail-on-validation-warnings`. La opción `--validation-max-history` es análoga al argumento `--max-history` de `rasa data validate`.

Los siguientes argumentos pueden utilizarse para configurar el proceso de entrenamiento:

```
usage: rasa train [-h] [-v] [-vv] [--quiet]
                  [--logging-config-file LOGGING_CONFIG_FILE]
                  [--data DATA [DATA ...]] [-c CONFIG] [-d DOMAIN] [--out OUT]
                  [--dry-run] [--skip-validation]
                  [--fail-on-validation-warnings]
                  [--validation-max-history VALIDATION_MAX_HISTORY]
                  [--augmentation AUGMENTATION] [--debug-plots]
                  [--num-threads NUM_THREADS]
                  [--fixed-model-name FIXED_MODEL_NAME] [--persist-nlu-data]
                  [--force] [--finetune [FINETUNE]]
                  [--epoch-fraction EPOCH_FRACTION] [--endpoints ENDPOINTS]
                  {core,nlu} ...
```

Argumentos posicionales:

- `{core,nlu}`:
  - `core`: Entrena un modelo Rasa Core utilizando tus historias.
  - `nlu`: Entrena un modelo Rasa NLU utilizando tus datos NLU.

Opciones:

- `-h, --help`: Muestra este mensaje de ayuda y sale.
- `--data DATA [DATA ...]`: Rutas a los archivos de datos Core y NLU. (por defecto: ['data'])
- `-c CONFIG, --config CONFIG`: La configuración de la política y el pipeline NLU de tu bot. (por defecto: config.yml)
- `-d DOMAIN, --domain DOMAIN`: Especificación del dominio. Puede ser un único archivo YAML o un directorio que contenga varios archivos con especificaciones de dominio. El contenido de estos archivos se leerá y combinará. (por defecto: domain.yml)
- `--out OUT`: Directorio donde se deben guardar tus modelos. (por defecto: models)
- `--dry-run`: Si se habilita, no se realizará ningún entrenamiento real. En su lugar, se determinará si se debe reentrenar un modelo y esta información se imprimirá como salida. El código de retorno es una máscara de bits de 4 bits que también se puede utilizar para determinar qué exactamente necesita ser reentrenado:
  - 0 significa que no se requiere entrenamiento extenso.
  - 1 significa que el modelo necesita ser reentrenado.
  - 8 significa que el entrenamiento fue forzado (se especificó el argumento `--force`).
- `--skip-validation`: Omite el paso de validación antes del entrenamiento. (por defecto: False)
- `--fail-on-validation-warnings`: Produce un error en caso de advertencias de validación. Si se omite, solo los errores saldrán con un código de estado distinto de cero (por defecto: False).
- `--validation-max-history VALIDATION_MAX_HISTORY`: Número de turnos considerados para la validación de la estructura de las historias. (por defecto: None)
- `--augmentation AUGMENTATION`: Cuánto aumento de datos se utilizará durante el entrenamiento. (por defecto: 50)
- `--debug-plots`: Si se habilita, creará gráficos que muestren los puntos de control y sus conexiones entre bloques de historias en un archivo llamado `story_blocks_connections.html`. (por defecto: False)
- `--num-threads NUM_THREADS`: Número máximo de hilos a utilizar durante el entrenamiento. (por defecto: None)
- `--fixed-model-name FIXED_MODEL_NAME`: Si se establece, el nombre del archivo/directorio del modelo se establecerá con el nombre dado. (por defecto: None)
- `--persist-nlu-data`: Persiste los datos de entrenamiento NLU en el modelo guardado. (por defecto: False)
- `--force`: Fuerza el entrenamiento de un modelo incluso si los datos no han cambiado. (por defecto: False)
- `--finetune [FINETUNE]`: Ajusta un modelo previamente entrenado. Si no se proporciona una ruta del modelo, Rasa Open Source intentará ajustar el último modelo entrenado desde el directorio del modelo especificado mediante `--out`. (por defecto: None)
- `--epoch-fraction EPOCH_FRACTION`: Fracción de épocas que se utilizarán al ajustar un modelo. (por defecto: None)
- `--endpoints ENDPOINTS`: Archivo de configuración para los conectores como archivo yml. (por defecto: endpoints.yml)

Opciones de registro de Python:

- `-v, --verbose`: Sé detallado. Establece el nivel de registro en INFO. (por defecto: None)
- `-vv, --debug`: Imprime muchas declaraciones de depuración. Establece el nivel de registro en DEBUG. (por defecto: None)
- `--quiet`: ¡Silencio! Establece el nivel de registro en WARNING. (por defecto: None)
- `--logging-config-file LOGGING_CONFIG_FILE`: Si se establece, el nombre del archivo de configuración de registro se establecerá con el nombre dado. (por defecto: None)

Consulta la sección sobre [aumento de datos](https://rasa.com/docs/rasa/policies#data-augmentation) para obtener información sobre cómo funciona el aumento de datos y cómo elegir un valor para la bandera. Nota que `TEDPolicy` es la única política afectada por el aumento de datos.

Entrenamiento incremental (Incremental Training)

Esta característica es experimental. Introducimos funciones experimentales para recibir retroalimentación de nuestra comunidad, por lo que te animamos a probarla. Sin embargo, la funcionalidad podría cambiar o eliminarse en el futuro. Si tienes comentarios (positivos o negativos), compártelos con nosotros en el [Foro de Rasa](https://forum.rasa.com).

Para mejorar el rendimiento de un asistente, es útil practicar el [CDD (Desarrollo Impulsado por la Conversación)](https://rasa.com/docs/rasa/conversation-driven-development) y agregar nuevos ejemplos de entrenamiento basados en cómo los usuarios han interactuado con tu asistente. Puedes usar `rasa train --finetune` para inicializar la tubería con un modelo ya entrenado y afinarlo con el nuevo conjunto de datos de entrenamiento que incluye los ejemplos adicionales. Esto ayudará a reducir el tiempo de entrenamiento del nuevo modelo.

Por defecto, el comando utiliza el último modelo en el directorio `models/`. Si tienes un modelo específico que deseas mejorar, puedes especificar la ruta al mismo ejecutando `rasa train --finetune <ruta al modelo para afinar>`. Afinar un modelo generalmente requiere menos épocas para entrenar componentes de aprendizaje automático como `DIETClassifier`, `ResponseSelector` y `TEDPolicy` en comparación con entrenar desde cero. Puedes usar una configuración de modelo para afinado que defina menos épocas que antes o usar la bandera `--epoch-fraction`. `--epoch-fraction` usará una fracción de las épocas especificadas para cada componente de aprendizaje automático en el archivo de configuración del modelo. Por ejemplo, si `DIETClassifier` está configurado para usar 100 épocas, especificar `--epoch-fraction 0.5` solo usará 50 épocas para el afinado.

También puedes afinar un modelo solo de NLU o solo de gestión de diálogos utilizando `rasa train nlu --finetune` y `rasa train core --finetune`, respectivamente.

Para poder afinar un modelo, deben cumplirse las siguientes condiciones:

1. La configuración proporcionada debe ser exactamente la misma que se utilizó para entrenar el modelo que se está afinando. El único parámetro que puedes cambiar es `epochs` para los componentes y políticas individuales de aprendizaje automático.