Conversión de Archivos Word a Markdown en Google Cloud

# Resumen del Proyecto

El objetivo de este proyecto es desarrollar una imagen Docker que aproveche la potencia de la herramienta Pandoc para llevar a cabo conversiones de documentos en formato Word (.docx) a formato Markdown (.md). La imagen Docker se basará en la versión oficial de Pandoc e incorporará el paquete zip para comprimir los archivos resultantes.

Para gestionar las solicitudes de conversión de Word a Markdown, crearemos un servidor utilizando la librería Flask de Python. En el servidor, alojaremos una pequeña aplicación de Python que será responsable de procesar las solicitudes a través de una API REST. Esta aplicación estará contenida dentro del mismo contenedor Docker.

La versatilidad del contenedor Docker permitirá su fácil implementación en Google Cloud Run, lo que permitirá a los usuarios enviar archivos Word mediante comandos POST a través de la API. A cambio, recibirán el contenido convertido en formato .zip, el cual incluirá el archivo .md junto con una carpeta de medios que contendrá todas las imágenes extraídas del documento original.

El servicio resultante proporcionará una solución sencilla, rápida y eficiente para la conversión y compresión de documentos Word. Esto permitirá a los usuarios integrar la herramienta en diversas aplicaciones y flujos de trabajo, mejorando la productividad y la facilidad de uso al realizar estas tareas de manera automatizada.

# Pasos a seguir

Para lograr esto, necesitas crear un Dockerfile que contenga Pandoc y otros componentes necesarios para realizar la conversión y la compresión. Luego, puedes implementar este Dockerfile en Google Cloud Run. A continuación, te guiaré a través de los pasos para crear el Dockerfile y construir la imagen Docker:

## Paso 1: Crear el Dockerfile

Crea un archivo llamado "Dockerfile" (sin extensión) y agrégale el siguiente contenido:

*# Usar una imagen de Python*

*FROM* python:3.9

*# Instalar pandoc*

*RUN* apt-get update && apt-get install -y pandoc

*# Copiar los archivos necesarios al contenedor*

*COPY* app.py requirements.txt ./

*# Instalar las dependencias*

*RUN* pip install --no-cache-dir -r requirements.txt

*# Iniciar la aplicación al arrancar el contenedor*

*CMD* ["python", "app.py"]

## Paso 2: Crear app.py

Crea un archivo app.py con este contenido:

*from* flask *import* *Flask*, request, send\_file

*from* io *import* *BytesIO*

*import* os

*import* subprocess

*import* shutil

*import* tempfile

*import* zipfile

app *=* *Flask*(\_\_name\_\_)

*def* process\_word\_file(file\_path):

    """

    Procesa un archivo Word (.docx) utilizando pandoc y crea un archivo ZIP

    que contiene el archivo de salida en formato markdown (output.md) y la

    carpeta 'media' con todas las imágenes extraídas del archivo.

    :param file\_path: Ruta del archivo Word de entrada (.docx).

    :return: Ruta del archivo ZIP generado que contiene output.md y la carpeta 'media'.

    """

*# Procesar el archivo Word usando pandoc*

*try*:

        subprocess.run(['pandoc', '-t', 'markdown\_mmd', '--extract-media', '.', file\_path, '-o', 'output.md'], check*=*True)

*except* subprocess.*CalledProcessError* *as* e:

*raise* *RuntimeError*("Error al procesar el archivo Word. Asegúrate de que el archivo tenga el formato correcto.") *from* e

*# Verificar si el archivo 'output.md' se generó correctamente*

*if* *not* os.path.exists('output.md'):

*raise* *RuntimeError*("El archivo 'output.md' no se generó correctamente.")

    temp\_dir *=* tempfile.gettempdir()

    zip\_file\_path *=* os.path.join(temp\_dir, 'salida.zip')

    current\_dir *=* os.path.abspath('.')

    output\_md *=* os.path.join(current\_dir, 'output.md')

    media\_dir *=* os.path.join(current\_dir, 'media')

*# Crear el archivo ZIP incluyendo el contenido de la carpeta 'media'*

*with* zipfile.*ZipFile*(zip\_file\_path, 'w') *as* zip\_file:

        zip\_file.write(output\_md, os.path.basename(output\_md))

*for* foldername, subfolders, filenames *in* os.walk(media\_dir):

*for* filename *in* filenames:

                file\_path *=* os.path.join(foldername, filename)

                zip\_file.write(file\_path, os.path.relpath(file\_path, current\_dir))

*# Eliminar archivos temporales*

    os.remove('output.md')

    shutil.rmtree('media')

*return* zip\_file\_path

@app.route("/", methods*=*['GET', 'POST'])

*def* process\_file():

*if* request.method *==* 'GET':

*return* """<h1>API de conversión Word a Markdown</h1>

<p>Bienvenido a la API de conversión de archivos Word a Markdown.</p>

<p><strong>Descripción:</strong></p>

<p>Puedes enviar un archivo en formato Word (.docx) utilizando el método POST y la API te devolverá un archivo ZIP que contiene el archivo de salida en formato Markdown (output.md) y la carpeta 'media' con todas las imágenes extraídas del archivo Word.</p>

<p><strong>Endpoint:</strong></p>

<p>Método POST: /</p>

<p><strong>Parámetros:</strong></p>

<p>file: Archivo Word (.docx) a convertir (en el cuerpo de la solicitud).</p>

<p><strong>Respuesta:</strong></p>

<p>La API devolverá un archivo ZIP que contiene output.md y la carpeta 'media'. Puedes descargar el archivo resultante desde la respuesta.</p>

"""

*elif* request.method *==* 'POST':

*if* 'file' *not* *in* request.files:

*return* "No se proporcionó un archivo en la solicitud.", 400

*try*:

            file *=* request.files['file']

            temp\_dir *=* tempfile.gettempdir()

            file\_path *=* os.path.join(temp\_dir, 'entrada.docx')

            file.save(file\_path)

            zip\_file\_path *=* process\_word\_file(file\_path)

*with* open(zip\_file\_path, 'rb') *as* f:

                zip\_content *=* f.read()

            os.remove(zip\_file\_path)

*return* send\_file(*BytesIO*(zip\_content), as\_attachment*=*True, download\_name*=*'salida.zip')

*except* *Exception* *as* e:

*return* *f*"Ocurrió un error durante el procesamiento del archivo: {*str*(e)}", 500

*if* \_\_name\_\_ *==* "\_\_main\_\_":

    app.run(host*=*'0.0.0.0', port*=*5000)

Puedes probar app.py usando Python en local. Tal vez necesites instalar Flask con el comando:

pip install Flask

## Paso 3: Crear los requerimientos

En el archivo requirements.txt pon estas dependencias:

Flask

Flask es una librería para crear un servidor HTTP que implemente una API REST.

## Paso 4: Construir la imagen Docker

Los archivos "Dockerfile", “app.py” y “requirements.txt” han de estar en la misma carpeta, en la que ejecutaremos el siguiente comando para construir la imagen Docker:

docker build -t pandoc\_converter .

Esto construirá la imagen con el nombre "pandoc\_converter".

## Paso 5: Ejecutar la imagen localmente (opcional)

Antes de implementar la imagen en Google Cloud Run, puedes probarla localmente para asegurarte de que funciona correctamente. Para hacerlo, ejecuta:

docker run -it -p 5000:5000 pandoc\_converter

Esto arrancará el servidor API REST en el puerto 5000.

Ahora puedes probar la API REST usando Postman o una herramienta similar.

## Paso 6: (opcional) Instalar la consola de Google Cloud

Instala, si acaso no lo tienes todavía, la consola de Google Cloud. Sigue los pasos indicados en:

<https://cloud.google.com/sdk/docs/install?hl=es-419>

## Paso 7: Preparar el proyecto y las APIS en Google cloud

Primero debes crear un proyecto en Google Cloud. Una vez lo hayas creado debes habilitar la API de Google Cloud Run, y de Artifacts.

Habilita la API de artifacts desde [Google Cloud Console](https://console.cloud.google.com/flows/enableapi?apiid=artifactregistry.googleapis.com&hl=es-419) o con el siguiente comando gcloud:

gcloud services enable artifactregistry.googleapis.com

## Paso 8: Creamos un repositorio para las imágenes Docker

Crearemos un repositorio en artifacts en una de las regiones:

gcloud artifacts repositories create REPO\_NAME --repository-format=docker --location=REGION

Por ejemplo:

gcloud artifacts repositories create services --repository-format=docker --location=us-west1

En este caso hemos creado un repositorio llamado ‘services’ en la región us-west1.

Paso 9: Subir la imagen de Docker a Artifact Registry

Etiquetar la imagen con la dirección del repositorio de Artifact Registry: La dirección del repositorio de Artifact Registry se encuentra en el formato: REGION-docker.pkg.dev/PROJECT\_ID/REPO\_NAME. Asegúrate de tener el SDK de Google Cloud instalado y haber iniciado sesión con tu cuenta de GCP.

docker tag pandoc\_converter REGION-docker.pkg.dev/TU\_PROYECTO\_ID/REPO\_NAME/nombre-de-la-app

Reemplaza REGION con la región donde esté ubicado tu repositorio de Artifact Registry, TU\_PROYECTO\_ID con el ID de tu proyecto de GCP, REPO\_NAME con el nombre del repositorio de Artifact Registry donde deseas almacenar la imagen y nombre-de-la-app por el nombre que deseas para tu imagen en Artifact Registry.

Por ejemplo:

docker tag pandoc\_converter us-west1-docker.pkg.dev/TU\_PROYECTO\_ID/services/word\_to\_md

Paso 10: Iniciar sesión en Artifact Registry

Antes de subir la imagen, debes iniciar sesión con la herramienta de línea de comandos de Docker para que pueda autenticarse en Artifact Registry.

gcloud auth configure-docker REGION-docker.pkg.dev

Reemplaza REGION con la región correspondiente donde está ubicado tu repositorio de Artifact Registry.

## Paso 11: Subir la imagen al repositorio

Subir la imagen a Artifact Registry: Utiliza el comando docker push para enviar la imagen etiquetada a Artifact Registry.

docker push REGION-docker.pkg.dev/TU\_PROYECTO\_ID/REPO\_NAME/nombre-de-la-app

Reemplaza nuevamente REGION, TU\_PROYECTO\_ID, REPO\_NAME y nombre-de-la-app por los valores correspondientes.

Por ejemplo:

docker push us-west1-docker.pkg.dev/TU\_PROYECTO\_ID/services/word\_to\_md

## Paso 12: Desplegar la imagen de en Google Cloud Run

Para desplegar la imagen del Docker en Google Cloud Run usa el comando:

gcloud run deploy --image=REGION-docker.pkg.dev/TU\_PROYECTO\_ID/REPO\_NAME/nombre-de-la-app --platform=managed --allow-unauthenticated

Ejemplo:

gcloud run deploy --image=us-west1-docker.pkg.dev/TU\_PROYECTO\_ID/services/word\_to\_md --platform=managed --allow-unauthenticated

Esto va a darnos una salida de este estilo:

Deploying container to Cloud Run service [wordtomd] in project [tseyor2023] region [us-west1]

OK Deploying... Done.

OK Creating Revision...

OK Routing traffic...

OK Setting IAM Policy...

Done.

Service [wordtomd] revision [wordtomd-00002-sim] has been deployed and is serving 100 percent of traffic.

Service URL: https://wordtomd-zpzbiu7ooq-uw.a.run.app

Ahora podemos acceder al servicio según la URL indicada finalmente.

Una vez que el servicio esté desplegado, podrás llamarlo desde tu aplicación enviando un comando POST con el archivo .docx en el campo ‘file’ y recibirás el archivo .zip resultante en la respuesta.