Informe Técnico del Proyecto: API-SPEECH-S2S

Santiago 2025

1. Introducción

El proyecto API-SPEECH-S2S fue desarrollado con el objetivo de crear una solución automatizada que permita obtener, procesar y almacenar información extraída desde archivos de audio ubicados en Google Cloud Storage, utilizando técnicas de integración de servicios en la nube y desarrollo modular en Python. El sistema cuenta con pruebas automatizadas, contenedorización con Docker y despliegue continuo a través de Google Cloud Build.

2. Estructura Técnica del Proyecto

La estructura del proyecto **API-SPEECH-S2S** fue diseñada siguiendo una **arquitectura modular** que permite mantener el código organizado, reutilizable y fácil de escalar. Esta estructura favorece la separación de responsabilidades, asegurando que cada componente del sistema cumpla una función específica sin depender directamente de otros módulos. A continuación, se describen las principales carpetas y archivos del proyecto:

Estructura General del Proyecto

API-SPEECH-S2S/

Carpeta/Archivo	Descripción
app.py	Archivo principal que orquesta la ejecución de los módulos
requirements.txt	Lista de dependencias necesarias para el proyecto
Dockerfile	Define la imagen del contenedor Docker
cloudbuild.yaml	Configura el despliegue automático en Google Cloud
.gitignore	Archivos y carpetas ignoradas por Git
config/	Configuración general del proyecto
└── settings.py	Parámetros de conexión, rutas y ajustes globales
modules/	Módulos funcionales con la lógica de negocio
├── bigquery_table.py	Creación y carga de datos en BigQuery
├── audio_list.py	Obtención de archivos de audio desde Google Cloud Storage
└── extraction_from_audio.py	Extracción de metadatos de los archivos de audio
test/	Pruebas automatizadas con Pytest
├── test_auth.py	Verificación de autenticación y permisos con GCP

Detalles Clave de la Arquitectura:

- **app.py**: Es el punto de entrada del sistema. Este archivo coordina los pasos principales del flujo: creación/verificación de la base de datos, obtención de audios, procesamiento y carga final.
- **modules/**: Contiene la lógica del negocio organizada por función. Cada módulo puede ejecutarse de forma independiente, permitiendo pruebas y mantenimiento más ágiles.
- **test/**: Todos los módulos tienen una prueba unitaria asociada. Esto facilita el control de calidad y permite asegurar que los cambios futuros no rompan funcionalidades ya existentes.
- **config/**: Centraliza los parámetros del proyecto (por ejemplo, credenciales, rutas de almacenamiento, nombres de tablas), permitiendo ajustes rápidos sin modificar directamente el código fuente.
- **Dockerfile y cloudbuild.yaml**: Permiten contenerizar el proyecto y desplegarlo automáticamente en Google Cloud Run mediante integración continua.

3. Proceso de Desarrollo Técnico

El desarrollo del proyecto se estructuró en cuatro etapas principales, cada una orientada a cumplir objetivos específicos para garantizar la funcionalidad y calidad del sistema:

1. **Creación y configuración de la base de datos en BigQuery:** Se diseñó y creó la tabla necesaria para almacenar los metadatos extraídos de los archivos de audio.
2. **Obtención de archivos de audio desde Google Cloud Storage (GCS):** Se implementó la lógica para listar y descargar los archivos de audio alojados en GCS, asegurando el acceso seguro mediante autenticación con Google Cloud.
3. **Procesamiento de archivos para extracción de metadatos:** Se desarrollaron algoritmos para analizar cada archivo de audio y extraer la información relevante, como duración, formato, calidad, y otros datos técnicos.
4. **Carga de datos en BigQuery:** Finalmente, los metadatos procesados se cargaron en la tabla de BigQuery, permitiendo su posterior consulta y análisis.

Para garantizar la calidad y el correcto funcionamiento de cada componente, se diseñaron pruebas automatizadas con Pytest que validan el comportamiento esperado en cada etapa del proceso.

Detalle de las etapas de desarrollo

Paso 1: Estructura inicial del proyecto

- Crear la carpeta raíz del proyecto:
- mkdir API-SPEECH-S2S
- Crear un entorno virtual para aislar las dependencias.
- Generar las carpetas base necesarias (config, modules, test, docs).

Paso 2: Configuración del control de versiones

- Inicializar un repositorio Git con:
- git init
- Crear y configurar el archivo .gitignore para excluir archivos temporales y sensibles.

Paso 3: Gestión de dependencias

- Definir las librerías necesarias en requirements.txt para facilitar la instalación y reproducibilidad del entorno.

Paso 4: Implementación del punto de entrada

- Crear el archivo app.py que actúa como punto de entrada principal y servidor básico con Flask para orquestar la ejecución del sistema.

Paso 5: Desarrollo de módulos funcionales

- bigquery_table.py: módulo para crear y administrar la tabla en BigQuery.
- audio_list.py: módulo para listar y obtener archivos de audio desde GCS.
- extraction_from_audio.py: módulo encargado de procesar los archivos y extraer los metadatos.
- Cada módulo cuenta con su correspondiente archivo de pruebas (*_test.py) para validar su correcto funcionamiento.

Orden lógico de ejecución

1. Crear la tabla en BigQuery y validar su existencia.
2. Listar y obtener archivos de audio desde Google Cloud Storage.
3. Procesar los archivos para extraer metadatos y validar los resultados.
4. Cargar los datos extraídos en la tabla de BigQuery.

4. Pruebas Automatizadas con Pytest

Para asegurar la estabilidad y robustez del sistema, se implementaron pruebas automatizadas que cubren las funcionalidades clave:

- **test_auth.py**: verifica la autenticación y permisos para acceder a Google Cloud y a los buckets de almacenamiento.
- **test_sftp.py**: valida la conexión al servidor SFTP y acceso correcto a los directorios configurados.
- **audio_list_test.py**: comprueba la obtención correcta de los archivos de audio desde Google Cloud Storage.
- **extraction_from_audio_test.py**: asegura que la extracción de metadatos desde los archivos de audio se realiza correctamente.
- **bigquery_table_test.py**: verifica la existencia y estructura de la tabla en BigQuery.

Estas pruebas se ejecutan con Pytest y permiten detectar errores en etapas tempranas del ciclo de desarrollo, facilitando el mantenimiento y la calidad del software.

5. Despliegue con Docker y Google Cloud

Para facilitar la portabilidad, escalabilidad y automatización del proyecto, se utilizó **Docker** para contenerizar la aplicación, y **Google Cloud Build** junto con **Cloud Run** para automatizar el despliegue en la nube.

Contenerización con Docker

- **Dockerfile:** Se creó un archivo Dockerfile que define el entorno de ejecución necesario para la aplicación, incluyendo el sistema operativo base, instalación de dependencias y configuración del servidor Flask.
- La imagen Docker resultante contiene todo lo necesario para ejecutar el proyecto de forma aislada y reproducible.

Gestión y despliegue en Google Cloud

- **Pruebas locales:** La imagen Docker se construyó y probó localmente para asegurar que la aplicación funcionaba correctamente en el contenedor.
- **Google Container Registry (GCR):** Una vez validada, la imagen se etiquetó y se subió al Google Container Registry usando la interfaz de línea de comandos gcloud.
  Ejemplo de comandos:
- docker build -t gcr.io/[PROJECT-ID]/api-speech-s2s:latest .
- docker push gcr.io/[PROJECT-ID]/api-speech-s2s:latest
- **Automatización con Cloud Build:** Se configuró el archivo cloudbuild.yaml para automatizar la construcción y despliegue de la imagen en Google Cloud Run cada vez que se realiza un push al repositorio. Esto facilita la integración continua y despliegue continuo (CI/CD).

Despliegue en Cloud Run

- El servicio se desplegó en **Cloud Run**, un entorno serverless que ejecuta contenedores Docker con escalabilidad automática.
- El comando principal para desplegar la aplicación es:
- gcloud run deploy api-speech-s2s \
- --image gcr.io/[PROJECT-ID]/api-speech-s2s:latest \
- --platform managed \
- --region [REGION] \
- --allow-unauthenticated
- Esto permite exponer la aplicación a internet con alta disponibilidad y escalabilidad automática según la demanda.

6. Resumen Técnico del Proyecto

El proyecto cumplió con todos los objetivos técnicos planteados inicialmente, garantizando un desarrollo robusto, modular y escalable. Los aspectos más relevantes alcanzados son:

- **Arquitectura modular:** La aplicación se estructuró en módulos claros y separados, facilitando mantenimiento, escalabilidad y comprensión del código.
- **Pruebas automatizadas:** Se implementaron pruebas unitarias y de integración con Pytest para asegurar la calidad y correcto funcionamiento de cada componente.
- **Despliegue automatizado:** Se utilizó Docker para contenerizar la aplicación y Google Cloud Build para automatizar la construcción y despliegue en Cloud Run, garantizando un flujo de integración y entrega continua (CI/CD).
- **Integración con servicios en la nube:** El proyecto integra eficazmente servicios clave como BigQuery (para almacenamiento y consulta de datos), Google Cloud Storage (para manejo de archivos de audio) y SFTP (para la transferencia segura de archivos).
- **Escalabilidad y portabilidad:** Gracias a la contenerización y el uso de Cloud Run, la solución es altamente portable y puede escalar automáticamente según la demanda.

Este enfoque integral asegura una base sólida para futuras mejoras y la evolución del proyecto.

7. Recomendaciones Técnicas

Para mejorar la calidad, seguridad y mantenibilidad del proyecto, se sugieren las siguientes acciones:

- **Implementar autenticación OAuth2:**
  Añadir un sistema de autenticación basado en OAuth2 para asegurar el acceso a los recursos y mejorar la seguridad en la comunicación con servicios externos.
- **Añadir validaciones más robustas para archivos de audio:**
  Incorporar controles más estrictos sobre el formato, tamaño y contenido de los archivos de audio para evitar errores durante el procesamiento y mejorar la integridad de los datos.
- **Integrar un sistema de logging centralizado:**
  Utilizar herramientas de logging centralizado (por ejemplo, Stackdriver Logging en Google Cloud) para registrar eventos, errores y métricas, facilitando la monitorización y diagnóstico en producción.
- **Ampliar la cobertura de pruebas automatizadas:**
  Desarrollar pruebas adicionales que cubran casos límite, errores esperados y escenarios de integración, aumentando la confiabilidad del sistema ante distintos contextos.
- **Documentar la API con Swagger (OpenAPI):**
  Crear documentación interactiva y estandarizada para los endpoints de la API usando Swagger, mejorando la comunicación con otros desarrolladores y facilitando futuras integraciones.

8. Conclusiones

El proyecto **API-SPEECH-S2S** representa una solución técnica sólida y escalable para el procesamiento automatizado de archivos de audio en la nube. Gracias a su arquitectura modular y uso de tecnologías modernas como Docker, Google Cloud Storage y BigQuery, el sistema ofrece un desempeño eficiente y confiable.

Además, la incorporación de pruebas automatizadas y un proceso de despliegue continuo garantiza la calidad y facilita la evolución futura del proyecto. La estructura diseñada permite integrar fácilmente nuevas funcionalidades, tales como análisis avanzado mediante inteligencia artificial o dashboards web para la visualización de resultados.

En resumen, este proyecto no solo cumple con los objetivos iniciales, sino que también sienta las bases para futuras mejoras y ampliaciones, asegurando su relevancia y utilidad en entornos productivos.