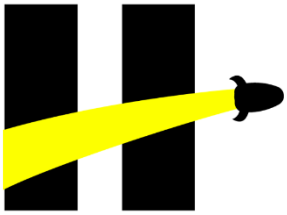


PROYECTO GRUPAL SOY HENRY

ALERTA DE SISMOS

INGENIERIA DE DATOS

INTEGRANTES:

CRISTIAN LEONARDO ZAMUDIO

EDGARD ALFREDO HUANCA

JUAN MANUEL LUNA

DAVID ALEJANDRO ORTIZ GARCIA

LIZNEL OSWALDO GRIMALDO

FASE DE INGENIERIA DE DATOS:

En esta fase se realizaron los procesos necesarios para implementar la infraestructura y adoptar las tecnologías que permiten la extracción transformación y carga de datos de forma automatizada. Para esto se encuentran disponibles dos servicios que cumplen el objetivo, pero con características diferentes. Adicionalmente se pone en producción una API para insertar usuarios y registrarlos en una base de datos.

OBJETIVOS

- Implementar la infraestructura tecnológica que permita obtener una base de datos de sismos estandarizada de los países de Estados Unidos, Japón y Chile, que contenga la información de los sismos que se registran en esos países, y que se actualice de forma automática en tiempo real.

- Producir una API que permita realizar el registro de usuarios interesados en recibir alertas de sismos y recomendaciones según corresponda en su zona de residencia.

Para cumplir con los objetivos se presenta la documentación de los servicios propuestos y se espera que el PO se decline por uno de ellos después de revisarlos y definir cual se ajusta más a sus necesidades.

SERVICIO 1 MICROSERVICIO RAILWAY:

La primera instancia es traer conseguir la información de los sismos de los países de Estados Unidos, Japón y Chile. Para esto nos basamos en las APIs que dispone cada página, en el caso de Estados Unidos estuvo disponible desde un principio. Ya en el caso de Japón y Chile no fue lo mismo.

Para obtener la información de los sismos de estos países en tiempo real, teníamos dos opciones. La primera fue realizar un webscraping y la segunda buscar la API que alimente a estas páginas. Se optó por la segunda opción debido a la rapidez en la obtención de los datos que tenemos a través de esta metodología.

Una vez que obtuvimos los datos, el siguiente paso era realizar el ETL para poder estandarizar la información y que el análisis en la posterior etapa sea consistente.

El proceso de ETL se realizó con diversas librerías, entre las principales están: Pandas, Numpy, Requests, Json, entre otras.

Para tener una base de datos única era necesario que todos los países brindaran la misma información, por lo que se decidió contar solo con las columnas de “time”, “latitude”, “longitude”, “depth”, “mag” y “place”.

Si bien todas las APIs contaban con esta información, no tenían la misma estructura, por lo que hubo que realizar tareas de limpieza diferentes.

En el caso de Estados Unidos, se eliminaron columnas innecesarias y se cambió el formato de la fecha y hora para mantener la consistencia en los tres datasets.

En el caso de Japón, aquí se tuvo un poco más de trabajo, no solo se tuvo que cambiar el formato de fecha y hora, sino que también, se eliminaron varias columnas innecesarias, como así también extraer información de una columna (columna “cod”) para poder dividirla en tres columnas diferentes (pasó a ser: “latitude”, “longitude” y “depth”).

Finalmente se prosiguió con Chile, en donde toda la información estaba en una sola columna, por lo que se extrajo de esa columna toda la información necesaria, dando como resultado las 6 columnas nombradas anteriormente.

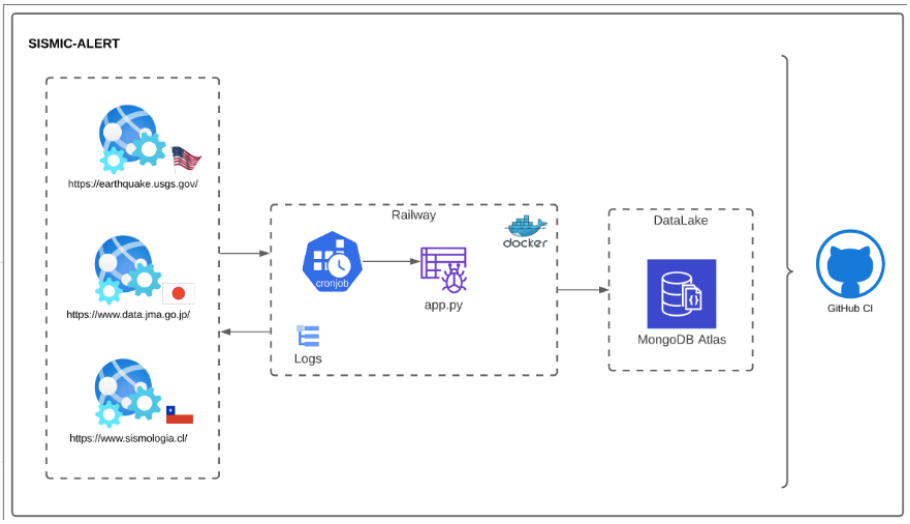
Con las bases de datos estandarizadas, se les agregó una columna extra a cada dataset con el nombre del país al que pertenecen, con el fin de identificarlos. Y para finalizar, la unificación de todos los datasets en uno solo.

Una vez terminado el ETL el objetivo siguiente era realizar el pipeline con la respectiva automatización, llevando toda la información a una base de datos en la nube en Mongo DB.

Para esto se plantearon dos caminos, el primero haciéndolo desde un entorno local con y desplegándolo en RailWay a través de un contenedor de Docker y el otro a través de Google Cloud.

Debido a nuestra duda en cuál de estas opciones se iba a acomodar a las necesidades y preferencias de nuestro cliente, se optó por seguir con ambas y dejar a elección a nuestro cliente.

La primera opción se realizó a través de Docker, como se dijo en un entorno local, colocando dentro del contenedor dos archivos:



- App.py: que incluye la extracción de la información de las APIs de cada país (“get_last_quake”), el ETL de cada país (“get_quake_by_country”), la conexión a Mongo BD Atlas a través del link brindado por este (a través del archivo db.py) y las funciones de guardado de la información a Mongo DB Atlas (“save_mongo”) y la que permite el envío solo del último sismo registrado para evitar duplicar la información (“check_not_exist” y “get_last_quake”).

```python
def get_last_quake():
    countries={'usa':'https://earthquake.usgs.gov/fdsnws/event/1/query?format=csv&orderby=
    for country in countries:
        quake=get_quake_by_country(country, countries[country])
        if check_not_exist(quake['time'],quake['country']):
            save_mongo(quake)
            print('1 new quake for "{0}" added'.format(quake['country']))
        else:
            print('0 new quake for "{0}" added'.format(quake['country']))

def check_not_exist(time, country): #Verify if quake exists for 'country'
    if conn.sismology.quakes.find_one({'time':time,'country':country})!=None:
        return False
    else:
        return True

def save_mongo(quake): #Save into mongo latest quake
    #Inserting document
    db=conn['sismology']
    collection=db['quakes']
    collection.insert_one(quake)
```

- La automatización realizada de manera local a través de Linux (cronworker.sh)

```
#!/bin/bash
while true; do
        python3 app.py
        sleep 60
done
```

Para poder realizar esta tarea de manera automática se lo desplegó en RailWay.

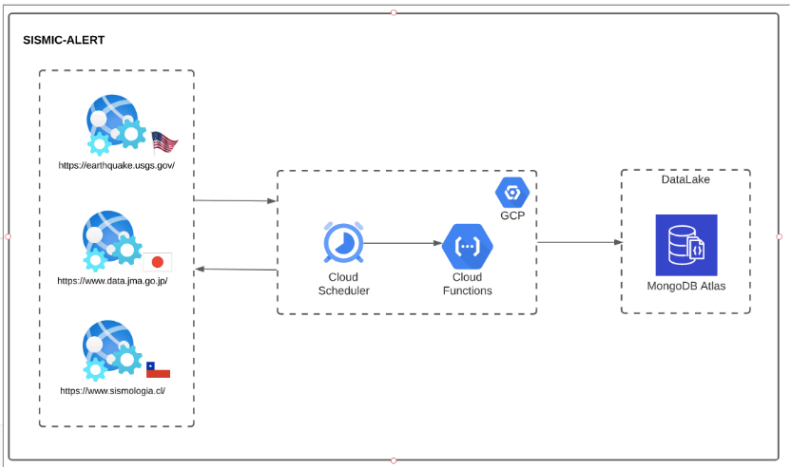
Ventajas

- Alta disponibilidad (Mediante orquestación con Kubernetes | paralelización en servidores on-premise)
- Costo
- CI/CD
- Escalabilidad (Puede ejecutarse en instancias de AWS, Azure o GCP)

Desventajas

- Complejidad en el mantenimiento

SERVICIO 2 GOOGLE CLOUD PLATFORM:



Por esta vía el camino es un poco diferente. El primer paso es configurar el Mongo DB Atlas, crear la base de datos y la colección. En nuestro caso, se llaman “Alert_Sismic_Last” la base de datos y “Last” la colección.

Pasamos ahora a configurar el “network access” en donde colocamos la IP para dar acceso al Mongo.

El próximo paso es tomar el proceso de ETL descrito más arriba en este trabajo junto con las APIs (“cloud-function.py”), con el fin de tener un dataset estandarizado y unificado.

Una vez obtenidos los datos, es hora de automatizarlos y la opción elegida en este camino es Google Cloud, el cual nos provee varias herramientas, de las cuales usamos Google Functions y Google Scheduler.

Una vez ingresado a Google Cloud, lo primero que tenemos que hacer es ir a Google Functions y configuramos nuestro Activador, dejando la opción de HTTP para que el Activador se ejecute en la nube.

Una vez configurado el Activador, pasamos al código, donde aquí se puede colocar los scripts de Python realizados anteriormente. Para esto, debemos seleccionar el entorno de ejecución, el cual el que mejor se adaptó a nuestras necesidades fue el Python 3.7. Seleccionamos “editor directo” en código fuente, lo que nos muestra dos opciones. La primera, “main” que es donde colocamos nuestros scripts desarrollados en la etapa de ETL, incluyendo los enlaces de las APIs de los 3 paises, y además una función que nos conecta a Mongo DB Atlas.

Si bien hemos usado las mismas funciones que en el ETL, aquí toca adaptarlas un poco, ya que no necesitamos que envíe los 3 datasets completos cada vez que Google Cloud envíe información a Mongo sino, solo los últimos sismos a medida que van ocurriendo, por lo que la base de datos unificadas se reducen solo a los últimos 3 registros de cada país, y con una última función, se ingresa solo aquel que ya no esté dentro de la base de datos ya declarada en Mongo, con la siguiente parte del código:

```python
# Validar y filtrar terremotos con la misma fecha
for record in combined_df.to_dict('records'):
    date = record['time']
    existing_record = collection.find_one({'time': date})
    if existing_record is None:
        # Insertar el registro en la colección de MongoDB
        collection.insert_one(record)
```

Ya la segunda opción nos pide las librerías utilizadas en nuestro script, en este caso:

```
Presiona Alt + F1 para ver las opciones de accesibilidad.
1   # Function dependencies, for example:
2   # package>=version
3   pymongo>=4.3.3
4   requests>=2.28.1
5   pandas
6   numpy
```

Y finalmente nos queda la opción de punto de entrada, donde tenemos que colocar “main” para que nos tome la información del archivo “main” donde están los códigos de Python.

Una vez ejecutados estos pasos, solo nos queda hacer click en la opción “implementar” donde después de unos minutos, nos dice si está todo ok, y podemos proceder a la opción “activador”, y aquí entramos al enlace, ya configurado anteriormente, y nos fijamos si esta todo bien o nos puede mostrar algún error. En caso de que alguno aparezca, es necesario entrar a la opción de “registros” para verificar cual es el error, para poder solucionarlo.

Una vez realizados estos pasos, nuestro Google Cloud ya está configurado y disponible para enviar información cuando nosotros se lo pidamos.

Ahora nos toca entrar al Google Scheduler para configurar la programación de cuándo se va a ejecutar de manera automática.

Para configurar la frecuencia de la ejecución automática, tenemos que hacerla mediante el lenguaje CRON, el cual podemos traducir mediante la página: https://crontab.cronhub.io/

En nuestro caso, lo configuramos para que se ejecute de a 1 minuto.

Ventajas:

- Escalabilidad
- Alta disponibilidad
- Administración automática del servidor
- Integración con otros servicios de Google Cloud
- Monitoreo y registro

Desventajas

- Costo

API REGISTRO DE USUARIOS:

Dentro de la infraestructura necesaria para el proyecto se requiere una base de datos de los usuarios que recibirán las alertas de los sismos en su zona de residencia. Para esto se dispuso de una API que registra usuarios.

Para el desarrollo de este API se usaron las siguientes tecnologías:

Python con librerías FastApi y Uvicorn.

MongoDB atlas (servicio web de tratamiento de bases de datos de MongoDB)

Github (repositorio que contiene la API)

Render.com (Servicio web para despliegue de APIs REST)

Como primera instancia en el desarrollo de la API se definieron las necesidades de la base de datos de usuarios donde se definen los siguientes campos:

```
class User(BaseModel):
    name: str
    email: str
    phone_number : str
    ref_geographic: str
```

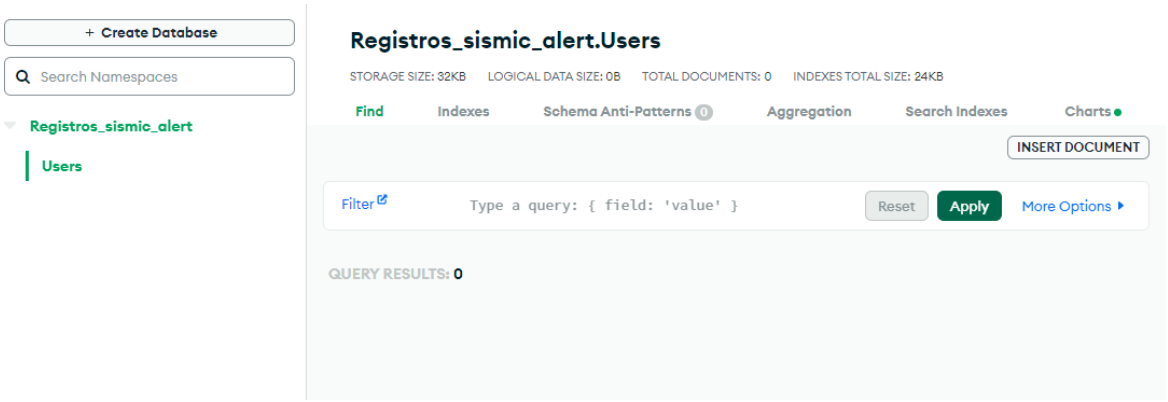
Con estos campos se obtiene la información de cada usuario para posteriormente poder ser usada en el sistema de alertas.

Se definen los endpoints para la API, esta API permite el tratamiento C.R.U.D. de la base de datos para usuarios registrados.

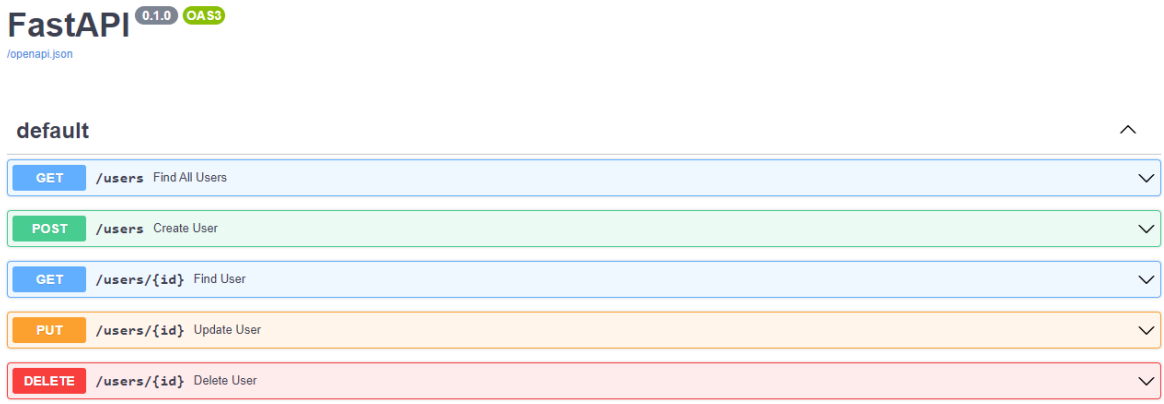
```python
@user.get('/users')
def find_all_users():
    return usersEntity(collection.find())

@user.post('/users')
def create_user(user: User):
    new_user = dict(user)

    collection.insert_one(new_user).inserted_id

    return {"message": "Datos guardados exitosamente"}


@user.get('/users/{id}')
def find_user():
    return "id"

@user.put('/users/{id}')
def update_user():
    return "id"

@user.delete('/users/{id}')
def delete_user():
    return "id"
```

Con esos endpoints podemos crear, obtener, actualizar o eliminar registros de usuarios de la API.

Los registros dentro de esta base de datos deben estar disponibles en un servidor web para poder acceder a ella desde los servicios cloud que se implementaran para el sistema de alertas. Por esa razón la API está conectada a MongoDB atlas y allí se guardan los datos registrados por la API.



Para que esta API este disponible en cualquier momento se utilizo el servicio de RENDER que permite alojar en sus servidores web la api y la disponibliliza para cualquier usuario.



El enlace de la API en Render es el siguiente:

https://base-de-usuarios-sismic-alert.onrender.com/docs

para finalizar este un diagrama de la arquitectura de la API