Challenge Data Engineer

Felipe Bahamonde

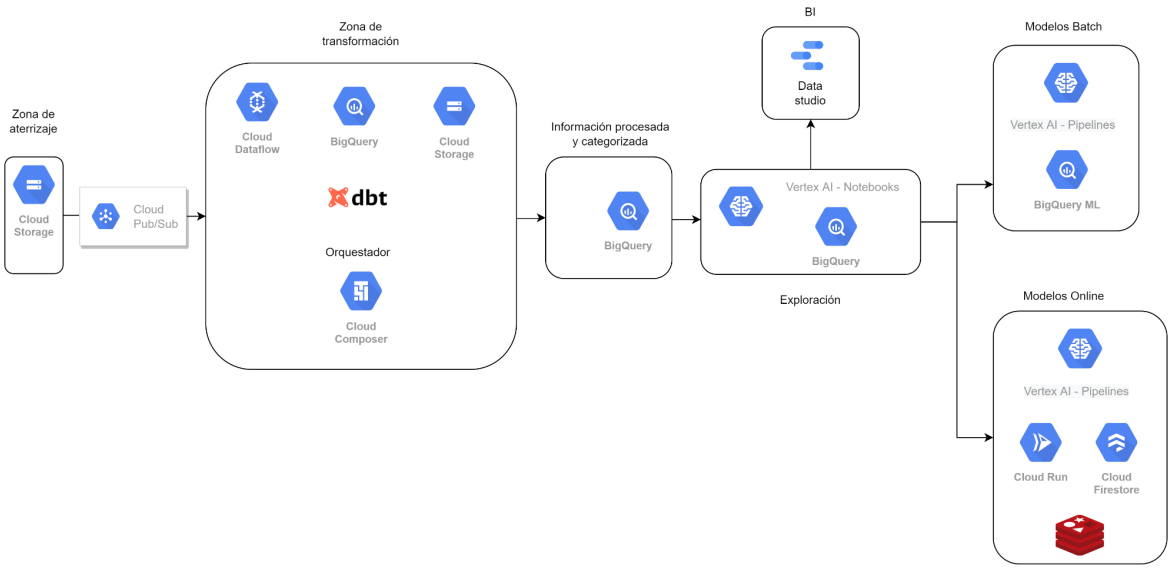
Diseño y Arquitectura

R:

Según los requerimientos estipulados:

Al día de hoy difícilmente se justifica mantener todo en servidores on-prem, por tanto el data warehouse a generar debiese vivir en la nube, alimentarse de los sistemas legados, fuentes de información que vivan en la nube y servicios de terceros (ej Google Analytics, Freshdesk, CleverTap).

Por comodidad y experiencia se propone el siguiente esquema basado en **GCP**



En donde a grandes rasgos se plantea lo siguiente (de izquierda a derecha):
1.- Zona de “aterrizaje” donde llega la información de sistemas externos y/o on-prem. Esta etapa se basa en almacenamiento de archivos (Google Cloud Storage).

2.- Una vez que llega la información, se desencadenan procedimientos mediante PubSub o orquestados por algún servicio tipo Airflow o similar según la necesidad (si se necesita stream de datos PubSub y luego Dataflow, caso contrario programado en un DAG en Composer o en DBT).

3.- Esa información es transformada y limpiada según lo exijan las reglas de negocio para luego ser inyectada en Bigquery. Es importante considerar que gran parte de las fuentes de información que se mencionaron en el documento de requerimiento poseen conectores directos a Bigquery, por lo tanto algunas de esas fuentes seguirán otros flujos (Ej: Google

Analytics). Otro punto relevante a recalcar es que debe existir un claro gobierno sobre los datos, que dictamine las directrices a seguir.

4.- Una vez que la información se encuentra en Bigquery, los diferentes usuarios ya son capaces de consumirla para los distintos análisis o reportes que deban hacer. Aquí se considera solo Data Studio por simplicidad en conexiones y dado que no requiere de una licencia adicional, pero si la compañía ya cuenta con una plataforma de reportería, se puede seguir utilizando.

5.- En caso de que analistas o científicos de datos quieran realizar un análisis más detallado, se les habilitaran Notebooks en AI platform, con todos los recursos y conexiones necesarias para consumir la información.

6.- Para que los científicos de datos puedan exponer sus modelos, se ofrecen 2 soluciones:

- Batch: Creación de pipelines en Vertex AI, que permitan hacer predicciones en formato batch y devolverlas a Bigquery o algún otro formato para que los sistemas puedan ir a buscar la información (FTPs, Storage, entre otros). Es importante destacar que esos pipelines permiten también manejar el ciclo de vida completo del modelo, agregando funcionalidades de monitoreo y características de MLops. Otra alternativa interesante y que permite democratizar aún más el modelamiento es bigquery ML, dado que se escribe en SQL.

- Online: Para la predicción en tiempo real, se considera Vertex AI nuevamente junto a Cloud Run, que permite exponer API fácilmente y sin administrar infraestructura. Redis/Firestore se consideran como formas de disponibilizar las predicciones y que puedan ser consumidas por otros servicios (Look up Tables), así como también una manera práctica y rápida de disponibilizar features que podrían ser necesarias para realizar la predicción.

*Consideraciones adicionales:

- Al encontrarse todo en GCP, Stackdriver o Logging (servicio que guarda un registro de todo lo que ocurre dentro de un proyecto) puede ser usado como herramienta de monitoreo, generando alertas en caso de comportamientos anómalos, gastos excesivos, fallas en los procesos, entre otros)

- Gran parte de los servicios mencionados se basan en contenedores.

- El esquema no considera “ir” a buscar información a otros sistemas, sino que los otros sistemas puedan depositar la información de manera ordenada en GCS. Este punto se puede flexibilizar y modificar según la necesidad.

Implementación de soluciones

R:

Para la implementación del modelo se considera nuevamente GCP. Antes de comenzar deberíamos tener al menos las siguientes preguntas resueltas:

- ¿Cuál es el caso de uso asociado?
- ¿Cómo va a ser consumido? Real Time - Batch
- ¿Cúal es la frecuencia en que se va a consumir?
- En el caso de que sea como API Real Time, ¿El sistema solicitante tiene la capacidad de envíar todos los campos necesarios?
- ¿Cómo debe responder el sistema en caso de que falten datos y/o el input tenga errores?

Dado que no manejamos toda esa información, vamos a tomar los siguientes supuestos:

- Se necesita Real time y Batch.
- Dado que hace referencia a clientes, a lo más puede ser consumido una vez al día (los usuarios no son tan dinámicos) y como cota superior no debiese superar las 20M de consultas diarias (Caso hipotético en que el banco tuviese al 100% de la población del país como cliente)
- En la consulta van a estar incluidos todos los campos necesarios para que el modelo pueda realizar la predicción.

Bajo esas consideraciones, el camino a seguir va a ser levantar un par de endpoints en Cloud Run, utilizando python como lenguaje y FastApi como biblioteca para levantar las APIs.

Funcionamiento predicción Online (single prediction):

- Recibe todos los campos/features necesarios para realizar la predicción.
- El modelo serializado se almacena dentro del contenedor para poder ser utilizado con la menor latencia posible.
- Respuesta es devuelta en la misma consulta, en donde va la probabilidad asociada a ambas clases (1 y 0).

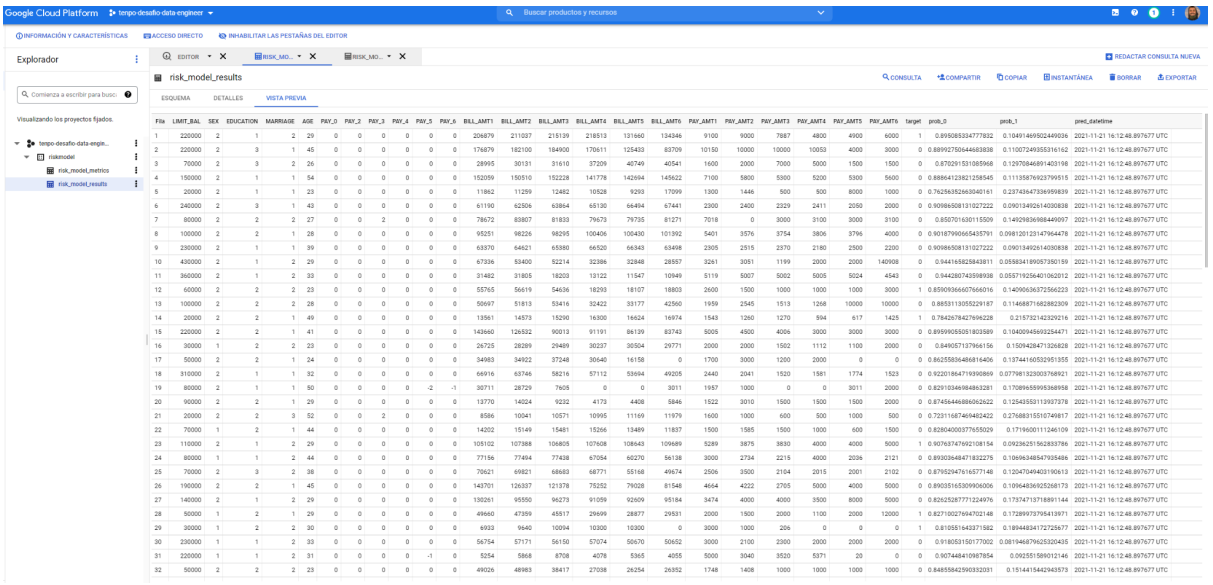
```python
@app.post("/risk-online-prediction", description="Risk model: Online Prediction")
async def risk_online_pred(inputs: OnlinePredInput):
    try:
        input_df = pd.DataFrame([jsonable_encoder(inputs)])
        features_df = input_df[[col for col in input_df.columns if col not in ['TARGET','BATCH_FLAG']]]
        result = model.predict_proba(features_df)[0]

        if inputs.BATCH_FLAG == 1:
            return result.tolist()
        else:
            return JSONResponse(content=json.dumps(result.tolist()))

    except Exception as e:
        logging.error(e)
        raise HTTPException(
            status_code=500, detail="Internal prediction error"
        )
```

Funcionamiento predicción Batch:

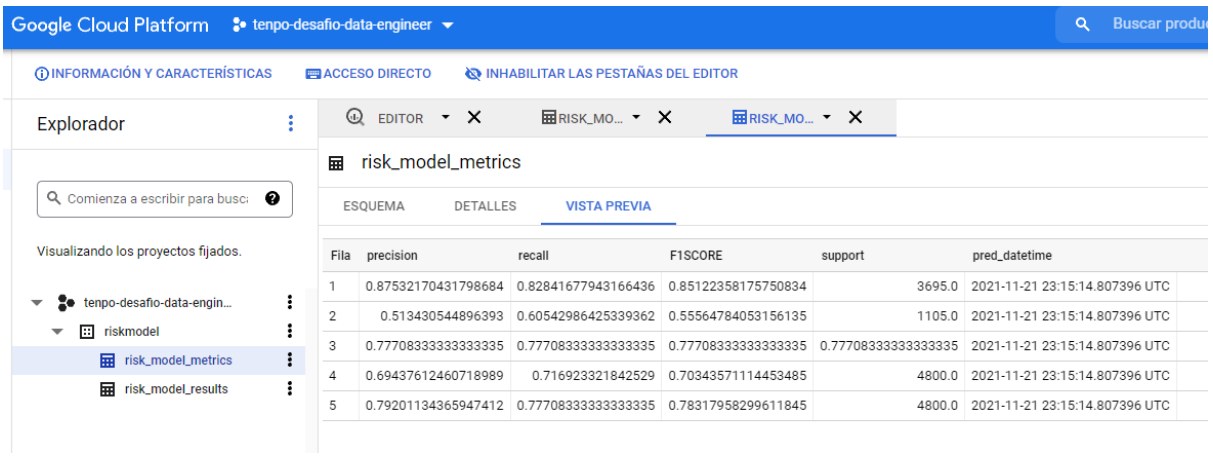
- Recibe una ubicación en Google Cloud Storage que contiene el json de entrada que representa varios casos a predecir con todas las columnas/features.
- Para reutilizar lo desarrollado en la predicción online, se toma cada fila del archivo json y se le entrega de manera asíncrona a la función de single prediction. Una vez que todos las predicciones son devueltas, se almacenan como tabla en Bigquery para que el sistema que lo necesite lo pueda consultar a futuro.
- En caso de que el archivo json contenga el valor real de la predicción, se calculan las métricas de precision - recall - F1 score & support y son almacenadas de igual forma en Bigquery.

```python
@app.post("/risk-batch-prediction", description="Risk model: Batch Prediction")
async def risk_batch_pred(inputs: BatchPredInput):
    try:
        json_loaded = get_json(inputs.BUCKET_NAME, inputs.FILEPATH)
        input_df = pd.json_normalize(json_loaded)
        async with ClientSession() as session:
            response = await asyncio.gather(*[risk_online_pred(OnlinePredInput(BATCH_FLAG = 1,**input)) for input in json_loaded])
        df_response = pd.DataFrame(response, columns = ['prob_0', 'prob_1'])
        df_final = pd.merge(input_df, df_response, left_index=True, right_index=True)
        now = pd.to_datetime('now')
        df_final['pred_datetime'] = now
        upload_to_bq(df_final, "tenpo-desafio-data-engineer.riskmodel.risk_model_results", 'WRITE_APPEND')
        try:
            metrics = get_metrics(df_final)
            df_metrics = pd.DataFrame(metrics).transpose()
            df_metrics['pred_datetime'] = now
            df_metrics.rename(columns={"f1-score": "F1SCORE"}, inplace = True)
            upload_to_bq(df_metrics, "tenpo-desafio-data-engineer.riskmodel.risk_model_metrics", 'WRITE_APPEND')
        except Exception as e:
            logging.info(' Column Target was not detected')
        logging.error(e)
        return('Done')
    except Exception as e:
        logging.error(e)
        raise HTTPException(
            status_code=404, detail="File not found"
        )
```

La tabla en Bigquery con el resultado de la predicción se ve de la siguiente manera:



y la tabla que contiene las métricas asociadas al desempeño del modelo



Fila	precision	recall	F1SCORE	support	pred_datetime
1	0.87532170431798684	0.82841677943166436	0.85122358175750834	3695.0	2021-11-21 23:15:14.807396 UTC
2	0.513430544896393	0.60542986425339362	0.55564784053156135	1105.0	2021-11-21 23:15:14.807396 UTC
3	0.77708333333333335	0.77708333333333335	0.77708333333333335	0.77708333333333335	2021-11-21 23:15:14.807396 UTC
4	0.69437612460718989	0.716923321842529	0.70343571114453485	4800.0	2021-11-21 23:15:14.807396 UTC
5	0.79201134365947412	0.77708333333333335	0.78317958299611845	4800.0	2021-11-21 23:15:14.807396 UTC

Todo el codigo asociado se encuentra en el siguiente repositorio
https://github.com/fbahamonde/tenpo-desafio-data-engineer

Hay una instancia de Cloud Run disponible para consultar (**solo para efectos de probar la solución, en un entorno real debería considerar acceso a través de cuentas de servicio**): https://risk-model-n7vbibqkga-uc.a.run.app/docs

Probar Batch:

```
curl -X 'POST' \
  'https://risk-model-n7vbibqkga-uc.a.run.app/risk-batch-prediction' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "BUCKET_NAME": "risk-model-tenpo",
  "FILEPATH": "datos_prueba.json"
}'
```

Probar Single:

```
curl -X 'POST' \
  'https://risk-model-n7vbibqkga-uc.a.run.app/risk-online-prediction' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
 "LIMIT_BAL": 0,
 "SEX": 0,
 "EDUCATION": 0,
 "MARRIAGE": 0,
 "AGE": 0,
 "PAY_0": 0,
 "PAY_2": 0,
 "PAY_3": 0,
 "PAY_4": 0,
 "PAY_5": 0,
 "PAY_6": 0,
 "BILL_AMT1": 0,
 "BILL_AMT2": 0,
 "BILL_AMT3": 0,
 "BILL_AMT4": 0,
 "BILL_AMT5": 0,
 "BILL_AMT6": 0,
 "PAY_AMT1": 0,
 "PAY_AMT2": 0,
 "PAY_AMT3": 0,
 "PAY_AMT4": 0,
 "PAY_AMT5": 0,
 "PAY_AMT6": 0,
 "BATCH_FLAG": 0
}'
```

Bonus Points:

Rendimiento API: Para el caso de single predicción, las consultas no exceden los 300ms. Si bien la instancia de Cloud Run se encuentra limitada a solo 1 (para efectos del desafío), se puede incrementar la cantidad para atender más consultas en paralelo.

En el caso de batch, el proceso puede tardar hasta 20 segundos para los 4500 datos de prueba, esto considerando que tiene que ir a leer a storage y luego escribir en 2 tablas. Entendiendo que es un proceso batch y no requiere de una respuesta inmediata, la solución cumple con lo esperado.

Rendimiento Modelo con nuevo set de datos: Tal y como se muestra en la tabla con métricas expuesta anteriormente, los resultados se asemejan bastante a los del notebook en cuanto a performance, por lo que no hay mayor comentario al respecto. En primera instancia podemos decir que los datos fueron separados en distintos sets de manera correcta.

Posibles mejoras:

- Mantener un caché por si en un corto periodo consultan a la misma persona

- Cambiar la función batch para optimizar tiempo. En la implementación actual toma cada fila y se pasa por el modelo de manera independiente, una alternativa que podría resultar más rápida es tomar todos los registros y pasarlos por el modelo al mismo tiempo.
- Los registros no tienen llave ni identificador, se debiese incluir algún tipo de identificador para facilitar la utilización de los datos en batch.
- Pruebas de performance (API) en mayor detalle, utilizar locust o algún framework similar para medir respuesta.

Notas al margen: Me gustó bastante el problema planteado, pero creo que escapa del dominio de un Ingeniero de Datos y empieza a mezclarse con un Machine Learning Engineer.