**Arquitectura de Datos, BI y ML en GCP para un Banco**

1. **Definición del problema**

Un banco busca fortalecer su capacidad para **detectar fraudes en tiempo real** en transacciones con tarjetas, integrando al mismo tiempo una visión de **Business Intelligence (BI)** sobre el comportamiento de sus clientes. Actualmente, el banco dispone de flujos de datos en **tiempo real** provenientes de las transacciones con tarjetas (streaming) y de fuentes **batch** que contienen información estructurada del **maestro de clientes y productos**, así como del **catálogo de comercios**.

**Objetivos del proyecto**

1. **Detección temprana de fraude:** Implementar un modelo de analítica predictiva capaz de evaluar en tiempo real la probabilidad de fraude en cada transacción.
2. **Integración de datos:** Diseñar una arquitectura que combine flujos de datos *streaming* (transacciones) con datos *batch* (clientes, productos y comercios) de forma eficiente y segura.
3. **Visualización y monitoreo:** Desarrollar un tablero BI con indicadores clave que permitan monitorear la actividad de los clientes, las tendencias de fraude y la efectividad del modelo predictivo.
4. **Optimización operativa:** Reducir falsos positivos en la detección de fraude para minimizar el impacto en clientes legítimos.

**KPIs (Indicadores Clave de Desempeño)**

| **Categoría** | **Indicador** | **Descripción / Meta** |
| --- | --- | --- |
| **Eficacia del modelo** | *Tasa de detección de fraude (Recall)* | ≥ 90% de fraudes reales identificados. |
|  | *Tasa de falsos positivos (FPR)* | ≤ 5% de transacciones legítimas marcadas como sospechosas. |
| **Tiempo de respuesta** | *Latencia promedio del sistema* | < 2 segundos por evaluación de transacción. |
| **Valor financiero** | *Monto total de fraude evitado* | Suma en USD de operaciones fraudulentas bloqueadas. |
| **Rendimiento operativo** | *Número de alertas procesadas por hora* | Capacidad de detección en flujo continuo. |
| **Inteligencia de negocio** | *Volumen de transacciones diarias / mensuales* | Seguimiento del comportamiento financiero de clientes. |
|  | *Tasa de crecimiento de clientes activos* | Variación porcentual mensual. |

1. **Diseño de Arquitectura**

Ver archivo draw.io 🡪 Diseño de Arquitectura

Ver archivo .png 🡪 Diseño de Arquitectura

1. **Justificación Técnica**
2. **Ingesta de Datos**

Pub/Sub (Streaming)

* Latencia: baja, ideal para procesar eventos en tiempo real de transacciones de tarjetas.
* Elasticidad: Escala automáticamente con el volumen de mensajes sin intervención manual.
* Costo: Se cobra por volumen de mensajes y ancho de banda (barato para millones de eventos).
* Facilidad de operación: Totalmente administrado, sin servidores que mantener ni particiones que ajustar.

Cloud Storage (Batch)

* Latencia: No es crítico, ya que se usa para cargas diarias.
* Elasticidad: Escala de forma ilimitada para almacenar datasets históricos o diarios.
* Costo: Uno de los servicios más económicos por GB almacenado (y sin costo por inactividad).
* Facilidad de operación: Integración directa con Dataflow, Composer y BigQuery; manejo simple mediante APIs o consola.

1. **Procesamiento**

Dataflow (Apache Beam)

* Latencia: Procesamiento en streaming casi en tiempo real (< segundos por evento).
* Elasticidad: Escala dinámica según throughput de datos (auto-scaling gestionado).
* Costo: Se paga por recursos consumidos (vCPU + memoria); costo controlable por ventanas de tiempo.
* Facilidad de operación: Completamente administrado, compatible con Python/Java y con monitoreo nativo en Cloud Monitoring.

Cloud Composer (Airflow)

* Latencia: No aplica (procesos batch programados).
* Elasticidad: Permite ejecutar DAGs concurrentes, escalando workers según demanda.
* Costo: Basado en número de workers y tiempo de ejecución.
* Facilidad de operación: Simplifica la orquestación y dependencias entre procesos ETL, con interfaz gráfica y logs integrados.

1. **Almacenamiento y Analytics**

BigQuery

* Latencia: Consultas en segundos incluso sobre terabytes gracias a arquitectura columnar y ejecución distribuida.
* Elasticidad: Escala automáticamente el cómputo según tamaño de la consulta.
* Costo: Paga solo por almacenamiento y por bytes procesados; sin infraestructura fija.
* Facilidad de operación: No requiere indexación ni particiones manuales; integración nativa con Looker, Vertex AI y Dataflow.

BigQuery BI Engine

* Latencia: Acelera dashboards hasta sub-segundos mediante caché en memoria.
* Elasticidad: Se expande dinámicamente según tamaño de los datasets BI.
* Costo: Costo adicional bajo, basado en memoria reservada.
* Facilidad de operación: Integración nativa con Looker y Looker Studio, sin configuración adicional.

Looker / Looker Studio

* Latencia: Optimizada por BI Engine (consulta en segundos).
* Elasticidad: Maneja múltiples usuarios concurrentes sin necesidad de servidores dedicados.
* Costo: Looker Studio es gratuito; Looker es licenciado por usuario.
* Facilidad de operación: Interfaces visuales intuitivas, gobernanza centralizada y conexión directa a BigQuery.

1. **Machine Learning (Vertex AI)**

Vertex AI Workbench / AutoML

* Latencia: En entrenamiento no crítica; en inferencia depende del endpoint.
* Elasticidad: Escala horizontalmente para entrenar modelos grandes o AutoML concurrentes.
* Costo: Pago por uso de notebooks, GPUs o AutoML jobs (se puede pausar).
* Facilidad de operación: Entorno unificado (Jupyter administrado) con conexión nativa a BigQuery y Model Registry.

Vertex AI Model Registry

* Latencia: No aplica (metadatos).
* Elasticidad: Repositorio centralizado y escalable.
* Costo: Sin costo adicional.
* Facilidad de operación: Permite versionar modelos, auditar y controlar despliegues fácilmente.

Vertex AI Endpoints

* Latencia: Inferencia en milisegundos; ideal para scoring online en tiempo real.
* Elasticidad: Escalado automático según tráfico.
* Costo: Pago por hora de instancia activa; escalado a cero disponible.
* Facilidad de operación: Despliegue directo desde el Model Registry; monitoreo y logging integrados.

Vertex AI Feature Store

* Latencia: <10 ms en lecturas online.
* Elasticidad: Gestiona millones de features sincronizados entre entrenamiento y producción.
* Costo: Por almacenamiento y lecturas; menor que mantener Redis o BQ manualmente.
* Facilidad de operación: Simplifica la consistencia entre train/serve sin infraestructura externa.

1. **Serving / Integración**

Cloud Run

* Latencia: Muy baja (arranque frío <1 s); adecuada para microservicios API.
* Elasticidad: Escala automáticamente a 0 cuando no hay solicitudes (serverless).
* Costo: Pago por invocación y tiempo de CPU; muy eficiente.
* Facilidad de operación: Despliegue con un solo comando (gcloud run deploy), sin gestión de servidores ni balanceadores.

Pub/Sub (Alertas)

* Latencia: ~100 ms, útil para disparar flujos de respuesta ante fraudes.
* Elasticidad: Escala de manera transparente ante picos de alertas.
* Costo: Bajo; se paga por mensajes publicados.
* Facilidad de operación: Integración directa con Cloud Run, Dataflow y funciones serverless.

1. **Gobierno, Seguridad y Operación**

| Servicio | Elasticidad | Facilidad de Operación |
| --- | --- | --- |
| Cloud IAM | Escalable con políticas por recurso | Centralizado y granular por rol |
| VPC Service Controls | Protege perímetros de datos | Administración por consola/política |
| Audit Logs | Escalable y automático | Activado por defecto, visible en Logging |
| Data Catalog | Indexa millones de objetos | Etiquetado automático y buscador intuitivo |
| Cloud KMS | Alta | Gestión simple de claves y rotación |
| Monitoring / Logging | Colecta métricas de múltiples servicios | Dashboards y alertas integradas |
| Budgets & Alerts | Maneja múltiples proyectos | Notificaciones automáticas vía correo o Pub/Sub |

**Flujo de datos**

**1) Streaming (transacciones en tiempo real)**

1. Emisores → Pub/Sub  
   Los POS/apps publican eventos JSON con event\_id, timestamp, merchant\_id, amount, card\_hash, etc.  
   *Garantías:* al menos una vez; usar event\_id para deduplicar.
2. Pub/Sub → Dataflow (Beam)
   * Parsing/validación de esquema (data contract).
   * Enriquecimiento online (lookup BigQuery/Feature Store/Redis opcional).
   * Reglas de calidad (ver “Líneas de defensa”).
   * Watermarks y manejo de *late data* (p. ej., allowed lateness 5–10 min).
   * Rutas de salida:  
     a) Vertex AI Endpoint (scoring online, respuesta ms).  
     b) BigQuery RAW (streaming inserts) para auditoría.  
     c) Pub/Sub Alertas si score supera umbral.
3. Cloud Run (microservicio)  
   Orquesta la llamada a Vertex AI, agrega metadatos (latencia, versión de modelo), persiste resultados de scoring en BigQuery (PROD).
4. BigQuery → BI Engine → Looker(Studio)  
   Dashboards de fraude/KPIs casi en tiempo real.

**2) Batch (maestros y catálogo)**

1. Fuentes on-prem/SaaS → Cloud Storage  
   Archivos diarios (CSV/Parquet) por cliente/producto/comercio con nombres versionados (dt=YYYY-MM-DD).
2. Cloud Composer (Airflow) → Dataflow/BigQuery
   * Carga a BigQuery RAW (landing tables particionadas).
   * Transformaciones a STAGING (tipificación, *surrogate keys*, *SCD* si aplica).
   * Modelado lógico a PROD (marts de fraude y BI).
3. ML (batch)
   * Entrenamiento en Vertex AI con datasets de BigQuery.
   * Batch scoring periódico con Dataflow o BigQuery ML → resultados a PROD.

**Líneas de defensa (calidad/validación)**

Contrato de datos (nivel 0):

* Esquema obligatorio (tipos, nulos, dominios). Rechazo temprano en Dataflow (dead-letter).

Defensa 1 – Validación sintáctica:

* *Schema enforcement*: campos requeridos, tipos, longitudes.
* Deduplicación por event\_id + ventana temporal.

Defensa 2 – Validación semántica:

* Reglas de negocio: amount > 0, currency ∈ {USD, CLP,…}, merchant\_id existente, card\_status=ACTIVE.
* *Cross-checks* con maestros (join a STAGING).
* Quarantine bucket/table para registros fallidos con motivo y *replay* planificado.

Defensa 3 – Calidad continua (DQ):

* *Checks* de completitud, unicidad, consistencia, puntualidad, volumen esperado.
* Métricas DQ a Monitoring y alertas (p. ej., tasa de fallos > 1%).
* En batch, conteos de control (source vs BQ), SCD y *late arriving dimensions*.

Defensa 4 – ML/Modelo:

* Validación offline (AUC/PR, drift de features/labels).
* Canary/Rollback de modelos en Vertex AI; *feature skew detection*.
* Umbrales de score y reglas explicables (p. ej., bloquear si MCC + país + monto alto).

**Seguridad**

IAM (quién puede hacer qué)

* Principio de mínimo privilegio:
  + *Service Accounts* separadas por dominio (ingesta, ETL, ML, BI).
  + Roles granulares gestionados por IAM Conditions (p. ej., acceso solo a dataset=fraud\_raw).
* Separación de zonas:
  + RAW/STAGING/PROD con datasets distintos y políticas Row-/Column-Level Security en BigQuery para PII.
* Aislamiento:
  + VPC Service Controls para perímetro de datos (BQ, GCS, AI).
  + Workload Identity Federation (si hay identidades externas).

KMS (cómo se cifran los datos)

* CMEK habilitado en BigQuery, Pub/Sub, GCS, Vertex AI.
* Key ring por entorno (dev/qa/prod) y región; rotación automática.
* Envelope encryption para secretos; credenciales en Secret Manager (no en código/vars plano).
* Data masking y tokenización para PAN/RUT u otros identificadores.

**Operabilidad (logging/monitoring)**

Observabilidad de pipelines

* Cloud Logging:
  + Logs estructurados (con trace\_id, event\_id, model\_version).
  + *Log-based metrics* (p. ej., tasa de DLQ, excepciones por transform).
* Cloud Monitoring:
  + Dashboards: *throughput* (msg/s), lag de suscriptor, latencia E2E, errores 5xx en Cloud Run.
  + SLO/SLA: p. ej., *P95 latencia scoring* < 300 ms; *disponibilidad endpoint* ≥ 99.9%.
  + Alerting (correo/PagerDuty/Pub/Sub):
    - Lag > umbral N min.
    - Registros inválidos > X% por 5 min.
    - Aumento de costo diario > Y%.
    - *Drift* de features > Z desviaciones (Vertex AI Model Monitoring).
* Error Reporting / Trace / Profiler:
  + Centraliza *stack traces* de Cloud Run/Dataflow.
  + *Tracing* de request→scoring→persistencia (latencia por salto).