Big Data
Proyecto de curso
Segundo Cuatrimestre - 2025

Cloud Provider Analytics

(ETL + Streaming + Serving en Cassandra)

Contexto

Eres el equipo de datos de un proveedor de nube. Tu misión es **ingestar, limpiar, conformar y publicar** datos de clientes para analítica de **FinOps, Soporte y Producto**. La organización exige dos capacidades:

1. **Near real-time** para métricas operativas (uso/consumo, costos incrementales).
2. **Batch diario/mensual** para maestros (CRM) y facturación.

Los datos **aterrizan crudos** con inconsistencias, **nulos**, tipos ambiguos, **anomalías** y una **evolución de esquema** a mitad del histórico (aparecen campos nuevos como `carbon_kg` y `genai_tokens`).

Tu objetivo: construir un pipeline con **PySpark en Google Colab**, usando **Parquet** como storage intermedio **gestionado por Spark** y publicar **marts** de analítica en **AstraDB (Cassandra)** para ser consultados por herramientas de visualización (Tableau/Power BI/Superset/Grafana, etc.).

Datos provistos (Landing)

El zip incluye (carpeta `datalake/landing/`):

- `customers_orgs.csv`: clientes/tenants (industria, región, plan, NPS… con nulos y valores ruidosos).
- `users.csv`: usuarios por organización (roles, actividad, timestamps).
- `resources.csv`: recursos cloud por servicio (`compute`, `storage`, `database`, `networking`, `analytics`, `genai`).
- `support_tickets.csv`: tickets (categoría, severidad, SLA, CSAT con nulos, abiertos/cerrados).
- `marketing_touches.csv`: toques de marketing (canales, conversiones).

- `nps_surveys.csv`: NPS temporales por organización (con nulos).
- `billing_monthly.csv`: facturación 3 meses (impuestos, créditos/ajustes, monedas).
- `usage_events_stream/*.jsonl`: **feed de eventos de uso** (para **Structured Streaming**).
    - **Evolución de esquema**: a partir de ~45 días atrás aparece `schema_version=2` con `carbon_kg` y, para `genai`, `genai_tokens`.
    - Campos con nulos y valores atípicos (p. ej., `cost_usd_increment` negativo ocasional o spikes grandes).

Nota: los archivos de eventos están intencionalmente fragmentados para simular micro-lotes.

Arquitectura y Zonas del Data Lake

- **Landing (Raw Inmutable):** archivos originales. **No modificar.**
- **Bronze (Raw Estándar):** mismo grano que la fuente pero con **tipificación explícita**, **dedupe por `event_id`**, y **marcas de ingesta** (`ingest_ts`, `source_file`).
- **Silver (Conformado):** datos **limpios y conformance** (campos normalizados, joins con dimensiones, tratamiento de nulos y outliers, control de versiones de esquema, **SCD si aplica**).
- **Gold (Marts de negocio):** vistas/tables orientadas a analítica (**FinOps**, **Soporte**, **Producto/Usage**), listas para servir en Cassandra (AstraDB).

Formato intermedio: Parquet particionado (p. ej. `date=` y/o `service=`).
Control de calidad: reglas de validación (propias o con Expectations) para bloquear/aislar datos malos.

Patrón arquitectónico (elige y justifica)

- **Lambda:**
    - **Batch**: maestros (orgs, users, resources), facturación y encuestas (NPS).

    - **Streaming**: `usage_events_stream` (ventanas, watermarks, upserts idempotentes).

- **Kappa:**

    - Trata **todo** como stream, alimentando las capas Silver/Gold con vistas derivadas (re-stream/backfill desde los archivos).

Requisito: Implementar **al menos** streaming de eventos y batch de maestros/facturación. Explicar por qué tu elección es adecuada.

Requerimientos técnicos (obligatorios)

1. **Ingesta**

    - **Batch**: leer CSV/JSON desde `landing/` a **Bronze** Parquet, **particionado**.

    - **Streaming**: Structured Streaming (fuente `json` por directorio), **esquema explícito**, `withWatermark`, dedup por `event_id`, manejo de **late data**.

2. **Calidad de datos**

    - Tipos consistentes (p. ej., `value` puede venir string → castea con fallback).

    - Reglas mínimas (ejemplos):

        - `event_id` **no nulo** y **único** por ventana.

        - `cost_usd_increment` ∈ [−0.01, +∞) y flag de anomalía si > p99*X.

        - `unit` no nulo cuando `value` no nulo (o imputación).

        - `schema_version` manejada y **compatibilizada** (v1/v2).

    - **Registro de errores** (quarantine) en Parquet aparte.

3. **Transformaciones (Silver)**

    - Normalización de números, fechas, regiones/servicios.

    - **Enriquecimiento**: join a orgs/users/resources.

    - **Features** calculadas:

- `daily_cost_usd` por org/servicio, `requests`, `cpu_hours`, `storage_gb_hours`, `genai_tokens`, `carbon_kg`.
- Flags de **anomalía** (3 métodos a elección: z-score, MAD, p-tiles).

4. **Gold (marts) — granos y esquemas** sugeridos
   - **FinOps:**
     - `org_daily_usage_by_service` (org_id, usage_date, service → métricas y costos).
     - `revenue_by_org_month` (org_id, month → revenue USD, créditos, impuestos, FX aplicado).
     - `cost_anomaly_mart` (org_id, date, service → score/flag).
   - **Soporte:**
     - `tickets_by_org_date` (org_id, date, severidad, counts, SLA breach rate, CSAT prom.).
   - **Producto/Usage (GenAI):**
     - `genai_tokens_by_org_date` (org_id, date → total_tokens/costo_estimado).
5. **Serving en AstraDB (Cassandra)**
   - Crear **keyspace** (p. ej., `cloud_analytics`).
   - Diseñar **tablas por consulta** (modelado por **query-first**).
   - Cargar **Gold** a Cassandra con **conector Spark** o `foreachBatch` + driver Python.
6. **Idempotencia**
   - Reprocesar sin duplicar (checkpointing, keys naturales, upserts).
7. **Performance**
   - Particionado sensato; **reparquet** si hace falta; **coalesce/repartition**.

8. **Documentación**
    - Diagrama simple de la arquitectura, diccionario de datos clave, **decisiones y trade-offs** (p. ej., particionamiento, claves Cassandra, umbrales de anomalías).
9. **Demo**
    - Ejecutar **5 consultas** típicas sobre AstraDB que respalden dashboards (ver más abajo).

Consultas mínimas que deben responderse (desde AstraDB)

1. Costos y requests diarios por **org** y **servicio** en un rango de fechas.
2. **Top-N servicios** por costo acumulado en los últimos 14 días para una organización.
3. Evolución de **tickets críticos** y tasa de **SLA breach** por día (últimos 30 días).
4. **Revenue mensual** con créditos/impuestos aplicados (normalizado a USD).
5. **Tokens GenAI** y costo estimado por día (si existen).

(Incluye los CQL y captura de resultados en el notebook.)

Entregables

1. Una presentación donde se detalle la arquitectura, las tecnologías y todo el proceso de desarrollo: los recursos utilizados, propuesta de ingeniería para el contexto de negocio, las acciones realizadas en cada una de las etapas: Arquitectura y Zonas del Data Lake, Patrón Arquitectónico, Requerimientos Técnicos y las consultas mínimas de demostración.
2. Un video explicativo basado en la presentación y en la solución realizada Spark y Cassandra. Debe mostrar el funcionamiento del pipeline y los resultados del ETL.

Evaluación

1. Diseño y desarrollo: Se evaluará el ingenio de la solución a partir del contexto de negocio, la arquitectura de la solución y la implementación en Spark y Cassandra (todos los datos deben estar en un Keyspace de Cassandra utilizando colecciones)..
2. Calidad: Correcto análisis de los datasets y ajuste de los datos de acuerdo a las reglas de negocio.

3. Profundidad: Decisiones tomadas y estrategias utilizadas, tanto de diseño como técnicas.
4. Storytelling: Se valorará el storytelling para presentarlo en el video.