Event-Driven Data Lineage for Banking Operations

1) Problem Statement

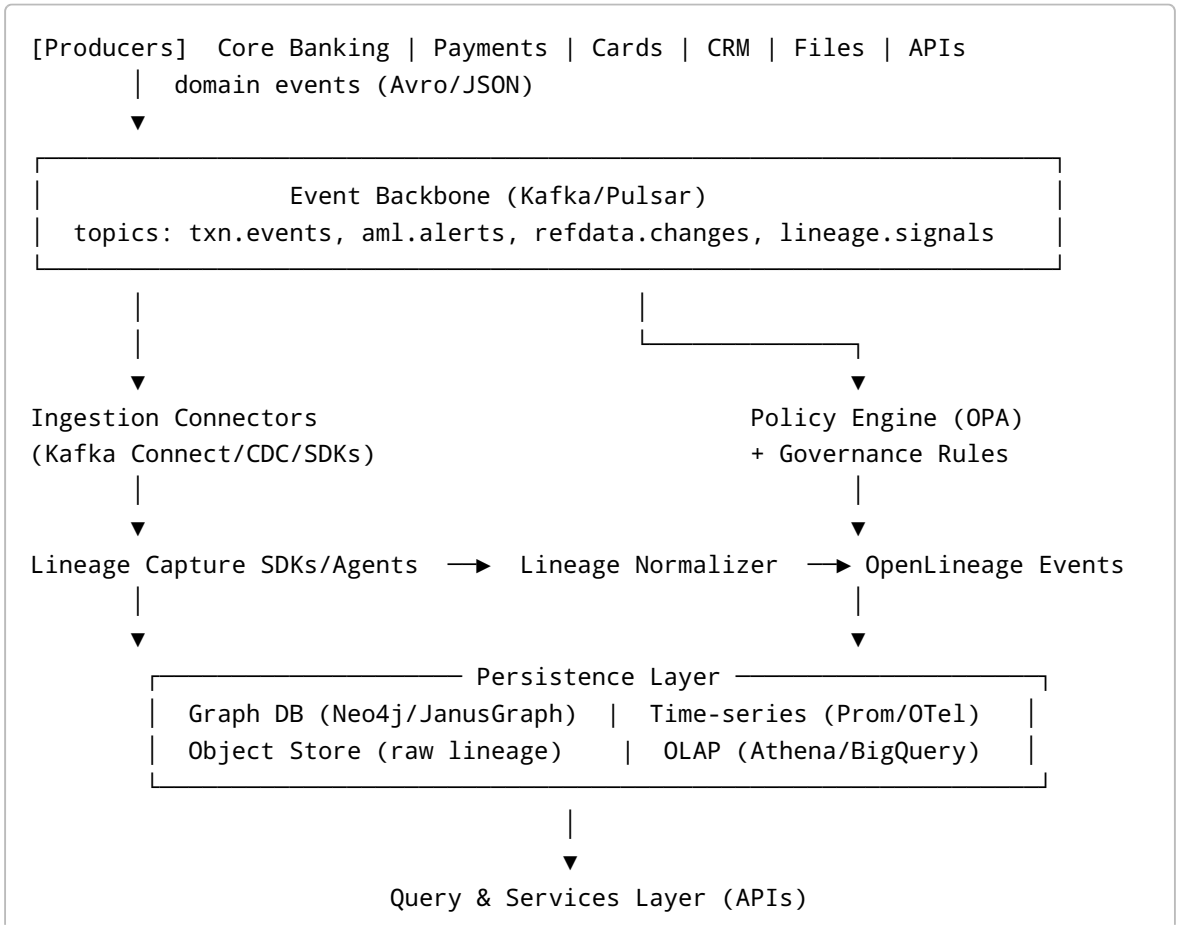
Large banks process millions of events daily across core banking, payments, AML, risk, and reporting systems. Data moves through batch and streaming pipelines, making it hard to **prove lineage, ensure governance**, and **pass audits**. Gaps cause regulatory risk (GDPR, SOX, BCBS 239), delayed investigations, and costly compliance.

2) Solution Overview

Build an **event-driven lineage and governance platform** that captures lineage as **events at the point of change** (publish/consume/transform/load), persists it in a **graph store**, enforces **policy-as-code** for governance, and generates **immutable audit trails** for regulators. The platform integrates with Kafka (or Pulsar), ETL/ELT tools, databases, and SaaS apps.

Outcomes: real-time lineage graph, searchable provenance, automated controls (masking/retention/access), auditor-facing reports, and anomaly detection on lineage & policy breaches.

3) Reference Architecture



```
- Lineage Query API  - Provenance Search API
- Policy Evaluation  - Audit/Report Service
                    |
                    ▼
            UX: Ops, Risk, Audit Portals
- Lineage Graph UI  - Data Catalog views
- Evidence export (PDF/CSV) - Investigator workbench
```

4) Core Components

4.1 Lineage Capture SDKs & Agents - Lightweight interceptors for producers/consumers, ETL/ELT, Spark/Flink jobs, DB ingestion/egress. - Emit **OpenLineage-compatible** events: `Job`, `Dataset`, `Run`, `Facet` (schema, column-level, data quality checks, PII tags). - Connectors: Kafka Connect SMT, Debezium CDC, Spark Listener, Airflow/Flyte/Argo plugins.

4.2 Lineage Normalizer - Validates, de-duplicates, enriches with **business metadata** (domain, product, criticality), and maps to **graph schema** (node/edge types: `System`, `Dataset`, `Process`, `Run`, `Column`). - Performs **column-level propagation** (e.g., PII flag flows from `customer.ssn` → derived tables).

4.3 Governance & Policy Engine - **Policy-as-code (OPA/Rego)** for access, retention, masking, residency, SoD. - **Inline enforcement** (in pipelines) and **out-of-band checks** (pre-deploy approvals, nightly scans). - Integration with IAM (RBAC/ABAC), KMS/HSM for encryption.

4.4 Audit & Evidence Service - Writes an **append-only ledger** (e.g., Kafka + Object Store + hash chain) of policy evaluations & lineage snapshots. - Generates **auditor-ready evidence**: who accessed what, when; lineage at a timestamp; control outcomes.

4.5 Lineage Graph Store & APIs - Graph DB storing `(:Dataset)-[:DERIVED_FROM]->(:Dataset)` and `(:Run)-[:USED/:-WROTE]->(:Dataset)` edges. - **Time-travel lineage** via valid-from/to intervals. - **APIs**: `/lineage/upstream`, `/lineage/downstream`, `/lineage/impact`, `/policy/evaluate`, `/audit/evidence`.

4.6 Portals (UI) - **Lineage Explorer** (graph & column views), **Impact Analyzer** (change/incident blast radius), **Audit Console**, **Data Catalog plugin** (show lineage on dataset pages).

5) Data & Event Models

5.1 Event Contracts (Avro/JSON) - `BankTxnEvent`: `txnId, accountId, amount, currency, channel, ts, piiTags[]`. - `LineageEvent (OpenLineage)`: `run, job, inputs[], outputs[], facets{schema, columnLineage, dataQuality, pii}`. - `PolicyDecision`: `subject, resource(dataset/column), action, decision, obligations(masking), ts, evidencePtr`.

5.2 Graph Schema (simplified) - Nodes: `System`, `Dataset(schema, columns[], piiTags[], owner)`, `Process(jobType, codeRef)`, `Run(runId, ts, status)`. - Edges: `WROTE`, `USED`, `DERIVED_FROM`, `GOVERNED_BY`, `OWNS`.

6) Governance Controls

- **Access control**: ABAC on dataset & column; dynamic masking (tokenization, format-preserving encryption).
- **Retention**: policy per domain; automated **delete/retain** workflows with proof-of-deletion.
- **Data residency**: region-aware routing; prevent cross-border replication without approval.
- **Quality**: Great Expectations/Deequ checks; block writes on severity ≥ `HIGH`.

7) Security & Compliance

- End-to-end **TLS/mTLS**, **encryption-at-rest**.
- Secrets in vault; signed lineage events; ledger with hash chain for tamper evidence.
- Compliance mappings: **GDPR (Art. 5/30/32)**, **SOX**, **BCBS 239**, **PCI DSS** for card data.

8) User Flows

8.1 Incident Traceback (failed settlement) 1. Ops selects `txnId` → system fetches upstream lineage for `settlement_ledger` row. 2. Graph shows source topic → enrichment job → risk scoring → ledger write. 3. UI highlights where **data quality check failed** and **policy override** occurred. 4. Ops exports an **evidence bundle** (lineage snapshot + logs + policy decisions).

8.2 Impact Analysis (schema change) 1. Developer proposes column rename `customer_email → email`. 2. Pre-deploy hook queries **downstream dependencies**; shows impacted jobs, reports, ML features. 3. Change is blocked until owners **ack** or policy exemption granted.

8.3 Auditor Request - Auditor selects dataset/date → portal generates **point-in-time lineage** + access logs + control attestations.

9) APIs (Sketch)

```
GET  /lineage/upstream?dataset=payments.transactions&depth=3
GET  /lineage/downstream?dataset=core.ledger&depth=2
POST /policy/evaluate { subject, action, resource }
GET  /audit/evidence?dataset=core.ledger&at=2025-09-01T00:00:00Z
```

10) Tech Stack (reference options)

- **Streaming**: Kafka (MSK/Confluent) or Pulsar; Schema Registry.
- **Orchestration**: Airflow/Flyte/Argo; Spark/Flink for transforms.
- **Lineage**: OpenLineage emitters; Apache Atlas or Marquez for catalog.
- **Graph**: Neo4j/JanusGraph (+ Elasticsearch for text search).
- **Policy**: OPA/Rego; IAM (AAD/Okta); KMS/HSM.
- **Storage**: S3/GCS/Azure Blob (raw), Parquet/ORC; OLAP (Athena/BigQuery/Snowflake).
- **Observability**: OpenTelemetry + Prometheus; Grafana; lineage metrics (lag, completeness).
- **UI**: React + Graph visualization (Cytoscape/Dagre), RBAC.

11) Deployment & Ops

- **Zero-downtime** deploys; blue/green for policy engine.
- Multi-region DR (RPO ≤ 15 min, RTO ≤ 60 min); topic replication with filters honoring residency.
- Backfill strategy: replay topics into lineage normalizer; batch import from existing catalogs.

12) KPIs & ROI

- **Audit prep time** ↓ 70% (point-in-time lineage & evidence export).
- **Incident MTTR** ↓ 40% (blast radius & root-cause pathfinding).
- **Policy violations caught pre-prod** ↑ 60% (gates in CI/CD).
- **Data access review time** ↓ 50% (ABAC + self-service attestations).

13) Risks & Mitigations

- **Event volume explosion** → sampling + aggregation; partitioning by domain; async compaction.
- **Connector gaps** → SDKs + agent framework; prioritize high-risk systems first.
- **False positives in policy** → shadow mode, staged rollout, exception workflow.

14) Phased Roadmap (90–180 days)

- **Phase 1 (0–6 wks)**: Kafka lineage emitters, normalizer, graph store, basic UI.
- **Phase 2 (6–12 wks)**: Policy engine inline checks, auditor evidence service, catalog plugin.
- **Phase 3 (12–24 wks)**: Column-level lineage propagation, quality gates, impact analysis & CI hooks.
- **Phase 4 (24+ wks)**: Cross-region residency controls, ML-driven anomaly detection on lineage graphs.

15) Demo Script (15 minutes)

1. Ingest sample payment events → live lineage graph updates.
2. Trigger a schema change → see impacted jobs & blocked deploy.
3. Run an auditor query at a past timestamp → export PDF/CSV bundle.
4. Show masking policy in action for an analyst vs auditor persona.

Notes for Implementation - Favor **OpenLineage** to avoid vendor lock-in; provide adapters for Atlas/ Marquez. - Keep lineage events and policy decisions **idempotent** and **verifiable** (signing + hash chain). - Make everything **API-first** so Ops tools and catalogs can integrate easily.