Retail Analytics & Recommendation Service

Objective:

Build a cloud-native analytics pipeline for retail that ingests sales data, stores raw data in Cloud Storage, loads curated data into Cloud SQL for OLTP, and serves a recommendation microservice running on GKE with a Cloud Run dashboard. Focus: containers, GKE, Cloud Storage, Cloud SQL, Terraform, Cloud Run.

Learning goals

- Design a data-aware retail microservice + dashboard architecture on GCP.
- Use Terraform to provision Data + Compute infra reliably.
- Deploy analytics microservices on GKE and serverless dashboards on Cloud Run.
- Use Cloud Storage as data lake; implement lifecycle policies.
- Secure Cloud SQL connectivity and optimize DB schema for fast reads.
- Automate infra and app delivery with Cloud Build.
- Add monitoring, tracing, and cost controls.

High-level architecture

- **Data ingestion:** Batch CSV/JSON uploads to Cloud Storage (or pushed by an ingestion service).
- **Processing service (optional):** A container on GKE that processes files from Storage, enriches them, and writes aggregates to Cloud SQL. (Could also use Cloud Run job).
- **Recommendation API:** Microservice on GKE serving personalized product recommendations (calls Cloud SQL and a simple in-memory model).
- **Dashboard:** Cloud Run-hosted dashboard that queries Recommendation API and product metadata.
- **Storage:** Cloud Storage buckets for raw and processed data, with lifecycle rules.

- **Infra:** Terraform modules for all resources.
- **CI/CD:** Cloud Build pipelines to build images, run tests, deploy.

Lab Steps

Step 1 — Terraform provisioning

- Modules for VPC, GKE, Cloud Storage (raw/processed), Cloud SQL, Artifact Registry, IAM.
- Create Cloud Storage buckets with lifecycle rules and uniform bucket-level access.
- Apply infra.

Deliverable: Provisioned GCP infra.

Step 2 — Data & containers

- Provide sample dataset (sales.csv) to upload to gs://retail-raw/.
- Create a processing container (Python) that reads new objects, normalizes, writes to processed bucket and loads product/aggregate rows to Cloud SQL.
- Create Recommendation API (e.g., Flask) container that offers /recommendations?user_id=....
- Create Dashboard frontend (React/Flask) for product lists and saved recommendations.

Deliverable: Containers and sample datasets.

Step 3 — Deploy processing + recommendation service

- Deploy the processing service as a CronJob on GKE or Cloud Run Job to pick up new files periodically.
- Deploy Recommendation API to GKE with HPA and readiness/liveness probes.
- Ensure DB connectivity via Cloud SQL Proxy (sidecar or connector).

Deliverable: Processing working + Recommendation API live.

Step 4 — Cloud Run dashboard

- Deploy UI to Cloud Run.

- Configure CORS and authentication if needed.
- Integrate with backend recommendation API.

Deliverable: Live dashboard and API.

Step 5 — Observability & logging

- Dashboards: GKE (processing + API), Cloud Run (dashboard metrics), Cloud Storage (ingestion metrics), Cloud SQL.
- Export processed logs to Cloud Storage and BigQuery (optional) for analytics.

Deliverable: Monitoring and logs in place.

Step 6 — CI/CD

- Two pipelines:
    - cloudbuild-data.yaml: build processing image, push to Artifact Registry, run integration tests.
    - cloudbuild-app.yaml: build recommendation API and dashboard, deploy to GKE and Cloud Run.
- GitHub triggers to run pipeline on push to main.

Deliverable: End-to-end CI/CD.

Step 7 — Cost & optimization

- Implement lifecycle rules for raw-to-cold storage, smaller node pools for GKE with preemptible nodes for batch processing.
- Create budgets and alerts.

Deliverable: Cost governance artifacts.

Expected deliverables

1. Working analytics + recommendation system (dashboard + API + processing pipeline).
2. Terraform repo and applied infra.
3. CI/CD pipelines.

4. Dashboards and logging sinks.
5. Documentation explaining data flow and scaling decisions.