Smart Inventory & Promotions Platform (Retail)

Objective:

Design, deploy, and automate a cloud-native retail platform that tracks inventory in near real-time, serves product pages and promotions, and supports safe, private database connectivity. Emphasis: Containers, GKE, Cloud Run, Cloud Storage, Cloud SQL, Terraform, CI/CD, observability, and cost governance.

Learning goals

- Architect a cloud-native, 3-tier retail app on GCP.
- Implement Infrastructure-as-Code using Terraform (modules + state).
- Deploy containerized services to GKE and Cloud Run.
- Connect Cloud Run frontend to a private Cloud SQL backend.
- Use Cloud Storage for product images and log archival.
- Implement CI/CD (Cloud Build) for infra + app pipelines.
- Configure monitoring, logging, autoscaling and basic security controls.
- Apply cost controls, budgets, and lifecycle policies.

High-level architecture

- **Frontend (public):** Cloud Run — product catalog + promotion pages (containerized web app).
- **Backend API:** GKE (internal) — inventory, promotions, order intake microservices.
- **Database:** Cloud SQL (Postgres or MySQL) with private IP and Cloud SQL Proxy (or Serverless VPC Access).
- **Storage:** Cloud Storage bucket for product images, static assets, backups, and long-term logs.

- **Artifacts:** Artifact Registry for Docker images.
- **Automation:** Terraform for all infra provisioning.
- **CI/CD:** Cloud Build triggers: (1) Terraform plan/apply for infra branches; (2) Build/push images + apply K8s manifests + deploy Cloud Run.
- **Monitoring/Security:** Cloud Monitoring dashboards, Logging sinks to Cloud Storage, IAM roles, VPC + firewall rules, Security Command Center review.

Lab Steps

Step 1 — Terraform infra (IaC)

- Create reusable Terraform modules for: network (VPC, subnets), GKE cluster, Cloud SQL instance, Cloud Storage bucket, Artifact Registry, IAM roles, Cloud Run service account, and optional Cloud NAT.
- Configure remote state backend (e.g., GCS backend) and workspaces for dev/prod.
- terraform init → terraform plan → terraform apply.
- Verify resource creation in Cloud Console.

Deliverable: Terraform repo + applied infra.

Step 2 — App code & containerization

- Provide sample repo with two apps:
    - Frontend (React/Node.js or Flask + Jinja) that lists products and promotions; reads images from Cloud Storage; calls /api.
    - Backend (Python Flask / Node Express) offering /api/products, /api/inventory, /api/promo.
- Create Dockerfiles for frontend and backend; build locally and test.
- Push images to Artifact Registry via gcloud builds submit or Cloud Build.

Deliverable: Images in Artifact Registry.

Step 3 — Deploy backend to GKE

- Create K8s manifests: Deployments, Services (ClusterIP), ConfigMaps, Secrets (use Secret Manager or K8s secrets), HPA (CPU-based), and NetworkPolicies.
- Use Cloud SQL Auth Proxy sidecar or private IP connector for DB access.
- Apply manifests; test internal API endpoints from within cluster.

Deliverable: GKE-hosted backend with HPA.

Step 4 — Deploy frontend to Cloud Run

- Deploy frontend container to Cloud Run (fully managed).
- Configure ingress and allow unauthenticated if needed.
- Secure the Cloud Run → GKE communication: use Serverless VPC Access + internal HTTP(S) load balancer or an internal domain for GKE service using Ingress. Optionally implement authentication (IAP).
- Implement revision traffic splitting to simulate blue/green deploys.

Deliverable: Public frontend connected to GKE backend.

Step 5 — Storage & media

- Upload product images to Cloud Storage; set lifecycle rules (e.g., move old images to Nearline/Coldline after 90 days).
- Configure signed URLs for controlled image access (or public bucket if acceptable).

Deliverable: Cloud Storage with lifecycle and sample product images.

Step 6 — Observability & security

- Create Cloud Monitoring dashboards for GKE (pod/cluster CPU, memory), Cloud Run metrics (latency, requests), and Cloud SQL (connections, CPU).
- Set up Log Router to export logs to Cloud Storage for long-term retention and BigQuery for analytics (optional).
- Run Security Command Center checks and fix high-priority findings.
- Define IAM principle of least privilege for service accounts.

Deliverable: Dashboards, logging sinks, SCC report.

Step 7 — CI/CD with Cloud Build

- Create cloudbuild-terraform.yaml to run terraform fmt/validate/plan then apply (protected by approvals for prod).
- Create cloudbuild-app.yaml to build images, push to Artifact Registry, and deploy (kubectl apply for GKE and gcloud run deploy for Cloud Run).
- Wire automatic triggers on GitHub branches/tags.

Deliverable: Automated CI/CD pipelines.

Step 8 — Cost control & governance

- Create Billing budgets and alerts (e.g., 50%/75%/90% thresholds).
- Demonstrate rightsizing suggestions (reduce node pool sizes, use preemptible nodes for non-critical workloads).
- Implement resource cleanup scripts and bucket lifecycle rules.

Deliverable: Budget alerts + cost-optimization report.

Expected deliverables (final)

1. Running retail app (public frontend + backend + DB).
2. Terraform code (modular) + README for environment setup.
3. Kubernetes manifests & Cloud Run deployment config.
4. Cloud Build configs and GitHub triggers.
5. Observability dashboards and log sinks.
6. Billing budget & cost optimization notes.
7. Short demo video or README with screenshots.

Sample repo structure

```
smart-inventory/
├─ terraform/
│  ├─ modules/
│  │  ├─ network/
```

```
│   │   ├── gke/
│   │   ├── cloudsql/
│   │   ├── storage/
│   │   └── iam/
│   ├── envs/
│   │   ├── dev/
│   │   └── prod/
│   └── README.md
├── infra-pipeline/
│   ├── cloudbuild-terraform.yaml
├── apps/
│   ├── frontend/
│   │   ├── Dockerfile
│   │   └── src/
│   └── backend/
│       ├── Dockerfile
│       └── src/
├── k8s/
│   ├── backend-deployment.yaml
│   └── backend-hpa.yaml
├── cloudrun/
│   └── cloudrun-deploy.sh
└── docs/
    └── demo.md
```
