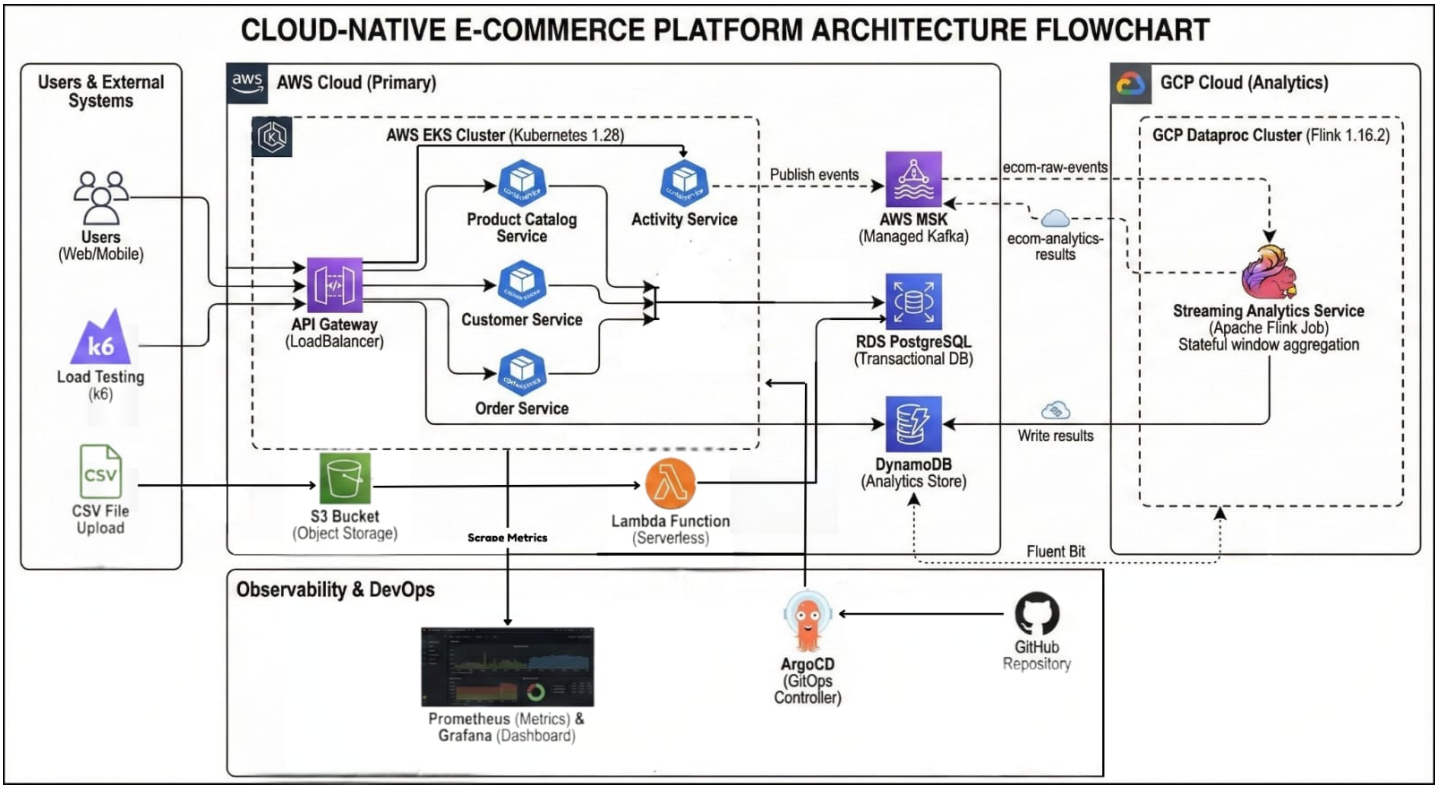
Cloud-Native E-Commerce Platform – Design Document (MVP)



1. System Overview

Domain: E-Commerce

Description: Multi-cloud, containerized e-commerce event-driven application with analytics.

Characteristics:

- Core transactional **microservices** (CRUD + order flows) run on **AWS EKS**.
- **Real-time streaming** analytics runs on **GCP Dataproc (Flink)**.
- Cross-cloud event flow via managed **Kafka (AWS MSK)**.
- **Serverless** ingestion via **AWS Lambda** triggered by **S3 file uploads** (batch product import).
- **Observability** via **Prometheus + Grafana + Fluent Bit + GCP Logging**.
- **GitOps** deployment via **ArgoCD** (declarative GitOps-managed application workloads).
- Infrastructure entirely **provisioned with Terraform** (AWS + GCP resources).

Group Details (G13):

Name	BITS ID
Arnav Bharti	2022B1A71585P
Praneel Maddula	2022A7PS0140P
Himanshu Kumar	2022A8PS0557P
Sahitya Singh	2022B4A70920P

2. Microservices Architecture

Six microservices are deployed. Five of them are on **AWS on an EKS Cluster** for the core ecommerce platform and the **Flink service** is running on **GCP on Dataproc**. There is also a **Lambda function** for data ingestion.

#	Microservice / Component	Provider	Purpose	Interfaces / Protocols
1	API Gateway	AWS (EKS)	Single public entry point (REST); routes requests to internal services; provides aggregated product/ customer/order operations; exposes analytics query (DynamoDB)	REST (HTTP)
2	Product Catalog	AWS (EKS)	Manage products (CRUD)	REST (HTTP), PostgreSQL
3	Customer Service	AWS (EKS)	Manage customer records (CRUD)	REST (HTTP), PostgreSQL
4	Order Service	AWS (EKS)	Manage orders and basic status updates	REST (HTTP), PostgreSQL
5	Activity Service	AWS (EKS)	Capture user activity events and publish them to Kafka raw topic	REST (HTTP), Kafka Producer
6	Streaming Analytics (Flink	GCP (Dataproc)	Consume raw Kafka events, perform 1-minute tumbling window unique-	Kafka Consumer/

#	Microservice / Component	Provider	Purpose	Interfaces / Protocols
	Job)		user aggregation per product, publish results topic, write aggregates to DynamoDB	Producer, DynamoDB SDK
*	AWS Lambda (Serverless Function)	AWS	Asynchronous product batch ingestion from S3 CSV -> API Gateway POST /products	S3 Event, REST (HTTP)

4. Topics & Storage

- **Kafka Topics (AWS MSK):**
  - `ecom-raw-events` – raw user activity events.
  - `ecom-analytics-results` – aggregated analytics results (still published for assignment compliance even though DynamoDB is primary query store).
- **Storage:**
  - **RDS PostgreSQL:** Product, Customer, Order tables.
  - **S3 Bucket:** CSV file uploads triggering Lambda for product bulk import.
  - **DynamoDB Table:** Partition key `product_id`, sort key `window_start`; items contain `unique_user_count`, `window_end`.
- **NoSQL because:** Low-latency keyed access to recent aggregated windows; avoids complex joins.

5. Infrastructure as Code (Terraform)

All AWS & GCP components are provisioned via **Terraform**:

- **AWS**: VPC, Subnets, EKS Cluster/Node Group, RDS Postgres, DynamoDB, MSK, S3 bucket, IAM roles & policies, Lambda function and trigger, ArgoCD (Helm release), ECR repositories.
- **GCP**: Dataproc cluster (with Flink component), Service Account & IAM bindings, Networks/Subnets, GCS buckets for staging & Flink job JAR.

6. GitOps Deployment Model

- ArgoCD installed via **Terraform Helm** release.
- ArgoCD Applications **reference Git repository** paths (e.g., `k8s/` for microservice manifests, `observability/` for monitoring).
- Microservice manifests (Deployments, Services, HPAs) reside under version control.
- Imperative manual application of manifests is eliminated (previous helper script section treated as legacy; final compliance uses ArgoCD sync).

7. Horizontal Pod Autoscalers (HPAs)

Two critical services auto-scale based on CPU & memory utilization:

- `api-gateway-hpa`
- `activity-service-hpa`

Scaling Policy:

- Min replicas: 2
- Max replicas: 10
- CPU target: ~60%
- Memory target: ~70%

8. Real-Time Analytics (Flink on GCP)

- GCP Dataproc.
- Consumes Kafka raw topic (`ecom-raw-events`).
- Event-time 1-minute tumbling windows with bounded out-of-order (10s).
- Aggregates unique user counts per product.
- Publishes JSON results to `ecom-analytics-results` .
- Writes each aggregated window directly into DynamoDB (AWS) using AWS SDK credentials exposed to Dataproc job.

9. Serverless Function

AWS Lambda:

- Trigger: S3 `ObjectCreated:*` events for CSV files (suffix `.csv`).
- Parses rows -> POST /products via API Gateway.
- Provides asynchronous ingestion (decoupled from user traffic).
- Infrastructure & trigger binding declared in Terraform.

10. Observability Stack

Components:

- **Prometheus** (kube-prometheus-stack).
- **Grafana** (enabled in the stack).
- **Fluent Bit** (forwarding logs to GCP Logging; centralizing microservice + Flink logs).
- **Metrics instrumentation**: `prometheus_flask_exporter` integrated in each Flask-based microservice (exposes `/metrics` automatically).
- **Dashboard** (to be imported into Grafana):
    - **Panels**: API Gateway CPU & Memory, Request Rate (derived from Flask metrics), Error Rate (non-2xx), p95 latency (histogram), Kafka consumer lag (optionally via JMX exporter later), EKS cluster node utilization.

Dashboard JSON stored in repository (e.g., `observability/grafana-dashboard-rps.json`) and applied via ConfigMap -> Grafana sidecar import.

Logging Path:

- **AWS EKS Pods** stdout/stderr -> Fluent Bit -> GCP Logging backend (single pane of glass).
- **Dataproc job** logs -> GCP Logging native integration.
- **Unified retrieval via GCP Logs Explorer** (filter by labels / resource).

11. Load Testing & Scaling Validation

Tool: **k6**
Script: `load-testing/load-test.js`
Stages: Ramped concurrency (50 -> 100 -> 200 VUs) sustaining throughput to force CPU increase in `api-gateway` & `activity-service`.
Metrics Observed:

- HPA describes scaling decisions.

- Prometheus graphs show rising CPU -> new pod replicas.
- DynamoDB writes appear (confirmation via counts in table or API Gateway analytics endpoint queries).

12. Configuration & Environment Variables

Variable	Purpose	Set In
DB_HOST / DB_PORT / DB_NAME / DB_USER / DB_PASSWORD	Postgres connectivity	K8s ConfigMap/ Secret
KAFKA_BOOTSTRAP_SERVERS	Kafka brokers string	ConfigMap
KAFKA_TOPIC / KAFKA_INPUT_TOPIC	Raw events topic	Default / env
KAFKA_OUTPUT_TOPIC	Aggregated results topic	Default / env
AWS_REGION	Region for DynamoDB/ other AWS services	ConfigMap
DYNAMODB_TABLE	Analytics table name	ConfigMap
PRODUCT_API_URL	Lambda calls API Gateway for product import	Lambda env
AWS_ACCESS_KEY_ID / AWS_SECRET_ACCESS_KEY	Provided to Dataproc job for DynamoDB writes	Job submission env

13. Data Models (Simplified)

Postgres Tables (Created by service init for MVP):

- `products(id, name, description, price, category, stock_quantity, created_at)`
- `customers(id, name, email, phone, address, created_at)`
- `orders(id, customer_id, product_id, quantity, total_price, status, created_at)`

Kafka Raw Event JSON:

```
{
```

```
  "user_id": "u123",
  "product_id": "42",
  "event_type": "VIEW_PRODUCT",
  "timestamp": "2024-01-01T12:00:00Z",
  "metadata": { "session_id": "abc" }
}
```

DynamoDB Aggregation Item:

```
{
  "product_id": "42",
  "window_start": "2024-01-01T12:00:00Z",
  "window_end": "2024-01-01T12:01:00Z",
  "unique_user_count": 17
}
```

14. Assignment Requirements

Requirement	Status	Notes
a. Terraform-only infra	Met	Terraform covers AWS & GCP; ArgoCD manages application deployments declaratively.
b. ≥6 microservices + analytics on Provider B + serverless	Met	Flink counts as analytic microservice on GCP; Lambda included separately.
c. Managed K8s + HPAs (≥2 services)	Met	EKS with HPAs for api-gateway & activity-service.
d. GitOps controller manages deployments	Met	ArgoCD Helm release + Applications declaratively manage microservices.
e. Real-time Flink job with windowed aggregation & Kafka topics	Met	Flink job on Dataproc performs 1-min unique user window, publishes results, writes to DynamoDB.
f. Distinct storage: Object + SQL + NoSQL	Met	S3 + RDS + DynamoDB.
g. Observability: Prometheus, Grafana dashboard, centralized logs	Met	Metrics exporter integrated; dashboard JSON included; logs via Fluent Bit + GCP Logging; improvement possible with more panels.

Requirement	Status	Notes
h. Load testing to demonstrate scaling	Met	k6 script exercises endpoints to trigger HPA scaling.

16. Conclusion

The current architecture satisfies all assignment specifications with minimal, functioning cloud-native patterns.