**📌 Real-Time Scenario: Event-Driven E-commerce Application on GCP**

**📝 Problem Statement**

A growing **e-commerce startup** wants to deploy a **highly scalable and event-driven application** on **Google Cloud**. The application consists of:  
1️ **Frontend Web App** (React, hosted on App Engine)  
2️. **Backend API** (Python FastAPI, running on Compute Engine)  
3️.**Database** (Cloud SQL - PostgreSQL)  
4️.**Event Processing System** (Google Pub/Sub)

The **security, scalability, and efficiency** requirements are:  
✅ Users should access the application via a **domain name (DNS)**.  
✅ The frontend should be **highly scalable** and handle traffic spikes.  
✅ The backend should interact with a **database** and process **asynchronous tasks**.  
✅ Orders should be processed in an **event-driven** manner.  
✅ The architecture should be **cost-efficient** and **secure**.

**🌐 Solution: Implementing Cloud Services for a Secure and Scalable E-commerce System**

We'll use **Compute Engine, App Engine, Cloud SQL, Pub/Sub, VPC, Load Balancers, Cloud NAT, DNS, and Firewall Rules** to design a **secure, scalable, and efficient cloud architecture**.

**Step 1️: Setting Up the VPC (Virtual Private Cloud)**

🔹 Create a **VPC (ecommerce-vpc)** to isolate our infrastructure.  
🔹 Inside this VPC, create **three subnets**:

A screenshot of a computer

AI-generated content may be incorrect.

✅ **Why?** This ensures logical separation of services for better security and network management.

**Step 2️⃣: Enabling Secure and Scalable Frontend Access**

🔹 The **React frontend** is deployed on **App Engine (Standard Environment)**, which **auto-scales** based on demand.  
🔹 Users should access the app via www.myecommerce.com.  
🔹 We register www.myecommerce.com in **Cloud DNS** and point it to an **HTTPS Load Balancer**.  
🔹 The Load Balancer directs traffic to **App Engine**, which dynamically scales instances.

✅ **Why?**  
✔️ Users access the app via a **domain name** instead of an IP.  
✔️ **Auto-scaling** ensures smooth handling of high traffic.  
✔️ **HTTPS Load Balancer** provides better performance and security.

**Step 3️: Backend API Hosted on Compute Engine**

🔹 The **FastAPI backend service** runs on **Compute Engine** in the backend-subnet.  
🔹 An **Internal Load Balancer (ILB)** distributes traffic across multiple backend instances.  
🔹 Only **App Engine** can access the backend via **ILB private IP**.

✅ **Why?**  
✔️ Backend remains **private** (no public access).  
✔️ **Load Balancer distributes traffic** to multiple instances for high availability.

**Step 4️⃣: Storing and Managing Data with Cloud SQL**

🔹 The application stores user data, products, and orders in **Cloud SQL (PostgreSQL)**.  
🔹 Cloud SQL resides in db-subnet and is **only accessible from the backend**.  
🔹 Firewall rules restrict access to **only backend Compute Engine instances**.

✅ **Why?**  
✔️ Database is **fully managed** (automatic backups, failover, and maintenance).  
✔️ Only backend API can **query the database**, ensuring security.

**Step 6️⃣: Allowing Backend API to Call External APIs Securely**

🔹 The backend API needs to call an **external payment gateway** (e.g., Stripe) for payment processing.  
🔹 Since the backend is in a **private subnet**, it needs **Cloud NAT** to access external APIs.

✅ **Why?**  
✔️ Backend can **make external API calls** while remaining private.  
✔️ No public IPs required, ensuring **better security**.

**Step 7️⃣: Restricting Access with Firewall Rules**

🔹 We define **firewall rules** to ensure security:

A screenshot of a computer

AI-generated content may be incorrect.

✅ **Why?**  
✔️ Ensures only **necessary services** can communicate.  
✔️ Prevents **unauthorized external access** to backend and database.

Users

│

▼

┌───────────────────┐

│ Cloud DNS (www.myecommerce.com) │

└────────┬──────────┘

│

┌──────────▼───────────┐

│ HTTPS Load Balancer │

└──────────┬───────────┘

│

┌────────▼────────┐

│ App Engine (React) │

│ (Auto-Scaling Frontend) │

└────────┬────────┘

│ Internal Traffic

▼

┌───────────────────────────────────┐

│ Internal Load Balancer (ILB) │

└───────────────────────────────────┘

│

▼

┌───────────────────────────┐

│ Backend API (FastAPI) │

│ Compute Engine (Auto-Scaling) │

└──────────┬───────────┘

│

┌────────────▼───────────────┐

│ Google Cloud Pub/Sub │

│ (Event-Driven Order System) │

└────────────┬───────────────┘

│

┌──────────▼───────────┐

│ Order Processing Worker │

│ (Cloud Function/VM) │

└──────────┬───────────┘

│

┌──────────────▼──────────────┐

│ Cloud SQL (PostgreSQL) │

│ Secure DB, Only API Access │

└──────────────────────────────┘

┌──────────────────────────────────┐

│ External Payment Gateway (Stripe)│

└───────────────┬──────────────────┘

│ Secure Outbound Connection

┌───────────▼───────────┐

│ Cloud NAT (No Public IPs) │

|  |
| --- |
|  |

**Key Takeaways**

✅ **Scalability**: App Engine auto-scales, and Compute Engine backend scales dynamically.  
✅ **Security**: No public IPs for backend and database, using **ILB and Cloud NAT**.  
✅ **Efficiency**: Pub/Sub enables **asynchronous** processing of orders.  
✅ **Cost-Effectiveness**: Managed services **reduce operational overhead**.