**📌 Real-Time Scenario: Secure Multi-Tier Web Application on Google Cloud**

**📝 Problem Statement**

A **financial services company** wants to deploy a **secure and scalable web application** on **Google Cloud**. The application consists of:  
1️.**Frontend Web App** (React)  
2️.**Backend API** (Python FastAPI)  
3️.**Database** (PostgreSQL)

The **security and connectivity** requirements are:  
✅ Users should access the application via a **domain name (DNS)**.  
✅ Backend API should **not be exposed to the internet** but should be accessible from the frontend.  
✅ Database should be **completely private** with access only from the backend.  
✅ The application should be able to **fetch updates from external APIs** but not be exposed to public access.  
✅ The company has an **on-prem data center** that should securely connect to cloud resources.

**🌐 Solution: Implementing Cloud Networking Components**

We'll use **VPC, Cloud VPN, Interconnect, Cloud NAT, ILB, DNS, and Firewall Rules** to design a **secure, scalable, and efficient cloud architecture**.

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

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

A screenshot of a phone

AI-generated content may be incorrect.

✅ **Why?** This ensures logical separation of frontend, backend, and database for better security.

**Step 2️⃣: Securely Connecting the On-Prem Data Center**

🔹 The company has an **on-prem data center** that needs to connect with the cloud securely.  
🔹 Two options:

* **Cloud VPN** → If a secure but cost-effective **encrypted connection** over the internet is acceptable.
* **Interconnect** → If **high-speed, low-latency private connectivity** is required.

🚀 **Solution Chosen:** We use **Cloud VPN** for a secure connection between the on-prem network (192.168.1.0/24) and the cloud VPC (10.1.0.0/16).

✅ **Why?**  
✔️ **On-prem database syncs securely** with cloud systems.  
✔️ Cloud services (backend API, analytics) can **access on-prem resources**.

**Step 3️. Allowing Secure Access to the Frontend App**

🔹 The **React frontend** should be accessible to users via www.mycompany.com.  
🔹 We register www.mycompany.com in **Google Cloud DNS** and map it to a public IP.  
🔹 A **Load Balancer** routes traffic to Compute Engine instances running the frontend.

✅ **Why?**  
✔️ Users access the app using a **domain name (DNS)** instead of an IP.  
✔️ **Load Balancer** distributes requests across multiple frontend servers for high availability.

**Step 4️: Securing Backend API with Internal Load Balancer (ILB)**

🔹 The **backend FastAPI service** should be accessible **only from the frontend** and not from the internet.  
🔹 We use an **Internal Load Balancer (ILB)** to distribute requests to backend instances.

✅ **Why?**  
✔️ Backend API **remains private** (no public exposure).  
✔️ **Frontend can access the API** via the ILB’s **private IP address**.

**Step 5️: Enabling Backend to Access External APIs Securely**

🔹 The backend needs to fetch stock market data from an **external API**.  
🔹 **Problem:** Backend is in a **private subnet** (no public IP).  
🔹 **Solution:** Use **Cloud NAT** to allow **outbound internet traffic** while keeping the backend private.

✅ **Why?**  
✔️ Backend can **call external APIs** (e.g., stock market, payment gateway).  
✔️ No need to **assign public IPs**, ensuring **better security**.

**Step 6️: Restricting Database Access with Firewall Rules**

🔹 The PostgreSQL database in db-subnet should **only be accessible from the backend API**.  
🔹 **Firewall Rule**:  
✅ **Allow traffic from backend-subnet to db-subnet on port 5432 (PostgreSQL)**.  
❌ **Deny all external traffic** to the database.

✅ **Why?**  
✔️ Protects the database from unauthorized access.  
✔️ Ensures only the backend API can query the database.