Scenario 5 – Run microservices application

In this exercise, you will work with a microservices-based application for **Online Boutique**, an e-commerce platform experiencing rapid growth. The company has decided to deploy its services in separate Docker networks for better scalability, security, and performance.

You will need to connect 10 microservices across different Docker networks and ensure that they communicate securely and efficiently. This exercise will require using both Docker commands and Docker Compose, with a focus on the manual orchestration of services first, followed by an automated Docker Compose solution in later upcoming exercises.

Note: This exercise is comprehensive and will require time to implement. Make sure to carefully review the project structure provided and the service details. Each service contains its own Dockerfile, which you will use to build Docker images.

There are additional services other than the below mentioned services. Please make sure to run the services below only as you may run into several errors if tried to start other services in the project as it is dependent on some GCP deployments.

Requirements

Your task is to connect 10 microservices across different Docker networks and ensure that they communicate with each other securely and efficiently. The services should be separated based on their network exposure (public-facing vs. private/internal).

As a part of this exercise, you need to:

1. Setup a docker compose file to build and run the application all at once
2. Use separate docker commands to build individual services and run each service separately.

You can do either of these approaches or both of them as well.

The following are the services to be managed:

Public-Facing Services:

1. **Frontend**: This service should be publicly exposed and must communicate with various internal services.

Internal Services (Private Network):

1. **Ad Service**: Provides ad-related data and must only be accessible internally.
2. **Cart Service**: Handles user cart information and should remain private.
3. **Checkout Service**: Manages order processing and requires several internal services to function.
4. **Currency Service**: Manages currency conversions and should not be publicly exposed.
5. **Email Service**: Sends email notifications; must only be accessible by other services.
6. **Payment Service**: Processes payments securely; should not be exposed externally.
7. **Product Catalog Service**: Provides product data to the frontend.
8. **Recommendation Service**: Suggests products based on user interactions.
9. **Shipping Service**: Provides shipping estimates and must be kept private.

Docker Networks:

- Create a **public network** for services that will be exposed to users, such as the **Frontend**.
- Create a **private network** for internal services like **Payment**, **Checkout**, **CartService** etc. that should not be exposed publicly.

Key Tasks:

1. **Docker Network Setup**: Create the appropriate networks public and private using Docker.
2. **Docker Image Builds**: Build Docker images for each of the services using the provided Dockerfiles.
3. **Service Communication**: Ensure services communicate across networks as needed, with internal services staying secure.
4. **Public Exposure**: Expose only the **Frontend** service publicly while keeping all other services private.
5. **Service Precedence**: It is important to understand which services can be started first so that other services don't fail. Order is important.

Acceptance Criteria:

- Docker networks for **public** and **private** services are created.
- Docker images for all services are built successfully.

- Proper service communication is established, ensuring internal services are not exposed publicly.
- You should be able to open the web app on browser locally.
- Frontend service is exposed correctly, while internal services like **Payment**, **Shipping** etc are only accessible internally.
- The solution should be implemented with docker commands only.

Resources

- Docker Networking Documentation
- The microservices demo repository can be found at Online Boutique Repository.

Hints

- **Plan the Network Architecture**: Understand which services need to communicate with each other and decide which should remain private.
- **Build Docker Images**: Build Docker images manually using docker build for each service.
- **Expose Services**: Carefully plan which services need to be publicly accessible (like **Frontend**) and which ones should remain isolated (like **Payment**).
- **Docker Compose**: After manually deploying the services, simplify the process using Docker Compose.