YUMMY – ONLINE FOOD ORDERING AND DELIVERY SYSTEM

LAB 8 REPORT – Deployment View & Quality Attribute Analysis (ATAM)

I. Summary

Lab 8 is the final lab of the Software Architecture course and shifts the focus from implementation to architecture documentation and evaluation. Building upon the Microservices Architecture designed in Labs 4–7, this lab aims to visualize the physical deployment of system components and evaluate how the selected architecture satisfies critical Non-Functional Requirements (NFRs).

The lab applies a simplified Architecture Trade-off Analysis Method (ATAM) to assess architectural decisions related to Scalability and Availability, and to compare the Microservices Architecture with the earlier Monolithic (Layered) approach. This evaluation helps validate whether the chosen architecture is appropriate for a real-world, high-traffic online food ordering system such as Yummy.

II. UML Deployment Diagram (Deployment View)

1. Objective

The UML Deployment Diagram illustrates how software artifacts are physically deployed onto execution environments such as servers, containers, or clusters. This view emphasizes runtime concerns including load distribution, fault isolation, and service independence, which are essential for evaluating quality attributes like scalability and availability.

2. Identified Nodes (Physical Environments)

Based on common cloud deployment patterns, the following nodes are defined:

- **Client Device**
  Represents the user's browser or mobile application used by customers, restaurants, drivers, and administrators to interact with the Yummy system.
- **Load Balancer (Nginx / Cloud Load Balancer)**
  Acts as the system's entry point, distributing incoming HTTP/HTTPS requests across backend services to ensure load balancing and high availability.
- **Application Cluster (Kubernetes / Virtual Machines)**
  Represents the execution environment hosting backend microservices. Services are deployed as independent containers, allowing flexible scaling and isolation.

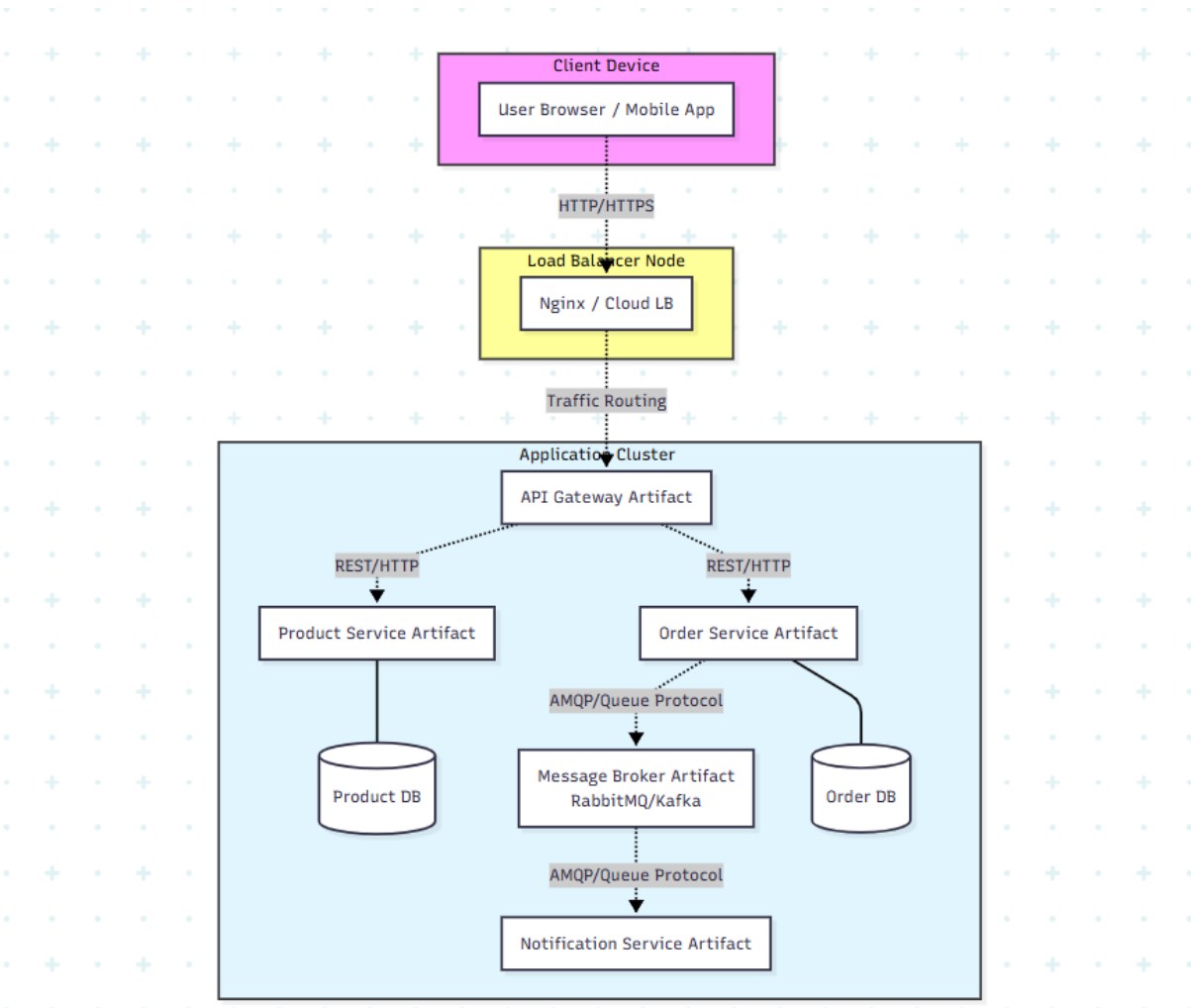

Figure 8.1: UML Deployment Diagram of the Yummy Microservices Architecture

Figure 8.1 illustrates the physical deployment of the Yummy microservices architecture, including the client device, load balancer, API gateway, independent backend services, and separate databases. Asynchronous communication via a message broker enhances scalability and fault isolation.

3. Software Artifacts (Services)

The following artifacts are deployed within the Application Cluster:

- **API Gateway Artifact**
  Central access point that routes requests from clients to internal services, handles authentication, and applies cross-cutting concerns.
- **Product Service Artifact**
  Manages food items, menus, and restaurant product information.
- **Order Service Artifact**
  Handles order creation, order processing, and order status management.
- **Notification Service Artifact**
  Responsible for sending notifications to users and drivers via external messaging services.
- **Message Broker Artifact (RabbitMQ / Kafka)**
  Enables asynchronous, event-driven communication between services, especially for non-blocking operations.

4. Data Stores

Each microservice owns a separate database instance, following the *Database per Service* principle:

- Product Service → Product Database
- Order Service → Order Database

This separation prevents tight coupling between services and allows independent scaling and evolution.

5. Communication Associations

The following communication paths are modeled using dashed arrows:

- Client Device → Load Balancer (HTTP/HTTPS)
- Load Balancer → API Gateway (Traffic routing)
- API Gateway → Product Service (Internal REST/HTTP)
- Order Service → Message Broker (AMQP / Queue protocol)
- Message Broker → Notification Service (Asynchronous messaging)

III. Quality Attribute Analysis (Simplified ATAM)

1. Selected Quality Attributes

Based on the Non-Functional Requirements identified in Lab 1, this lab focuses on:

- Scalability
- Availability

2. Defined Scenarios

Scalability Scenario (SS1)

During a 5-minute Black Friday promotion, the system must handle a sudden 10x increase in concurrent users viewing products and adding items to carts.

Availability Scenario (AS1)

The Notification Service fails completely for 1 hour due to a deployment error, while the system must continue to accept and process new orders.

3. Architecture Evaluation Table

Quality Attribute	Scenario	Monolithic (Layered) Approach	Microservices Approach
Scalability	SS1 (10x user spike)	Must scale the entire application instance, including UI, business logic, and database, even if only product browsing is affected. Inefficient resource usage.	Can independently scale Product Service and Cart/Order Service. Databases can be replicated or sharded per service. Efficient and flexible scaling.
Availability	AS1 (Notification Service fails)	Notification logic may be tightly coupled with core order processing, potentially causing failures or delays in order transactions.	Event-driven architecture ensures Order Service publishes events to Message Broker. Notification failure does not impact order processing. High fault isolation.

Table 8.1: Architecture Evaluation Table Using ATAM.

IV. Architecture Trade-offs

The ATAM analysis reveals the following key trade-offs:

Microservices Architecture provides superior scalability and availability, especially through independent service scaling and fault isolation enabled by asynchronous communication. However, these benefits come at the cost of increased deployment and operational complexity, requiring technologies such as Docker, Kubernetes, API Gateway, and Message Broker.

In contrast, the Monolithic (Layered) Architecture is simpler to develop and deploy, but sacrifices resilience, scalability, and flexibility, making it less suitable for large-scale, high-traffic systems like Yummy.

V. Conclusion

Lab 8 completes the architectural design lifecycle of the Yummy system by validating architectural decisions against key quality attributes. Through the Deployment Diagram and simplified ATAM analysis, it is evident that the Microservices Architecture better satisfies the system's scalability and availability requirements compared to a Monolithic approach.

Despite increased complexity, Microservices provide the necessary flexibility, fault tolerance, and performance needed for a modern online food ordering and delivery platform. This lab confirms that the chosen architecture is appropriate for real-world deployment and long-term system evolution.