PHENIKAA UNIVERSITY

SCHOOL OF COMPUTING

Lab Report: Lab 08 – Deployment View & Quality Attribute Analysis (ATAM) for a Real-Time Chat Application using Event-Driven and gRPC

Student Details: Hoàng Lê Đức Huy - 23010298 - Group 8

Nguyễn Hà Nguyên - 23010310 - Group 8

Nguyễn Đức Minh - 23010302 - Group 8

Course Info: Software Architecture

Ha Noi, January 22, 2026

Table of Contents

List of Figures

1. Abstract

This lab focuses on documenting the physical deployment architecture and evaluating key quality attributes of the Chat Real-time Application using a simplified Architecture Trade-off Analysis Method (ATAM). Building upon the outcomes of previous labs—from requirements elicitation and layered design to microservices decomposition and event-driven communication—this lab shifts the emphasis from logical design to execution environments and runtime quality evaluation.

A UML Deployment Diagram is constructed to illustrate how clients, a dedicated Realtime Gateway (WebSocket Gateway), microservices, message brokers, and databases are deployed across physical and virtual nodes. The Realtime Gateway plays a critical role in managing persistent connections, routing real-time messages, and isolating connection-level load from backend services. The deployment architecture also reflects the use of gRPC for efficient inter-service communication, RabbitMQ for asynchronous event processing, and Redis for distributed state management and connection tracking.

In addition, scalability and availability scenarios are defined and used to compare Monolithic and Microservices architectures. The analysis demonstrates that while Microservices introduce operational complexity, they provide superior scalability, fault isolation, and resilience, which are critical for a real-time chat system.

2. Lab Specific Section: Deployment View & ATAM Analysis

2.1 Activity Practice 1 – UML Deployment Diagram (Deployment View)

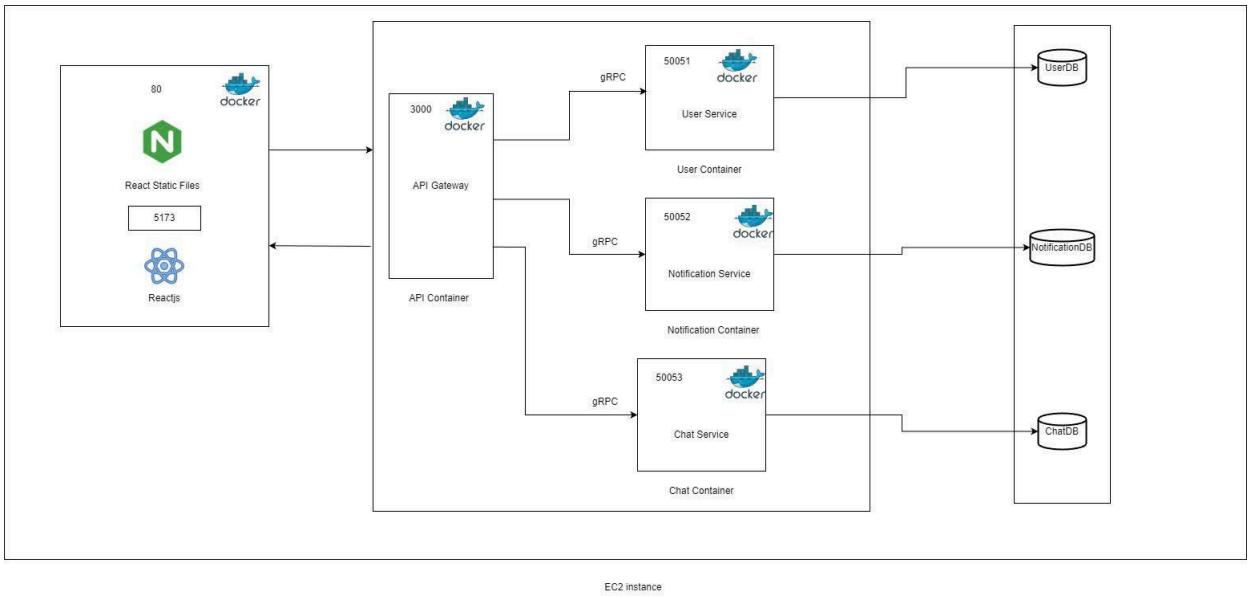

Figure 1: Deployment Diagram

The Deployment Diagram highlights the role of the Realtime Gateway as a specialized entry point for real-time communication. Client applications (Web and

Mobile) establish persistent WebSocket connections with the Realtime Gateway, which is deployed as a horizontally scalable service behind a Load Balancer.

The Realtime Gateway is responsible for WebSocket connection management and real-time message delivery, while the API Gateway handles REST-based client requests, authentication, and service orchestration via gRPC. Redis is deployed as a distributed state store to manage online user status, active socket connections, and rate limiting across multiple gateway instances. It forwards chat messages to the Chat Service via synchronous APIs or publishes events to the Message Broker for asynchronous processing. By decoupling connection handling from core business logic, the Realtime Gateway reduces load on backend services and improves overall system scalability and availability.

2.2 Activity Practice 2 – Quality Attribute Analysis – Simplified ATAM

2.2.1 Selected Quality Attributes

From the system's Non-Functional Requirements, two Architecturally Significant Quality Attributes are selected for evaluation:

- Scalability: The ability to handle a large number of concurrent users and messages.
- Availability: The ability to continue core operations despite partial system failures.

These attributes are critical for a real-time chat application with unpredictable traffic patterns. In particular, the Realtime Gateway is a key architectural element that directly influences both scalability and availability. Its ability to manage thousands of concurrent WebSocket connections independently from backend services is essential for maintaining real-time responsiveness under high load.

Scalability and availability are closely related to performance requirements, particularly low-latency message delivery, which is achieved through WebSocket-based communication and efficient gRPC interactions between services.

2.2.2 Defined ATAM Scenarios

Scalability Scenario (SS1):

- During a peak event, the number of concurrent users connected through the Realtime (WebSocket) Gateway increases by a factor of ten. Users simultaneously send and receive messages across multiple group chats, resulting in a sharp increase in persistent connections and message throughput.

Availability Scenario (AS1):

- The Notification Service becomes unavailable for one hour due to a deployment failure. The Chat Service continues publishing message

events to RabbitMQ, and the Realtime Gateway continues delivering messages to connected clients without interruption.

These scenarios explicitly define the stimulus, system response, and quality attribute measures, aligning with the core principles of the ATAM method.

2.2.3 Evaluation of Architectural Approaches

- Scalability – SS1
    - Monolithic (Layered) Architecture:
      The entire application must be scaled as a single unit, including user management, messaging logic, and persistence. This leads to inefficient resource usage, as not all components experience the same load.
    - Microservices Architecture:
      Only high-pressure components such as the Realtime Gateway and Chat Service need to scale horizontally. Message brokers and databases can be independently replicated or sharded. This targeted scaling is well-suited for real-time workloads.
    - The presence of a dedicated Realtime Gateway further enhances scalability by isolating connection-heavy workloads from backend services. The gateway can be replicated independently to handle large volumes of WebSocket connections, while the Chat Service focuses solely on message processing and business logic. This separation prevents connection spikes from overwhelming core services.
    - The use of gRPC for inter-service communication reduces payload size and improves internal performance, while Redis enables efficient distributed connection and state management under high concurrency.

- Availability – AS1
    - Monolithic (Layered) Architecture:
      If notification logic is embedded within the monolith, failures may block or slow down the message processing pipeline, reducing overall system availability.
    - Microservices Architecture:
      Thanks to the event-driven design, Chat Service publishes events to the Message Broker without depending on the Notification Service's availability. Notification failures do not affect message delivery, ensuring high fault isolation.
    - Moreover, the Realtime Gateway improves availability by acting as a buffer between clients and backend services. Even if downstream services such as Notification Service experience failures, active

WebSocket connections remain intact, allowing the system to continue accepting and routing chat messages in real time.

2.2.4 Identified Architectural Trade-offs

The identification of architectural trade-offs is a direct outcome of the ATAM evaluation rather than a standalone practice. Based on the analysis of the Scalability and Availability scenarios, the following trade-offs are observed:

- Benefits of Microservices:
  - Fine-grained scalability for real-time messaging components such as the WebSocket Gateway and Chat Service.
  - Strong fault isolation enabled by asynchronous, event-driven communication.
  - Better alignment with high-availability requirements for real-time systems.
  - Clear separation between connection management (Realtime Gateway) and business logic, improving scalability and fault isolation.

- Costs of Microservices:
  - Increased deployment and operational complexity.
  - Dependence on container orchestration platforms, service discovery, and distributed monitoring.
  - Additional complexity in managing stateful WebSocket connections in a distributed environment, requiring Redis for socket state synchronization and monitoring.

In contrast, the Monolithic (Layered) architecture offers lower operational complexity but sacrifices resilience and scalability, making it less suitable for a real-time chat application.

2.2.5 Summary of ATAM Results

Based on the evaluation of the defined ATAM scenarios (SS1 and AS1), the analysis demonstrates that the Microservices architecture consistently outperforms the Monolithic (Layered) architecture with respect to the selected quality attributes. In the Scalability scenario, Microservices enable selective horizontal scaling of load-intensive components, which is essential for handling sudden spikes in concurrent WebSocket connections. In the Availability scenario, the event-driven interaction between services ensures that failures in auxiliary services, such as the Notification Service, do not propagate to core messaging functionality. Overall, the ATAM results confirm that the chosen Microservices and Event-Driven Architecture effectively satisfies the architecturally significant requirements of a real-time chat system and provides a more suitable foundation for real-world deployment.

3. Conclusion & Reflection

Lab 8 completes the architectural journey of the Chat Real-time Application by shifting focus from design and implementation to deployment and quality evaluation. The UML Deployment Diagram clarifies how microservices, gateways, and messaging infrastructure interact at runtime. The simplified ATAM analysis confirms that the Microservices and Event-Driven Architecture chosen in earlier labs effectively satisfies the system's critical quality attributes, particularly scalability and availability.

While the architecture introduces additional complexity, this trade-off is justified for a real-time communication platform where performance, fault tolerance, and user experience are paramount. Overall, Lab 8 validates the architectural decisions made throughout the project and provides a strong foundation for real-world deployment and future evolution. A key architectural insight from this lab is the importance of the Realtime Gateway, which serves as a critical enabler for scalable and highly available real-time communication by decoupling persistent connection handling from backend microservices.

The integration of Redis for connection state management, RabbitMQ for asynchronous event handling, and the separation between API Gateway and Realtime Gateway further validates the architectural decisions made throughout the project.