PHENIKAA UNIVERSITY

PHENIKAA SCHOOL OF COMPUTING

COURSE: SOFTWARE ARCHITECTURE

TOPIC:

DEVLOPMENT OF TRAINING PROGRAM MANAGEMENT SOFTWARE

REPORT: Lab 8 - Deployment View & Quality Attribute Analysis (ATAM)

Code Course	: CSE703110-1-1-25
Class	: N01
Instructor	: THS. Vũ Quang Dũng
Group	: 12
Members	: 1. Đồng Đại Đạt : KTPM.EL2 - 23010877
	2. Nguyễn Cao Hải : KTPM.EL2 - 23013124
	3. Ngô Nhật Quang : KTPM.EL1 - 23010134
	4. Nguyễn Nam Khánh : KTPM.EL2 - 23010771

Hanoi, November 2025

Mục Lục

1. Abstract

This laboratory focuses on analyzing the system architecture from the perspectives of deployment and quality attributes. The system under consideration is a Training Curriculum Management System designed using a modular monolithic architecture combined with a supporting microservice and asynchronous communication.

The objectives of this lab include constructing a deployment diagram to describe how system components are deployed on the infrastructure, and performing a simplified Architecture Trade off Analysis Method analysis. The ATAM analysis concentrates on two important quality attributes, scalability and availability, in order to evaluate architectural decisions and understand the tradeoffs between monolithic and microservices architectures.

2. Objectives

The main objectives of this lab are to describe the physical deployment of the system using a UML deployment diagram, to analyze how architectural decisions affect system quality attributes, and to apply a simplified ATAM approach focusing on scalability and availability. In addition, this lab aims to compare monolithic and microservices architectures and justify the chosen architectural style for the system.

3. System Overview and Architectural Context

3.1 Background

The Training Curriculum Management System is developed to support academic and training related activities such as course management, user management, enrollment, scheduling, assessment, and notification.

From an architectural point of view, the system is implemented using a modular monolith for the main backend application. Core modules such as Course, User, and Enrollment are logically separated within the same application but are deployed together as a single unit. In addition, a Notification Service is implemented as a separate microservice to handle email and notification delivery. Communication between the main application and the notification service is performed asynchronously through a message broker.

This architecture reflects a hybrid approach that balances development simplicity and system flexibility.

4. Deployment Diagram

4.1 Deployment Environment

The system is assumed to be deployed on a single physical server or virtual machine using Docker containers. Docker Compose is used to manage and orchestrate the containers.

The deployment includes the following main nodes. A Client node represents user devices such as web browsers or mobile applications. Users interact with the system through HTTP or HTTPS.

A Server node represents the Docker host machine. Inside this server, multiple containers are deployed. The Main Application container runs the Django backend and includes modules for Course, User, and Enrollment management. A PostgreSQL database container stores persistent data for the main application. A RabbitMQ container acts as a message broker for asynchronous communication. The Notification Service container runs independently and is responsible for processing events and sending notifications.

4.2 Communication Between Nodes

Clients communicate directly with the Main Application using RESTful APIs over HTTP or HTTPS. The Main Application communicates with the PostgreSQL database through a database connection. When certain events occur, such as successful enrollment, the Main Application publishes messages to RabbitMQ. The Notification Service consumes these messages from RabbitMQ and performs notification related tasks.

This deployment design separates user facing operations from background processing and improves system resilience.

5. Architecture Trade Off Analysis Method Overview

Architecture Trade Off Analysis Method is a structured approach used to evaluate architectural decisions based on quality attributes. In this lab, a simplified ATAM process is applied. The analysis focuses on identifying how architectural choices affect scalability and availability and what tradeoffs arise when comparing monolithic and microservices architectures.

6. ATAM Analysis for Scalability

6.1 Scalability Scenario

The scalability scenario considered in this analysis is an increase in the number of users accessing the system simultaneously, for example during peak enrollment periods. The system should be able to handle increased load while maintaining acceptable performance.

6.2 Monolithic Architecture and Scalability

In a traditional monolithic architecture, the entire application is deployed and scaled as a single unit. When one module experiences high load, the whole application must be replicated even if other modules do not require additional resources. This approach often leads to inefficient resource usage. Furthermore, a shared database can become a bottleneck when the number of concurrent users grows significantly.

Although monolithic systems are simple to deploy and manage, their scalability is limited when workload is uneven across modules.

6.3 Microservices Architecture and Scalability

In a microservices architecture, each service can be scaled independently based on its workload. Services that experience high demand can be replicated without affecting other services. This approach allows more efficient use of resources and provides better support for large scale systems.

However, microservices introduce additional complexity in deployment, monitoring, and service orchestration. Network communication between services can also add latency.

6.4 Scalability Decision in the System

The system adopts a modular monolith for core functionality while extracting the Notification Service as a separate microservice. This allows notification processing to scale independently when message volume increases, without scaling the entire backend application.

This decision provides sufficient scalability for current system requirements while keeping architectural complexity manageable.

7. ATAM Analysis for Availability

7.1 Availability Scenario

The availability scenario considered is the failure of a non-critical component, such as the notification functionality. The system should continue to support core user operations even when such failures occur.

7.2 Monolithic Architecture and Availability

In a monolithic architecture, components are tightly coupled within a single application. A serious failure in one part of the system can cause the entire application to crash, resulting in complete service downtime. Maintenance and deployment activities often require stopping the entire system, which reduces availability.

7.3 Microservices Architecture and Availability

Microservices architectures provide better fault isolation. When one service fails, other services can continue operating. Services can also be deployed and updated independently, which reduces downtime.

However, distributed systems introduce additional failure points such as network issues or message broker failures. These risks require proper monitoring and recovery mechanisms.

7.4 Availability Decision in the System

By separating the Notification Service from the main application, failures related to notification delivery do not affect core system functionality. Users can still register for courses and manage their data even if notifications are temporarily unavailable.

This design improves overall system availability while keeping the core application stable.

8. Discussion

The architecture of the Training Curriculum Management System represents a practical compromise between simplicity and robustness. A fully microservices based architecture would improve scalability and availability but would significantly increase operational complexity. A pure monolithic architecture would be easier to manage but less resilient to failures.

The hybrid approach used in this system allows the team to achieve acceptable scalability and availability while maintaining development and deployment simplicity.

9. Conclusion

This lab presented a deployment diagram that illustrates how system components are deployed in a containerized environment. A simplified ATAM analysis was conducted to evaluate the system architecture in terms of scalability and availability.

The analysis shows that monolithic architectures are suitable for small to medium sized systems due to their simplicity, while microservices architectures provide better scalability and availability at the cost of increased complexity. The chosen modular monolith combined with a supporting microservice is appropriate for the current system requirements and allows future architectural evolution if system scale increases.