ESSAY DESCRIPTION

WEB PROGRAMMING WITH NODEJS - 502070

SEMESTER 1 – ACADEMIC YEAR 2024 – 2025

Lecturer: Mai Van Manh

Overview and Purpose

The purpose of this essay is to provide students with a comprehensive understanding of **Docker**, a powerful platform for developing, shipping, and running applications inside containers. Through this essay, students will explore the theoretical foundations of Docker and demonstrate practical skills by Dockerizing a Node.js web application. The objective is to help students learn how to use Docker for deploying services and web applications in a scalable and maintainable way.

By the end of the assignment, students should be able to:

- Understand the core concepts of Docker and containerization.
- Use Docker to deploy a multi-service web application.
- Scale applications using Docker Compose or other container orchestration tools.
- Apply advanced concepts such as load balancing, decoupling services, and orchestration.

Task Description

Students will work in groups (2-3 members) to:

- Research and understand the theoretical aspects of Docker, including its architecture, advantages, and use cases in modern software development.
- Implement a practical demonstration (lab) using Docker and Docker Compose. Advanced groups are encouraged to explore container orchestration tools such as Docker Swarm, Kubernetes or ECS.

Structure of the Report

The report must adhere to the department's standard format and include the following sections:

1. Introduction to Docker:
    - Define what Docker is and explain why it is important in modern web development.

- Introduce key concepts such as containers, images, Dockerfile, Docker Compose, and container orchestration.

2. Theoretical Survey:
   - Explain the architecture of Docker and its underlying technology (e.g., namespaces, cgroups, union file systems).
   - Discuss how Docker simplifies the development and deployment of Node.js applications.
   - Cover the benefits and challenges of containerization in web development (such as scalability, isolation, and deployment automation).
3. Project Overview:
   - Introduce the specific project your group has chosen to demonstrate.
   - Explain the project's context and why Docker is used in this scenario.
   - Describe the services involved (e.g., front-end, back-end, database) and how they interact.
4. Architecture and Implementation:
   - Provide a detailed explanation of the architecture designed by your group, focusing on how Docker and Docker Compose were used.
   - Include a clear description of the services deployed and their interactions, showing the Docker Compose file (if used) and Dockerfiles.
   - For advanced groups, describe how you implemented scaling, load balancing, and service decoupling.
   - If orchestration tools (e.g., Docker Swarm or Kubernetes) were used, explain their role in the project and how they enhanced the deployment.
5. Results and Discussion:
   - Present the outcomes of your Dockerization project.
   - Discuss the performance, scalability, and reliability of your solution.
   - Highlight any challenges faced and how they were overcome.
6. Conclusion:
   - Summarize the key learnings from this project.
   - Reflect on how Docker can enhance the development and deployment of modern web applications.

Levels of Difficulty

Each group must select a challenge level to demonstrate their understanding of Docker:

- Level 1:
  - Deploy a simple Docker setup using Docker Compose.

- The system must include at least 3 services: a front-end (e.g., HTML, React, Angular), a back-end (Node.js), and a database (e.g., MySQL, MongoDB).
- Ensure these services are in the same Docker network and communicate with each other.

- Level 2:
  - Includes all requirements from Level 1.
  - Demonstrate a scalable system by replicating a service (e.g., scaling the back-end service into multiple containers) and implementing load balancing between them.
  - Apply decoupling between services using intermediaries like RabbitMQ or Redis for messaging or caching purposes.
- Level 3:
  - Includes all requirements from Level 2.
  - Demonstrate how the project can be deployed on a container orchestration platform such as Docker Swarm or Kubernetes .
  - Explain how orchestration improves scalability, availability, and service management.

Deliverables

Each group must submit the following deliverables:

1. **Complete Report (Word or PDF)**
   - Ensure the report is structured and formatted according to departmental guidelines.
   - Cover all sections mentioned above, including the theoretical survey, project description, architecture, and results.
2. **Source Code and Dependencies**
   - Submit all the necessary source code for the Node.js web application.
   - Include Dockerfiles, Docker Compose files, and any other configuration files.
   - Ensure all dependencies (e.g., package.json) are provided and that the project is ready to be built and deployed using Docker.
3. **Video Presentation**
   - Create a video that explains your project and demonstrates your Dockerized application.
   - Walk through how your solution was built, deployed, and scaled.
   - The video should cover all the main features of your project and provide insights into how you overcame any challenges.
4. **ReadMe.txt**
   - Provide a clear and concise ReadMe.txt file that includes instructions for running the project locally.
   - Include details on how to build the images, run the containers, and test the application.

Criteria	POINT	0	25%	50% - 75%	Full Score
REPORT	**6.0**				
Theoretical Understanding	2.0	No understanding of Docker or missing theoretical survey.	Basic understanding, covers a few Docker concepts but lacks detail or clarity.	Covers key concepts but lacks technical depth or explanation in some areas.	Comprehensive understanding of Docker, including in-depth coverage of all key concepts (e.g., images, containers, Docker Compose, orchestration).
Project Overview & Context	1.0	No project overview or an unclear explanation.	Basic project description but fails to explain service interactions or Docker's role clearly.	Partial project description, mentions services but lacks detail on interactions or why Docker was chosen.	Clear and well-explained project context, justifies Docker usage, and gives a complete explanation of service interactions.
Architecture and Implementation	1.0	No architecture or incorrect implementation of Docker/Compose.	Basic architecture, with some services defined, but missing key configurations or interactions between services.	Partial architecture and implementation; services defined but missing details like Dockerfile configuration or networking between containers.	Full and correct architecture, all services defined with clear Dockerfile, Docker Compose, and networking configurations, and accurate implementation.
Report Presentation*	2.0	No report or highly incomplete report.	Report is basic, lacks structure, or has significant formatting or content issues.	Mostly complete but contains some formatting errors, lacks detail in certain sections, or has inconsistent clarity.	Well-organized and professionally formatted report, follows required structure, and provides clear explanations and logical flow throughout.
DEMO	**4.0**	Without a video demonstration, this section will not be graded even if the source code is provided.			

Working Demo (Level 1)	2.5	No functional demo or incomplete Docker implementation.	Demo runs but services are not fully connected or functional (e.g., front-end, back-end, or database issues).	Demo mostly works but may have minor issues with service interaction or incomplete scaling.	Fully functional Dockerized application with working services (front-end, back-end, database) and correct inter-service communication.
Advanced Features (Level 2)	1.0	No advanced features implemented (no scaling, load balancing, or decoupling).	Basic scaling/load balancing implemented but lacks integration with intermediaries like Redis/RabbitMQ.	Partial implementation of advanced features (e.g., scaling, load balancing, service decoupling) but not fully functional or integrated.	Full implementation of advanced features: correct scaling, load balancing, and service decoupling (using Redis/RabbitMQ or similar intermediaries).
Advanced Features (Level 3)	0.5	No orchestration tools used.			Full and correct deployment using orchestration tools (Docker Swarm, Kubernetes or ECS) with demonstrated scalability and service management.

* A well-written report should avoid the following mistakes (the list is not exhaustive):

- The cover page was presented carelessly, with serious errors such as: using a substandard logo, incorrect instructor information, incorrect group member information, incorrect topic name.
- Missing necessary appendix pages as required. Not having enough chapters (or contents) as required such as: theoretical survey, requirement analysis, system design, presentation of results with necessary analysis/comparison, conclusion chapter of the report.
- Spelling errors; inconsistent font, color, and font size; failure to use appropriate margins and line spacing; overuse of bullet points in presentation.
- The report has a lot of text but lacks images, diagrams, tables, and charts to illustrate and make the content easier for readers to understand. This also shows the possibility of abusing AI tools to write reports.
- Lack of investment in presenting images, tables, and diagrams: for example, images that are too large or too small, fonts/background colors that are difficult to read, screenshots of redundant content, leading to a poorly presented report.
- Too much mention of source code while lacking description or explanation of the system, the operating principles of the components, or how the team implemented and deployed the system, makes it difficult for readers to understand what the team is working on, how they did it, what the results were, or what good things can be learned from this topic.

Late Submission Penalty: For every day that an assignment is submitted late, one point will be deducted from the total possible score. Please note that even 1 second late will be considered as one full day late.

Points may aslo be deducted in the following cases:

- The report or video presentation is not in English.
- Not all team members are present in the video presentation, unless previously approved.
- The demo video has poor audio or video quality, making it difficult to understand or follow.
- There is an uneven distribution of work among team members.
- The submission does not adhere to the specified format or is missing required information, such as:
    - Missing or incomplete team member details.
    - Insufficient descriptive information, such as failing to include instructions for running the source code or not providing the necessary admin account credentials to access the application.