Bill of Materials (BOM) trong quản lý dependencies cho microservices

Trong quá trình phát triển hệ thống microservices, việc quản lý phiên bản của các thư viện và dependencies là một yếu tố quan trọng. Để giảm thiểu sự lặp lại và đảm bảo tính nhất quán giữa các microservices, chúng ta có thể sử dụng Bill of Materials (BOM). BOM là một kỹ thuật cho phép quản lý các phiên bản thư viện và dependencies trong một tệp cấu hình chung, giúp các microservices sử dụng cùng một phiên bản của các thư viện mà không cần khai báo lại trong từng microservice.

1. **Quản lý phiên bản chung trong BOM:** BOM giúp chúng ta dễ dàng kiểm soát phiên bản của các dependencies trong các microservices. Mỗi microservice có thể tham chiếu tới một BOM chung, từ đó sử dụng các phiên bản thư viện đồng nhất mà không cần phải tự khai báo phiên bản trong mỗi tệp pom.xml của microservice. Điều này không chỉ giúp giảm thiểu sự trùng lặp mà còn đảm bảo sự nhất quán khi cập nhật phiên bản của thư viện.
2. **Tùy chỉnh phiên bản riêng cho microservice:** Tuy nhiên, trong một số trường hợp, một số microservices có thể yêu cầu sử dụng phiên bản khác của một thư viện. Khi đó, chúng ta có thể tùy chỉnh phiên bản của các dependencies cho mỗi microservice mà không làm ảnh hưởng đến các microservice khác. Cụ thể, ta có thể xóa bỏ sự tham chiếu đến phiên bản trong BOM và khai báo phiên bản mong muốn trực tiếp trong tệp pom.xml của microservice đó.
3. **Lợi ích khi sử dụng BOM:** Việc sử dụng BOM không chỉ giúp quản lý phiên bản mà còn mang lại những lợi ích khác như:
    - **Quản lý thư viện chung:** Nếu các microservices cùng sử dụng một thư viện chung, thay vì phải khai báo lại dependencies trong từng

microservice, ta có thể khai báo chúng trong BOM và các microservices sẽ tự động sử dụng.

- **Tiết kiệm thời gian:** BOM giúp giảm thiểu công sức phải thay đổi hoặc cập nhật các dependencies trong từng microservice, đặc biệt khi cần đồng bộ hóa phiên bản của thư viện.
- **Đảm bảo sự nhất quán:** Việc đồng bộ phiên bản thư viện giữa các microservices giúp tránh các vấn đề về sự không tương thích khi giao tiếp giữa các service.

4. **Tích hợp BOM vào Docker và việc xây dựng Docker Image:** Việc sử dụng BOM không ảnh hưởng đến quy trình xây dựng Docker image. Sau khi cài đặt BOM, quá trình tạo Docker image cho các microservices vẫn diễn ra bình thường mà không gặp phải lỗi. Các microservices vẫn sẽ tham chiếu đúng các dependencies đã được định nghĩa trong BOM. Chúng ta cũng có thể xác nhận Docker image đã được tạo thành công và container khởi chạy mà không gặp vấn đề gì.

Tạo và sử dụng thư viện chia sẻ trong microservices

Trong quá trình phát triển hệ thống microservices, việc tái sử dụng mã nguồn là rất quan trọng để đảm bảo tính mở rộng và giảm thiểu sự lặp lại trong các dịch vụ. Một trong những cách để thực hiện việc này là thông qua việc xây dựng các thư viện chia sẻ (shared libraries). Dưới đây là các bước chi tiết để tạo và sử dụng thư viện chia sẻ trong một hệ thống microservices sử dụng Spring Boot.

1. **Tạo Submodule cho Thư Viện Chia Sẻ (Common Module)**: Đầu tiên, tạo một submodule mới trong dự án eazy-bom với tên là common. Module này sẽ chứa mã nguồn chung mà các microservices khác có thể tái sử dụng. Các thông tin cơ bản cho module bao gồm:

- Tên module: common
- Ngôn ngữ: Java, Loại dự án: Maven, Packaging: JAR
- JDK: 21
- Các dependency cần thiết: Spring Web, Lombok

2. **Cấu Hình pom.xml của eazy-bom**: Sau khi tạo submodule, ta cần cấu hình tệp pom.xml của eazy-bom để bao gồm module con mới này. Thêm thẻ <modules> trong phần <properties> của pom.xml để khai báo submodule common.
3. **Cấu Hình pom.xml của Common Module**: Tiếp theo, trong pom.xml của module common, thay thế phần khai báo parent từ Spring Boot parent sang eazy-bom để đảm bảo sử dụng BOM của dự án eazy-bom. Đặt phiên bản cho module này thông qua một thuộc tính (property) trong tệp pom.xml của eazy-bom. Thêm các dependency cần thiết như Spring Web, Lombok, và Spring Doc (cho Open API).
4. **Di Chuyển Mã Nguồn Chia Sẻ**: Sau khi cấu hình xong, ta chuyển các lớp DTO chung như ErrorResponseDto từ các microservices riêng biệt vào module common. Để tránh việc chạy ứng dụng từ module chung, xóa bỏ lớp common application trong module này.
5. **Cập Nhật Các Microservices để Sử Dụng Common Module**: Mỗi microservice sử dụng thư viện chung cần thêm dependency vào tệp pom.xml của nó. Các thông tin dependency cho module common sẽ bao gồm groupId, artifactId, và version của module common.
6. **Giải Quyết Các Lỗi Biên Dịch**: Sau khi thêm dependency vào các microservices, tiến hành xây dựng lại các dịch vụ. Trong quá trình này, các lỗi

biên dịch có thể xuất hiện do IntelliJ không tự động nhận diện các thay đổi. Việc cần làm là cập nhật lại các câu lệnh import trong các lớp sử dụng ErrorResponseDto.

7. **Xây Dựng Docker Image**: Để kiểm tra quá trình xây dựng, tiến hành tạo Docker image cho microservice accounts. Sử dụng lệnh Maven maven compile jib docker build để xây dựng image. Nếu quá trình xây dựng không thành công, cần đảm bảo rằng module common đã được xuất bản đúng cách vào repository Maven địa phương bằng cách chạy lệnh mvn clean install từ thư mục eazybom.
8. **Đảm Bảo Common Module Được Publish Đúng**: Nếu có sự cố khi xây dựng Docker image, cần xác minh rằng module common đã được xuất bản vào Maven repository cục bộ. Điều này có thể thực hiện thông qua việc chạy lệnh mvn clean install trong thư mục chứa module common để đảm bảo nó có sẵn cho các dịch vụ khác.