HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

ĐỒ ÁN

TỐT NGHIỆP ĐẠI HỌC

ĐỀ TÀI:

TÌM HIỂU VÀ ỨNG DỤNG CÁC CÔNG NGHỆ BIÊN DỊCH VÀ THỰC THI CODE TRÊN MÁY CHỦ CHO CÁC ỨNG DỤNG HỌC TẬP LẬP TRÌNH TRỰC TUYẾN

Giảng viên hướng dẫn	**: TS. NGUYỄN TẤT THẮNG**
Sinh viên thực hiện	**: PHẠM VĂN TIẾN**
Lớp	**: D19CNPM3**
Mã sinh viên	**: B19DCCN582**
Khóa	**: 2019 – 2024**
Hệ	**: ĐẠI HỌC CHÍNH QUY**

HÀ NỘI – 1/2024

LỜI CẢM ƠN

Lời đầu tiên chính là lời cảm ơn chân thành nhất mà em xin được gửi tới ban giám hiệu nhà trường cùng tất cả các thầy cô của Học viện Công nghệ Bưu chính Viễn thông. Các thầy cô đã tạo nên một môi trường giáo dục không chỉ thành công về mặt giáo dục mà còn cả về các phong trào đoàn thể, các sự kiện giúp cho chúng em có được cơ hội phát triển tốt nhất.

Em xin cảm ơn tất cả những người thầy, người cô đã dạy dỗ em nên người và giúp em có được ngày hôm nay. Các thầy cô đã không quản ngại vất vả để truyền cho chúng em những kiến thức vô cùng quý giá. Em sẽ luôn luôn trân trọng mang theo những hành trang này trên con đường sự nghiệp nhiều chông gai phía trước.

Đặc biệt, em bày tỏ lòng biết ơn sâu sắc đến thầy TS. Nguyễn Tất Thắng. Thầy không chỉ là người đã dạy dỗ em trong các môn học mà thầy còn còn là người hướng dẫn, chỉ bảo tận tình em trong môn quá trình làm đồ án. Nếu không được thầy tạo điều kiện thuận lợi, cùng sự bảo ban của thầy, chắc chắn em sẽ không thể nào hoàn thành được đồ án.

Cuối cùng, em xin kính chúc quý thầy, cô, gia đình và bạn bè dồi dào sức khoẻ, thành công trong sự nghiệp.

Em xin chân thành cảm ơn!

Hà Nội, 26 tháng 12 năm 2023

Sinh viên

Phạm Văn Tiến

NHẬN XÉT, ĐÁNH GIÁ, CHO ĐIỂM

(Của người hướng dẫn)

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

Điểm:…… **(bằng chữ:** ……**)**

Đồng ý/Không đồng ý cho sinh viên bảo vệ trước hội đồng chấm đồ án tốt nghiệp?

……**, ngày**…..... **tháng**…..... **năm 2024**

CÁN BỘ GIẢNG VIÊN HƯỚNG DẪN

NHẬN XÉT, ĐÁNH GIÁ, CHO ĐIỂM

(Của giáo viên phản biện)

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

Điểm:…… **(bằng chữ:** ……**)**

Đồng ý/Không đồng ý cho sinh viên bảo vệ trước hội đồng chấm đồ án tốt nghiệp?

……**, ngày**…... **tháng**…..... **năm 2024**

CÁN BỘ GIẢNG VIÊN PHẢN BIỆN

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

LỜI NÓI ĐẦU

Lý do chọn đề tài

Thế kỷ 21 chứng kiến sự phát triển mạnh mẽ của ngành công nghệ thông tin nói chung và trí tuệ nhân tạo nói riêng. Gần đây, chatGPT, một chatbot do OpenAI phát triển dựa trên mô hình Transformer của Google đã nổi lên như một cuốn "bách khoa toàn thư" có thể nói chuyện như con người với con người và có thể đưa ra các câu trả lời chính xác cho rất nhiều lĩnh vực khác nhau. Ngạc nhiên hơn, ChatGPT đã có thể tự động viết code dựa theo các yêu cầu được đặt ra từ con người với tốc độ nhanh, độ chính xác cực kỳ cao và hầu như không có lỗi. Hơn nữa, sự phát triển của ChatGPT-4 giờ đây đã có thể viết các đoạn mã giao diện với đầu vào câu lệnh là một ảnh chụp màn hình, một công nghệ thật đáng kinh ngạc. Do vậy, nhiều lập trình viên đang lo ngại trong tương lai gần bản thân có thể bị thay thế bởi AI.

Trước tình cảnh này, các công việc lặp đi lặp lại của lập trình viên sẽ được AI "trợ giúp" để tăng năng suất công việc. Do đó, các lập trình viên giờ đây phải học sâu từ bên trong hệ thống để nắm vững bản chất hơn, đặc biệt là các thuật toán cơ bản và nâng cao. Tất cả các hệ thống lớn đều phải được xây dựng từ các thuật toán tốt. Do đó các lập trình viên cần một hệ thống học tập lập trình để có thể luyện tập nâng cao tư duy thuật toán nói chung và tư duy hệ thống nói riêng.

Chính những lý do trên việc chọn đề tài "Tìm hiểu và ứng dụng các công nghệ biên dịch và thực thi code trên máy chủ cho các ứng dụng học tập lập trình trực tuyến" rất thiết thực và phù hợp với nhu cầu hiện nay của lập trình viên.

Mục tiêu của đề tài

Nghiên cứu và xây dựng hệ thống biên dịch và thực thi sử dụng nền tảng Docker: Việc sử dụng một công nghệ mới để thực thi mã nguồn tạo ra một hệ thống hiệu năng cao, đáng tin cậy và có khả năng mở rộng mạnh mẽ.

Xây dựng website học tập lập trình: website học tập lập trình cần có các chức năng chính: Xem các bài tập lập trình với các chủ đề khác nhau và với độ khó tăng dần, xem gợi ý bài giải và các bài tập tương tự, hệ thống biên dịch và thực thi code tự động kiểm tra đáp án, trình soạn thảo code giúp người dùng triển khai ý tưởng ngay trên

website, thống kê số bài tập đã làm được và điểm số… Ngoài ra hệ thống còn các chức năng thường gặp như đăng nhập, đăng ký tài khoản.

Bố cục của đồ án

Nội dung của đồ án sẽ bao gồm 5 chương chính:

Chương 1: Cơ sở lý thuyết sẽ là chương giới thiệu các công nghệ sử dụng để xây dựng hệ thống.

Chương 2: Các vấn đề và phương pháp tiếp cận sẽ nêu ra các vấn đề về bảo mật khi xây dựng một hệ thống thực thi và biên dịch, qua đó đưa ra giải pháp phù hợp.

Chương 3: Xây dựng hệ thống biên dịch và thực thi sẽ mô tả kiến trúc của hệ thống biên dịch và thực thi, cách hệ thống biên dịch giao tiếp với hệ thống website.

Chương 4: Cài đặt hệ thống website học tập lập trình sẽ mô tả các bước cài đặt hệ thống và kết quả thực tế của hệ thống.

CHƯƠNG I: CƠ SỞ LÝ THUYẾT

1.1. Spring Boot

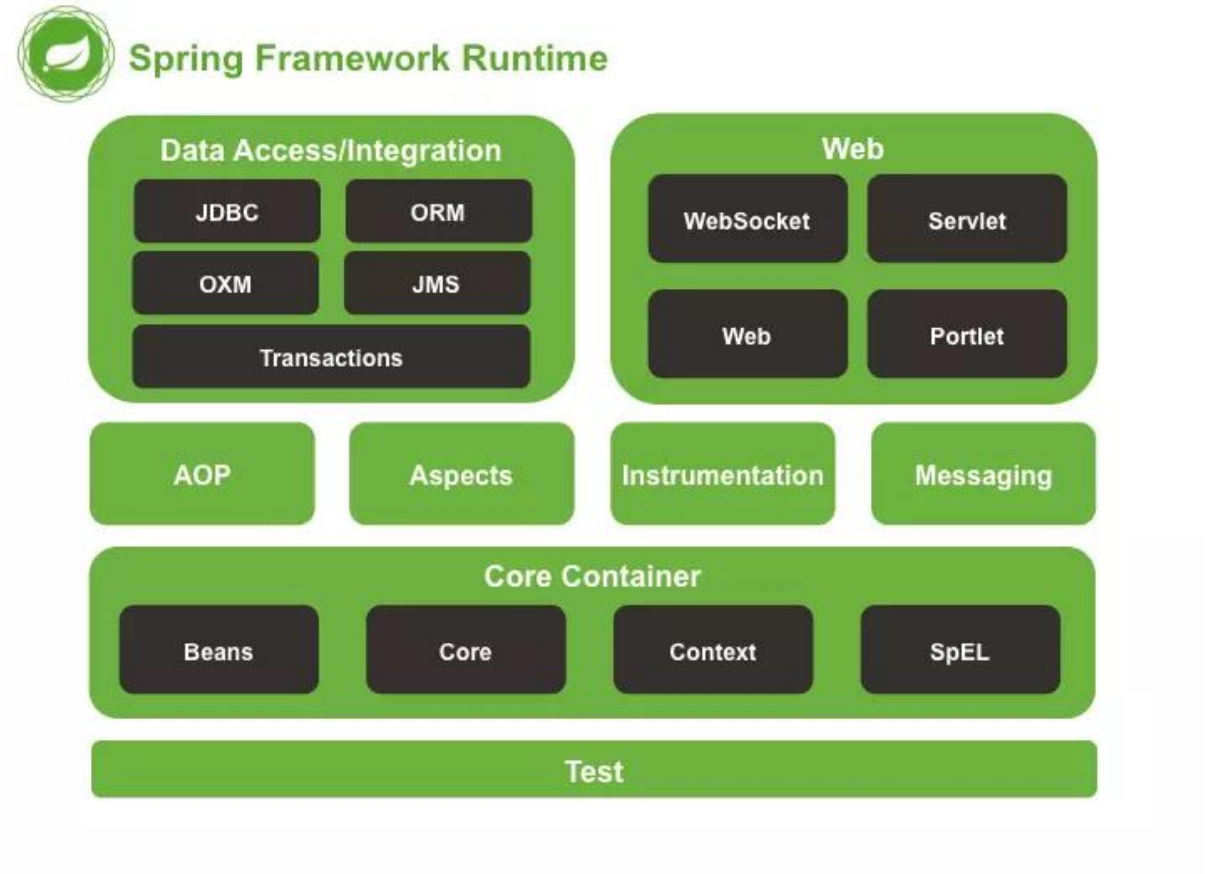
Giới thiệu về hệ sinh thái Spring

Spring là một Framework giúp phát triển các ứng dụng trên nền tảng ngôn ngữ Java. Hiện nay, Spring đang được sử dụng bởi hàng triệu lập trình viên và là Framework Java phổ biến nhất. Lý do khiến Spring được ưa chuộng đến vậy là do nó có thể giúp các lập trình viên tạo các ứng dụng có hiệu năng cao mà lại dễ kiểm thử. Spring Framework được xây dựng dựa trên hai nguyên tắc thiết kế chính đó là Dependency Injection và Aspect Oriented Programming.

Những tính năng cốt lõi của Spring có số lượng ứng dụng vô cùng phong phú ví dụ như Java Desktop, ứng dụng di động hay là Website như hệ thống này. Bằng cách dựa trên mô hình sử dụng POJO (Plain old Java Object), Spring đã thực hiện được mục đích chính của mình là việc giúp phát triển các ứng dụng J2EE một cách dễ dàng hơn.

Kiến trúc của Spring Framework

Spring Framework rất đồ sộ với nhiều module khác nhau. Các module này sẽ được lựa chọn dùng tuỳ theo mục đích phát triển ứng dụng mà người sử dụng mong muốn. Chính vì thế dù đồ sộ nhưng Spring vẫn có thể dễ dàng làm nên các hệ thống, chương trình phức tạp và trơn tru trong khi hoạt động.



Hình 1 Mô hình kiến trúc của Spring Framework

Test: là module giúp hỗ trợ kiểm thử với JUNIT hay TestNG

Spring Core Container: là tầng cốt lõi của Spring được tạo nên từ các module con là Spring Core, Beans, Context và Expression Language (EL). Các tính năng nền tảng như IOC và Dependency Injection có thể hoạt động được chính là nhờ tầng này.

AOP, Aspects, Instrumentation, Messaging: là những module hỗ trợ trong việc lập trình hướng khía cạnh (Aspect Oriented Programming). Đồng thời các module này cũng hỗ trợ tích hợp với AspectJ.

Data Access/Integration: là module có vai trò hỗ trợ kết nối và lấy dữ liệu từ Database. Nhóm này bao gồm JDBC, ORM, OXM, JMS và Transaction.

Web: còn được biết đến với cái tên Spring MVC (Model View Controller). Đây là nhóm module hỗ trợ trong việc lập trình Web.

Giới thiệu về Spring Boot

Trước đây ta có Spring MVC là một Framework tuyệt vời để phát triển website. Tuy nhiên do tồn tại một số nhược điểm như quá nhiều cấu hình,... , Spring Boot đã được xây dựng để khắc phục những nhược điểm đó.

Spring Boot là một dự án phát triển trên nền tảng ngôn ngữ Java và là một phần của hệ sinh thái Spring framework. Nó giúp cho các lập trình viên chúng ta đơn giản hóa quá trình lập trình một ứng dụng với Spring, chỉ tập trung phát triển nghiệp vụ cho ứng dụng…

Spring Boot cung cấp tính năng RAD (Rapid Application Development) - Phát triển ứng dụng nhanh, được dùng để tạo các ứng dụng độc lập dựa trên Spring và không yêu cầu cấu hình XML. Nhờ đó nó đã loại bỏ được những vấn đề cần phải cấu hình quá nhiều mà người tiền nhiệm Spring MVC đã gặp phải.

Spring Boot đã chiếm được cảm tình của đông đảo các lập trình viên Java nhờ các ưu điểm sau:

- Phát triển các hệ thống website một cách dễ dàng và tiết kiệm thời gian.
- Không có cấu hình XML và tự động cấu hình tất cả các components cho một ứng dụng Springs cấp sản xuất.
- Với các máy chủ nhúng được tạo sẵn như Tomcat, Jetty và Undertow, Spring Boot giúp việc triển khai hệ thống nhanh hơn và hiệu quả hơn.
- Điểm cuối HTTP, cho phép nhập các tính năng bên trong ứng dụng như chỉ số, tình trạng sức khoẻ, v.v.
- Nhiều lựa chọn bổ sung, hỗ trợ nhà phát triển làm việc với cơ sở dữ liệu được nhúng vào trong bộ nhớ. Dễ dàng truy cập cơ sở dữ liệu và các dịch vụ hàng đợi như MySQL, Oracle, MongoDB, Redis, ActiveMQ và các dịch vụ khác.
- Tích hợp trơn tru với hệ sinh thái Spring.
- Cộng đồng người sử dụng lớn với rất nhiều hướng dẫn, tạo điều kiện cho giai đoạn làm quen.

1.2. ReactJS

Giới thiệu về ReactJS

ReactJS là một opensource được phát triển bởi Facebook, ra mắt vào năm 2013, bản thân nó là một thư viện Javascript được dùng để để xây dựng các tương tác với các

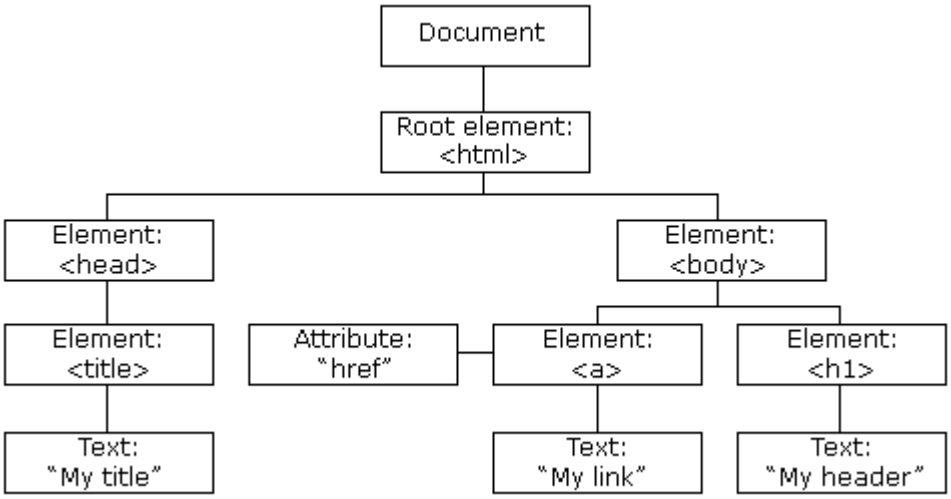
thành phần trên website. Một trong những điểm nổi bật nhất của ReactJS đó là việc render dữ liệu không chỉ thực hiện được trên tầng Server mà còn ở dưới Client nữa.

Trước khi có ReactJS, lập trình viên thường gặp rất nhiều khó khăn trong việc sử dụng "vanilla JavaScript" (JavaScript thuần) và JQuery để xây dựng UI. Điều đó đồng nghĩa với việc quá trình phát triển ứng dụng sẽ lâu hơn và xuất hiện nhiều bug, rủi ro hơn. Vì vậy vào năm 2011, Jordan Walke – một nhân viên của Facebook đã khởi tạo ReactJS với mục đích chính là cải thiện quá trình phát triển UI. Hơn nữa, để tăng tốc quá trình phát triển và giảm thiểu những rủi ro có thể xảy ra trong khi coding, React còn cung cấp cho chúng ta khả năng Reusable Code (tái sử dụng code) bằng cách đưa ra 2 khái niệm quan trọng bao gồm JSX và Virtual DOM.

JSX

Trọng tâm chính của bất kỳ website cơ bản nào đó là những HTML documents. Trình duyệt Web đọc những document này để hiển thị nội dung của website trên máy tính, tablet, điện thoại của bạn. Trong suốt quá trình đó, trình duyệt sẽ tạo ra một thứ gọi là Document Object Model (DOM) – một tree đại diện cho cấu trúc website được hiển thị như thế nào. Lập trình viên có thể thêm bất kỳ dynamic content nào vào những dự án của họ bằng cách sử dụng ngôn ngữ JavaScript để thay đổi cây DOM.

JSX (nói ngắn gọn là JavaScript extension) là một React extension giúp chúng ta dễ dàng thay đổi cây DOM bằng các HTML-style code đơn giản. Và kể từ lúc ReactJS browser hỗ trợ toàn bộ những trình duyệt Web hiện đại, bạn có thể tự tin sử dụng JSX trên bất kỳ trình duyệt nào mà bạn đang làm việc.

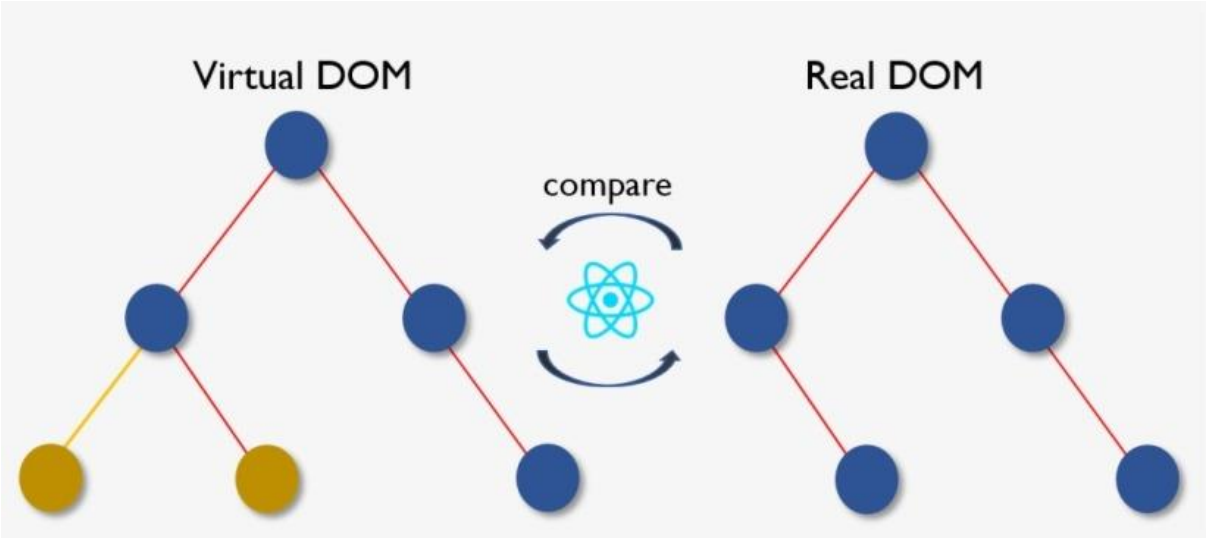


Hình 2 Cấu trúc cây DOM

Virtual DOM

Nếu bạn không sử dụng ReactJS (và JSX), website của bạn sẽ sử dụng HTML để cập nhật lại cây DOM cho chính bản nó (quá trình thay đổi diễn ra tự nhiên trên trang mà người dùng không cần phải tải lại trang), cách làm này sẽ ổn cho các website nhỏ, đơn giản, static website. Nhưng đối với các website lớn, đặc biệt là những website thiên về xử lý các tương tác của người dùng nhiều, điều này sẽ làm ảnh hưởng performance website cực kỳ nghiêm trọng bởi vì toàn bộ cây DOM phải reload lại mỗi lần người dùng nhấn vào tính năng yêu cầu phải tải lại trang).

Tuy nhiên, nếu bạn sử dụng JSX thì bạn sẽ giúp cây DOM cập nhật cho chính DOM đó, ReactJS đã khởi tạo một thứ gọi là Virtual DOM (DOM ảo). Virtual DOM (bản chất của nó theo đúng tên gọi) là bản copy của DOM thật trên trang đó, và ReactJS sử dụng bản copy đó để tìm kiếm đúng phần mà DOM thật cần cập nhật khi bất kỳ một sự kiện nào đó khiến thành phần trong nó thay đổi (chẳng hạn như user nhấn vào một nút bất kỳ).



Hình 3 DOM ảo và DOM thật

Với việc cập nhật đúng chỗ như vậy, khỏi phải nói nó tiết kiệm cho chúng ta rất nhiều tài nguyên cũng như thời gian xử lý. Ở các website lớn và phức tạp như thương mại điện tử, đặt món ăn, v.v bạn sẽ thấy việc này là vô cùng cần thiết và quan trọng để làm tăng trải trải nghiệm của khách hàng và performance được cải thiện đáng kể.

1.3. MongoDB

MongoDB là một hệ quản trị cơ sở dữ liệu mã nguồn mở, là CSDL thuộc NoSql và được hàng triệu người sử dụng.

MongoDB là một database hướng tài liệu (document), các dữ liệu được lưu trữ trong document kiểu JSON thay vì dạng bảng như CSDL quan hệ nên truy vấn sẽ rất nhanh. Với CSDL quan hệ chúng ta có khái niệm bảng, các cơ sở dữ liệu quan hệ (như MySQL hay SQL Server...) sử dụng các bảng để lưu dữ liệu thì với MongoDB chúng ta sẽ dùng khái niệm là collection thay vì bảng. So với RDBMS thì trong MongoDB collection ứng với table, còn document sẽ ứng với row, MongoDB sẽ dùng các document thay cho row trong RDBMS. Các collection trong MongoDB được cấu trúc rất linh hoạt, cho phép các dữ liệu lưu trữ không cần tuân theo một cấu trúc nhất định. Thông tin liên quan được lưu trữ cùng nhau để truy cập truy vấn nhanh thông qua ngôn ngữ truy vấn MongoDB.

MongoDB mang đến cho người sử dụng một số ưu điểm:

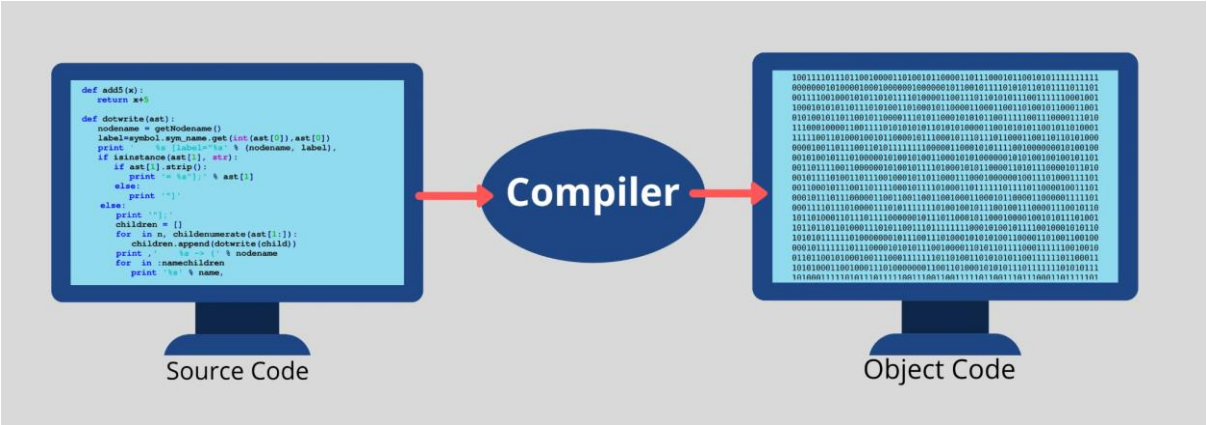
- Không schema: Giống như các cơ sở dữ liệu NoSQL khác, MongoDB không yêu cầu các schema được xác định trước.
- MongoDB lưu trữ bất kỳ loại dữ liệu nào: Điều này cho phép người dùng linh hoạt tạo số lượng trường trong document theo nhu cầu, và giúp việc mở rộng cơ sở dữ liệu MongoDB trở nên dễ dàng hơn so với cơ sở dữ liệu quan hệ truyền thống.
- Hướng document: Một trong những ưu điểm của việc sử dụng document là các đối tượng này ánh xạ tới các kiểu dữ liệu gốc trong một số ngôn ngữ lập trình. Việc có các document được nhúng cũng làm giảm nhu cầu kết nối cơ sở dữ liệu, điều này có thể làm giảm chi phí.
- Khả năng mở rộng: Kiến trúc mở rộng theo chiều ngang của MongoDB giúp bạn tạo ra một ứng dụng có thể xử lý được lưu lượng truy cập tăng đột biến khi doanh nghiệp của bạn phát triển. Ngoài ra, việc phân chia dữ liệu (sharding) cho phép cơ sở dữ liệu phân phối dữ liệu trên một cụm máy. MongoDB cũng hỗ trợ tạo vùng dữ liệu dựa trên shard key.
- Hỗ trợ bên thứ ba: MongoDB hỗ trợ một số công cụ lưu trữ và cung cấp API công cụ lưu trữ có thể cắm được (pluggable storage engine API) cho phép các bên thứ ba phát triển công cụ lưu trữ dữ liệu riêng.
- Linh hoạt lưu trữ tệp dung lượng lớn: MongoDB phát triển hệ thống tệp riêng GridFS, gần giống với hệ thống tệp phân tán Hadoop. Việc sử dụng hệ thống tệp nhằm để lưu trữ các tệp vượt qua kích thước giới hạn của BSON (16 MB cho mỗi document).

1.4. Trình biên dịch

1.4.1. Trình biên dịch là gì?

Một trình biên dịch (compiler) là một phần mềm lấy mã của các ngôn ngữ lập trình bậc cao (high level language) như Java, C++ làm đầu vào và chuyển đổi đầu vào đó thành mã của một ngôn ngữ lập trình bậc thấp. Nó liệt kê tất cả các lỗi nếu như mã đầu vào không tuân theo luật của ngôn ngữ đó, hay còn gọi là lỗi biên dịch. Quá trình này nhanh hơn các phần mềm thông dịch (interpreter) nhưng lại khó khăn hơn trong quá trình tìm đồng loạt các lỗi trong chương trình.

Cụ thể hơn trình biên dịch sẽ dịch các chỉ dẫn của ngôn ngữ lập trình bậc cao thành các ngôn ngữ bậc thấp hơn, thường là ngôn ngữ máy (machine level language). Chương trình là đầu vào cho compiler gọi là mã nguồn (source code). Chương trình này sẽ được chuyển đổi thành mã máy hay còn gọi là mã đối tượng (Object code). Các mã này chỉ chứa 2 số 0 và 1 nhị phân, do đó máy tính có thể trực tiếp xử lý và thực thi (execute).



Hình 4 Ngôn ngữ lập trình cấp cao được trình biên soạn dịch ra mã đối tượng

1.4.2. Lịch sử ra đời

Những bước đầu thiết kế trình biên dịch

Sự phát triển trình biên dịch đầu tiên có liên quan chặt chẽ đến sự ra đời của máy tính. Trong những năm 1940 và 1950, máy tính vẫn còn ở thời kỳ đầu, với giá cao và giới hạn máy đồng nghĩa với việc chỉ có một số công ty lớn và các cơ quan chính phủ có thể chi trả để sử dụng chúng. Tuy nhiên, tiềm năng của máy tính đã được ghi nhận sớm, và các nhà khoa học máy tính bắt đầu nghiên cứu cách làm cho chúng trở nên dễ tiếp cận và sử dụng hơn. Một trong những đổi mới chính xuất hiện trong thời kỳ này là sự phát triển của các ngôn ngữ lập trình cấp cao, như Fortran và COBOL, làm cho người dùng có thể viết chương trình bằng một ngôn ngữ gần với ngôn ngữ con người, thay vì mã máy.

Trình biên dịch đầu tiên được phát triển vào những năm 1950, trong thời kỳ đầu của ngành công nghiệp máy tính. Những chiếc máy tính đầu tiên là các máy tính mainframe, và chi phí cực kỳ đắt đỏ cùng với sự giới hạn về số lượng đã khiến chỉ có một số ít tập đoàn lớn và các cơ quan chính phủ có thể chi trả để sử dụng chúng. Để làm cho những máy tính này trở nên dễ tiếp cận hơn đối với công dân, các nhà khoa học máy

tính đã phát triển trình biên dịch cho phép người dùng viết chương trình bằng các ngôn ngữ cấp cao hơn. Trình biên dịch đầu tiên được phát triển vào những năm 1950, và nó được thiết kế để dịch các chương trình viết bằng ngôn ngữ Fortran thành mã máy. Việc phát triển trình biên dịch Fortran là một bước tiến lớn trong lịch sử của ngành công nghiệp máy tính, vì nó cho phép người dùng viết chương trình bằng một ngôn ngữ cấp cao, thay vì sử dụng mã máy, làm cho quá trình viết và duy trì các chương trình phức tạp trở nên dễ dàng hơn. Trình biên dịch Fortran là một trình biên dịch hướng batch (batch-oriented), có nghĩa là người dùng phải gửi chương trình của họ ở chế độ batch (batch mode), và trình biên dịch sẽ tạo mã máy cho mỗi chương trình trong batch. Quá trình này tốn thời gian và thường đòi hỏi người dùng phải đợi cho đến khi trình biên dịch hoàn thành quá trình tạo mã máy trước khi họ có thể tiếp tục công việc của mình.

Những tiến bộ trong thiết kế trình biên dịch

Trong những thập kỷ 1960 và 1970, khi máy tính trở nên phổ biến và giá cả trở nên dễ tiếp cận hơn, nhu cầu về trình biên dịch cũng gia tăng. Điều này dẫn đến sự phát triển của các trình biên dịch mới, phức tạp và hiệu quả hơn. Một trong những tiến triển quan trọng trong thiết kế trình biên dịch trong thời kỳ này là sự phát triển của trình biên dịch tối ưu hóa, được thiết kế để tạo ra mã máy nhanh hơn và hiệu quả hơn.

Một phần quan trọng trong sự tiến bộ này là sự phát triển của trình biên dịch tối ưu hóa, có khả năng phân tích chương trình và thực hiện các tối ưu hóa như tối ưu vòng lặp, loại bỏ mã chết (dead code), và cấp phát bộ nhớ cho thanh ghi. Những sự tối ưu hóa này có thể cải thiện đáng kể hiệu suất của mã máy được tạo ra. Trình biên dịch tối ưu hóa là một bước tiến lớn trong thiết kế trình biên dịch, vì nó cho phép những lập trình viên viết các chương trình phức tạp và yêu cầu hiệu suất cao. Với sự giúp đỡ của trình biên dịch tối ưu hóa, những kỹ sư có thể viết các chương trình có thể tận dụng hết sức mạnh của máy tính và họ tự tin rằng mã của họ sẽ được thực thi một cách hiệu quả.

Sự trỗi dậy của lập trình hướng đối tượng (OOP)

Những năm 1980 và 1990, lập trình hướng đối tượng (OOP) trở thành một mô hình lập trình phổ biến. Mô hình lập trình mới này mang lại những thay đổi đáng kể trong thiết kế trình biên dịch, vì trình biên dịch phải được điều chỉnh để xử lý cú pháp và ngữ nghĩa mới của ngôn ngữ OOP, chẳng hạn như C++ và Java. Trình biên dịch cho

các ngôn ngữ OOP thường phải đối mặt với các cấu trúc phức tạp như thừa kế lớp, đa hình và các khái niệm OOP khác, làm cho quá trình thiết kế trở nên khó khăn hơn. Tuy nhiên, việc phát triển những trình biên dịch này đã giúp các kỹ sư viết các chương trình phức tạp hơn và tạo ra mã có thể tái sử dụng, cải thiện năng suất và làm cho việc bảo trì các hệ thống phần mềm lớn trở nên dễ dàng hơn.

Ý tưởng về lập trình hướng đối tượng được giới thiệu lần đầu tiên bởi các nhà khoa học máy tính Alan Kay và Adele Goldberg vào cuối những năm 1960. Vào thời điểm đó, hầu hết các chương trình máy tính được viết bằng lập trình thủ tục (procedural programming), dựa trên ý tưởng viết mã từng bước một (step-by-step). Kay và Goldberg tin rằng có một cách tốt hơn để viết phần mềm, và họ phát triển ý tưởng về lập trình hướng đối tượng. Trong lập trình hướng đối tượng, chương trình được tạo thành từ các đối tượng, là những đơn vị đóng gói dữ liệu (data) và hành vi (behavior). Tiếp cận này giúp tạo ra mã mô-đun và có thể tái sử dụng, làm cho việc viết chương trình trở nên linh hoạt và hiệu quả hơn.

Ngôn ngữ lập trình hướng đối tượng đầu tiên là Simula, được phát triển vào giữa những năm 1960. Tuy nhiên, cho đến những năm 1980, lập trình hướng đối tượng mới bắt đầu nhận được sự ghi nhận rộng rãi, khi các ngôn ngữ hướng đối tượng như Smalltalk và C++ được giới thiệu. Sự ghi nhận của lập trình hướng đối tượng được thúc đẩy bởi nhiều yếu tố, bao gồm sự phức tạp các hệ thống phần mềm ngày càng gia tăng, nhu cầu về mã có thể tái sử dụng (reusable code) và sự mong muốn có ngôn ngữ lập trình thân thiện và gần gũi với con người. Lập trình hướng đối tượng cũng mang lại khả năng cho kỹ sư viết mã gần với các đối tượng thực tế mà chương trình của họ đại diện, làm tăng tính trực quan và dễ hiểu của mã nguồn.

Trong quá trình này, trình biên dịch đã phải điều chỉnh để xử lý cú pháp và ngữ nghĩa mới của ngôn ngữ lập trình hướng đối tượng. Điều này thách thức các nhà phát triển trình biên dịch phải thích ứng với cấu trúc phức tạp của lập trình hướng đối tượng, bao gồm cả việc xử lý quan hệ thừa kế, đa hình, và các khái niệm khác. Tuy nhiên, sự phát triển của trình biên dịch này đã giúp mở ra khả năng viết và bảo trì mã nguồn lập trình hướng đối tượng một cách dễ dàng hơn, đồng thời thúc đẩy sự hiểu biết và sử dụng rộng rãi của lập trình hướng đối tượng trong ngành công nghiệp phần mềm.

Kỷ nguyên của trình biên dịch hiện đại

Trong vài thập kỷ qua, đã có những tiến triển đáng kể trong công nghệ trình biên dịch, bao gồm sự phát triển của các kỹ thuật tối ưu hóa mới, cải thiện xử lý lỗi, và tích hợp các ngôn ngữ lập trình mới. Một trong những tiến bộ quan trọng nhất trong công nghệ trình biên dịch là sự phát triển của trình biên dịch just-in-time (JIT). Trình biên dịch JIT là một loại trình biên dịch biên dịch mã nguồn ngay lập tức, trong thời gian chạy, thay vì trước thời điểm thực thi. Trình biên dịch JIT được sử dụng trong nhiều ngôn ngữ lập trình hiện đại, như Java và Python, và cho phép thực thi chương trình nhanh hơn. Điều này tạo điều kiện để tối ưu mã nguồn cho môi trường phần cứng và phần mềm cụ thể, dẫn đến việc hiệu suất được cải thiện. Một tiến triển quan trọng khác trong công nghệ trình biên dịch là sự phát triển của trình biên dịch ahead-of-time (AOT), biên dịch mã nguồn trước thời điểm thực thi. Trình biên dịch AOT được sử dụng cho các hệ thống nhúng và thiết bị di động, nơi các ràng buộc về tài nguyên hạn chế khiến cho việc sử dụng trình biên dịch JIT không khả thi.

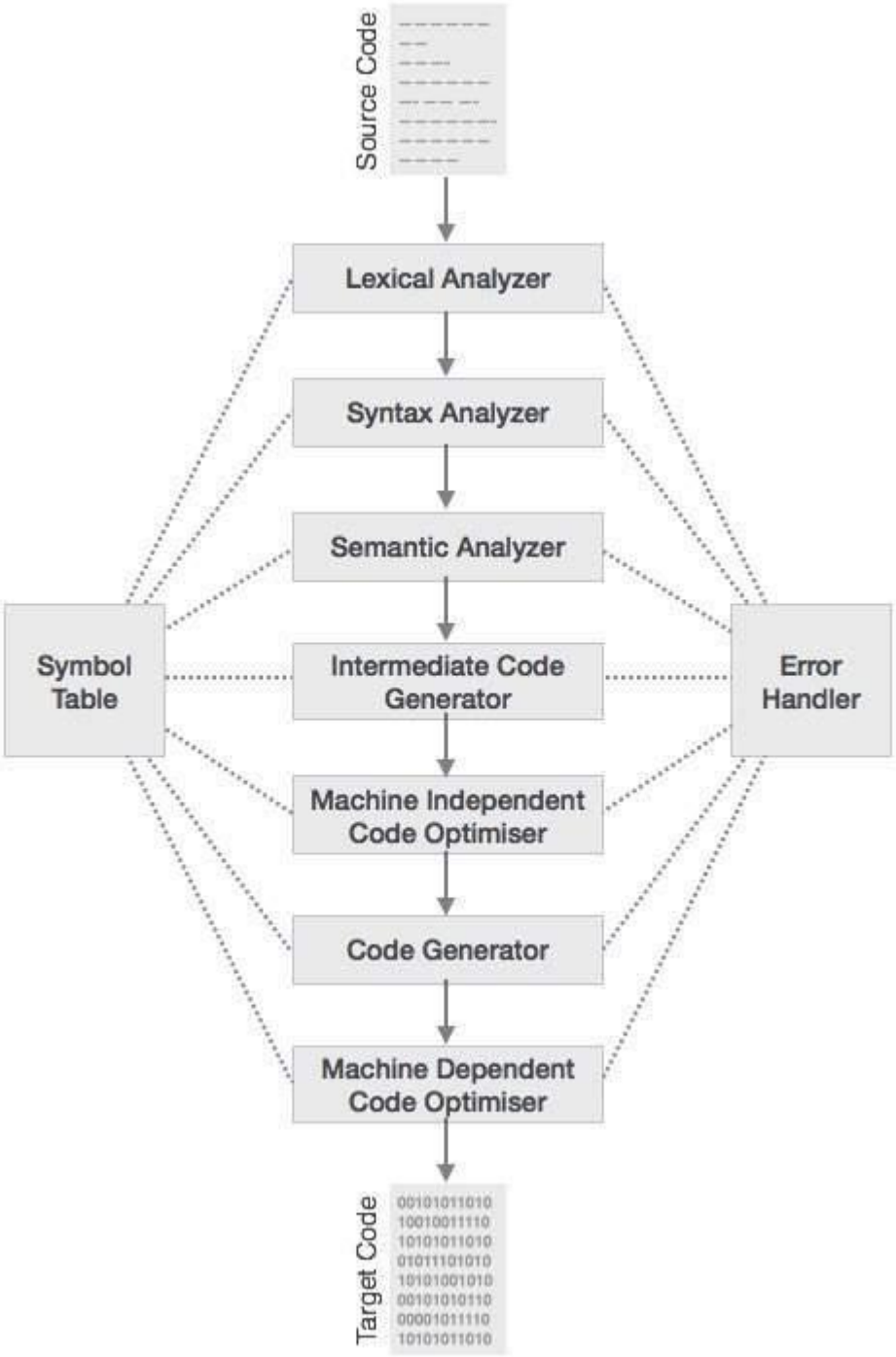
Tóm lại, lịch sử của trình biên dịch đã chứng kiến nhiều phát triển và tiến bộ quan trọng trong quá trình tiến hóa của ngành công nghiệp máy tính. Từ những bước đầu tiên của Grace Hopper với trình biên dịch Fortran, qua sự xuất hiện của trình biên dịch tối ưu hóa và sự nổi lên của lập trình hướng đối tượng, trình biên dịch đã đóng một vai trò quan trọng trong việc tạo ra mã máy hiệu quả và dễ bảo trì. Đồng thời, sự phát triển của trình biên dịch cũng đã thúc đẩy sự tiến bộ trong thiết kế ngôn ngữ lập trình và mô hình lập trình, làm nền tảng cho sự phát triển của các ứng dụng và hệ thống phần mềm phức tạp trong thời kỳ ngày nay.

1.4.3. Trình biên dịch hoạt động như thế nào?

Quá trình biên dịch mã nguồn bao gồm nhiều bước. Mã nguồn sẽ đi qua các bước này theo thứ tự, và nếu có bất kỳ lỗi nào trong mã nguồn, nó sẽ được kiểm tra qua các bước này và quá trình biên dịch sẽ dừng lại giữa chừng, hiển thị lỗi biên dịch. Ngược lại, nếu mọi thứ đều hoạt động bình thường, trình biên dịch sẽ không hiển thị lỗi nào và tiến hành biên dịch mã nguồn. Dưới đây là các bước trong quá trình biên dịch:

- Phân tích từ vựng (Lexical Analysis): Mã nguồn được kiểm tra bởi bộ phân tích từ vựng của trình biên dịch, nó phân chia mã nguồn thành các token như từ khóa, tên biến, toán tử và dấu câu.

- Phân tích cú pháp (Syntax Analysis): Bước tiếp theo là phân tích cú pháp, trong đó bộ phân tích cú pháp kiểm tra mã nguồn để xác định có lỗi cú pháp hay không và đảm bảo nó tuân theo quy tắc của ngôn ngữ lập trình. Trình biên dịch tạo ra một cây cú pháp trừu tượng (AST) biểu diễn cấu trúc của mã nguồn.
- Phân tích ngữ nghĩa (Semantic Analysis): Khi mã nguồn đã đúng cú pháp, trình biên dịch thực hiện phân tích ngữ nghĩa trên mã đã được phân tích cú pháp để tìm hiểu ý nghĩa. Trình biên dịch kiểm tra lỗi logic như không phù hợp kiểu dữ liệu, biến chưa được khai báo và cách sử dụng toán tử không chính xác.
- Tối ưu hoá (Optimization): Trình biên dịch có thể thực hiện nhiều tối ưu hóa để cải thiện hiệu suất của mã máy được tạo ra. Đây là một pha tùy chọn của trình biên dịch. Nó loại bỏ mã chết và sắp xếp thứ tự các lệnh để tăng hiệu suất của việc thực thi chương trình.
- Tạo mã (Code Generation): Bước cuối cùng là tạo mã, nơi trình biên dịch dịch cây cú pháp thành mã máy có thể đọc được. Trình tạo mã tạo ra mã ngôn ngữ hợp ngữ, sau đó được dịch thành mã nhị phân (binary code) có thể thực thi bởi máy tính.



Hình 5 Quá trình biên dịch từ mã nguồn thành mã máy

1.4.4. Phân loại trình biên dịch

- Trình biên dịch chéo (Cross compiler): Chương trình biên dịch được tạo ra để chạy trên một máy tính với CPU hoặc Hệ điều hành khác biệt so với máy tính mà trình biên dịch chạy trên đó. Một trình biên dịch chéo có thể tạo ra mã máy cho một hệ điều hành nhúng chạy trên vi điều khiển, trong khi trình biên dịch chạy trên máy tính cá nhân.

- Trình biên dịch khởi động (Bootstrap compiler): Là trình biên dịch được viết bằng chính ngôn ngữ mà nó biên dịch. Trong quá trình phát triển, một trình biên dịch đầu tiên thường được sử dụng để tự biên dịch lại chính nó để tạo ra phiên bản mới. Ví dụ, một trình biên dịch khởi động C phiên bản 11 được sử dụng để biên dịch chính nó để tạo ra một phiên bản mới của trình biên dịch C phiên bản 12.
- Trình giải mã (Decompiler): Là trình biên dịch dịch ngược từ một ngôn ngữ cấp thấp về một ngôn ngữ cấp cao hơn. Thường được sử dụng để phân tích và đọc mã máy đã được biên dịch trước đó. Ví dụ, một trình giải mã có thể chuyển đổi mã máy thành mã nguồn C.
- Trình dịch ngược (Transcompiler): Là trình biên dịch chuyển đổi mã nguồn từ một ngôn ngữ lập trình cao cấp sang một ngôn ngữ khác mà thường là tương đương với ngôn ngữ nguồn. Ví dụ, một trình dịch ngược có thể chuyển đổi mã nguồn từ Java sang C++.

Một trình biên dịch chỉ có thể dịch những chương trình nguồn được viết bằng ngôn ngữ mà trình biên dịch đó được thiết kế. Mỗi ngôn ngữ lập trình cấp cao đều yêu cầu một trình biên dịch riêng để thực hiện quá trình biên dịch.

- Ngôn ngữ C, C++:
    - Gcc (GNU compiler collection): Là trình biên dịch mạnh mẽ và phổ biến cho nhiều ngôn ngữ, bao gồm C. Gcc hỗ trợ nhiều kiến trúc và nền tảng khác nhau.
    - Clang: Một trình biên dịch C/C++ do dự án LLVM phát triển. Clang có khả năng phát hiện lỗi và cung cấp các thông báo lỗi chi tiết.
- Ngôn ngữ C++:
    - Gcc (GNU compiler collection): Ngoài việc hỗ trợ C, Gcc cũng là một trình biên dịch C++. Sử dụng lệnh g++ để biên dịch mã nguồn C++.
    - Clang: Tương tự, Clang cũng là một trình biên dịch cho C++ và được sử dụng rộng rãi trong cộng đồng.
- Ngôn ngữ Java:

  - Javac: Là trình biên dịch chính thức của Java Development Kit (JDK). Javac biên dịch mã nguồn Java thành mã bytecode chạy trên Java Virtual Machine (JVM).
  - Eclipse Compiler for Java (ECJ): Được sử dụng trong môi trường phát triển Eclipse. ECJ có thể được sử dụng như một trình biên dịch độc lập.
- Ngôn ngữ Python:
  - CPython: Là trình biên dịch mặc định cho Python. Cpython biên dịch mã nguồn Python thành bytecode và chạy trên máy ảo Python (PVM - Python Virtual Machine).
  - PyPy: Là một trình biên dịch JIT (Just-In-Time) cho Python. Pypy có thể cung cấp hiệu suất cao hơn so với CPython trong một số trường hợp.
  - Jython: Là một trình biên dịch giúp biên dịch mã nguồn Python thành bytecode chạy trên JVM. Do đó, Java có thể cung cấp thư viện giúp chạy mã nguồn Python trong môi trường JVM.

1.4.5. Môi trường thực thi của các trình biên dịch

Môi trường thực thi (runtime environment) là nơi mà các chương trình nói chung và các trình biên dịch nói riêng có thể chạy và thực hiện công việc của mình. Đây là không gian trong đó mã nguồn được biên dịch hoặc thông dịch để chuyển đổi thành mã máy và sau đó thực thi để thực hiện các nhiệm vụ cụ thể. Đây là một số ví dụ về môi trường thực thi:

- Máy ảo (Virtual Machine): Máy ảo như Java Virtual Machine (JVM) cung cấp môi trường thực thi cho các chương trình được viết bằng Java. Mã nguồn được biên dịch thành mã bytecode là mã trung gian. Mã này sau đó được đi qua trình biên dịch JIT có trong JVM để biến đổi thành mã máy, cuối cùng được thực thi bởi CPU.
- Hệ điều hành: Mỗi hệ điều hành như Windows, Linux, hoặc macOS đều cung cấp một môi trường thực thi cho các trình biên dịch. Các trình biên dịch thực thi trên hệ điều hành thông qua những câu lệnh riêng của chúng.
- Nền tảng Docker container: Nền tảng container như Docker cung cấp một môi trường thực thi cô lập (Isolated runtime environment) để chạy ứng dụng trong

các container. Các container chứa tất cả những gì cần thiết để chạy một ứng dụng cụ thể, bao gồm cả hệ điều hành và các tài nguyên cần thiết.

Tóm lại, các trình biên dịch là công cụ quan trọng không thể thiếu trong quá trình phát triển phần mềm nói chung và trong hệ thống học tập lập trình nói riêng. Chúng cho phép người sử dụng hệ thống viết mã trong các ngôn ngữ lập trình cấp cao, đảm bảo tính đúng đắn và hiệu quả của mã nguồn, và hỗ trợ việc phát triển phần mềm dành cho nhiều nền tảng và kiến trúc khác nhau. Hiểu biết về trình biên dịch và môi trường thực thi của chúng là cực kì quan trọng để xây dựng một hệ thống học tập lập trình hiệu quả, hiệu suất cao, đáng tin cậy và có khả năng mở rộng.

CHƯƠNG II: CÁC VẤN ĐỀ VÀ PHƯƠNG PHÁP TIẾP CẬN

2.1. Rủi ro khi hệ thống biên dịch và thực thi

Khi xây dựng một hệ thống học tập lập trình, ngoài các chức năng giao diện bên ngoài, chúng ta không thể bỏ qua một yếu tố cực kỳ quan trọng đóng vai trò thiết yếu chính là hệ thống biên dịch. Server trong hệ thống này có nhiệm vụ nhận đầu vào là code của người dùng, sau đó biên dịch và thực thi trả lại kết quả chạy cho người dùng. Đó là bài toán khá khó khăn khi server phải đảm bảo các đoạn code của người dùng được biên dịch mà không gây ra rủi ro cho hệ thống của chúng ta. Việc chúng ta cho phép người dùng chạy code trên chính server của mình có thể rất nguy hiểm nếu như các đoạn code của người dùng là các đoạn mã độc (malicious code). Dù cho vô ý hay cố tình, các đoạn code này có khả năng đánh sập cả một hệ thống.

2.1.1. Rủi ro từ code người dùng

Dưới đây là các kịch bản mà người dùng có thể gây tổn hại đến hệ thống:

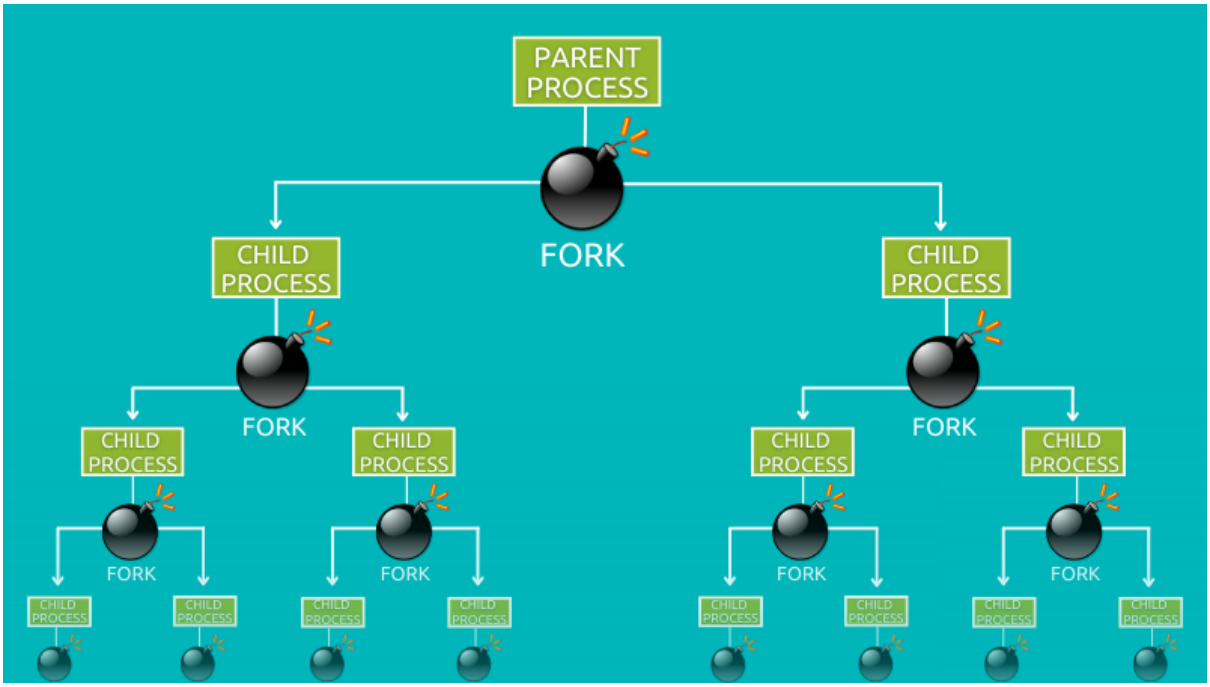
- Fork Bomb: :(){:|:&};:
- Vòng lặp vô tận (infinite loop)
- Đệ quy nhưng không có điều kiện cơ bản (base condition) để kết thúc
- Chạy một đoạn code tốn rất nhiều tài nguyên CPU và bộ nhớ RAM
- Tạo các file, xóa các file, tắt tiến trình chạy (kill process)...

Các chương trình vô tận như vòng lặp vô tận, đệ quy không có điều kiện cơ bản có thể gây ra sự đói tài nguyên cho khác tiến trình khác (process starvation). Để giải quyết vấn đề này cần đảm bảo rằng chương trình chạy trong một khoảng thời gian giới hạn, và nếu quá thời hạn tiến trình đó phải được tắt bằng cách sử dụng lệnh timeout trong Linux.

Nếu một người dùng chạy một chương trình chiếm nhiều tài nguyên CPU và bộ nhớ RAM có thể dẫn tới sự đói (starvation) cho các tiến trình khác. Điều này có thể sẽ không công bằng với những người dùng khác vì chương trình của họ sẽ bị thực thi chậm hơn. Do đó, cần đảm bảo rằng tất cả các tiến trình thực thi được cấp phát một lượng tài nguyên CPU và bộ nhớ RAM bằng nhau.

Về việc ngăn chặn người dùng tiêm những mã độc nhằm sửa đổi file hệ thống, chúng ta có thể sử dụng công nghệ chroot jail để cô lập một tiến trình và các tiến trình con của nó ra khỏi phần còn lại của hệ thống. Công nghệ này sẽ được làm rõ hơn ở phần dưới.

" :(){ :|: & };: " câu lệnh này nhìn tưởng chừng là một câu lệnh lỗi vô hại nhưng nó có thể làm đóng băng hệ thống trong vòng một giây. Đây chính là fork bomb, một loại tấn công từ chối dịch vụ (DoS), nó sẽ liên tục sinh ra những tiến trình giống hệt nhau cho đến khi hệ thống chạm ngưỡng 100% tài nguyên, khi đó hệ thống sẽ bị chậm đi rất nhiều, gần như không thể sử dụng được và có khả năng bị crash. Điều này có vẻ đáng sợ nhưng xử lý thì khá dễ, chúng ta chỉ cần giới hạn số tiến trình một người dùng có thể chạy.



Hình 6 Fork bomb liên tục nhân ra các tiến trình giống nhau

2.1.2. Rủi ro từ công nghệ cũ

Hệ quản trị học tập Moodle và VPL

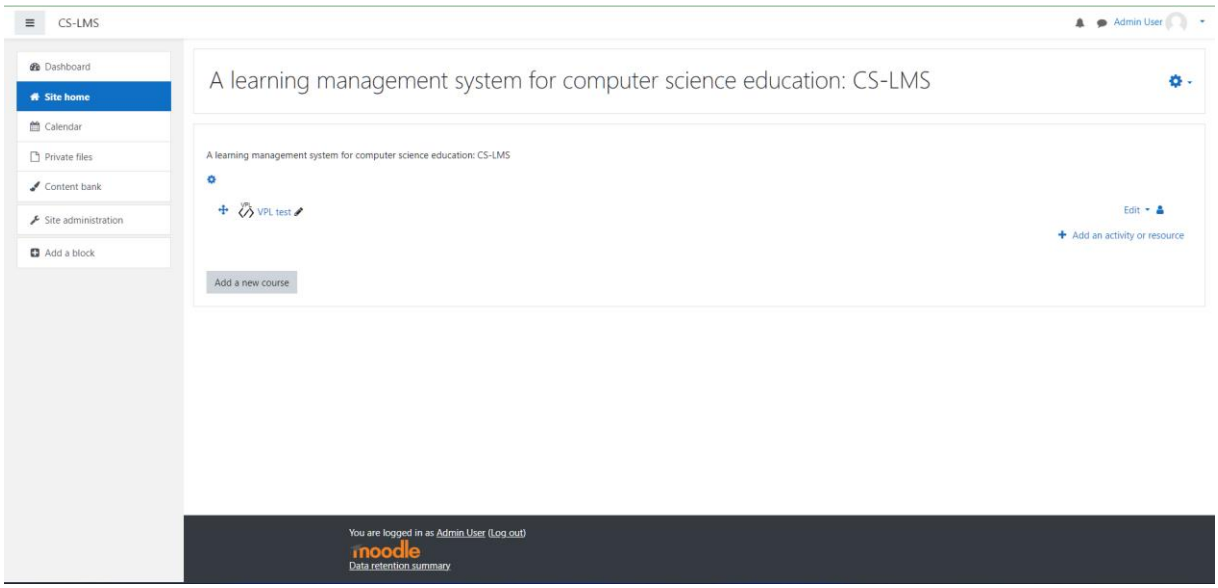
Moodle

Moodle là một hệ thống quản lý học tập (Learning Management System - LMS hoặc người ta còn gọi là Course Management System hoặc VLE - Virtual Learning Environment) mã nguồn mở (do đó miễn phí và có thể chỉnh sửa được mã nguồn), cho phép tạo các khóa học trên mạng Internet hay các website học tập trực tuyến. Moodle

nổi bật là thiết kế hướng tới giáo dục, dành cho những người làm trong lĩnh vực giáo dục.

Các doanh nghiệp đều đánh giá cao hệ thống đào tạo trực tuyến này, bởi những lợi ích mà Moodle mang lại:

- Mã nguồn mở và miễn phí: Bản thân Moodle là một phần mềm mã nguồn mở miễn phí được Giấy phép Public GNU phân phối. Điều này được hiểu đơn giản là người dùng hoặc các tổ chức có quyền tự do triển khai, nghiên cứu, sửa đổi và chia sẻ các phần mềm đào tạo trực tuyến để mọi nhu cầu riêng đều được đáp ứng.
- Có thể cấu hình và nhiều tính năng: Các tổ chức, doanh nghiệp đều ưa chuộng và đánh giá hệ thống đào tạo trực tuyến Moodle rất cao bởi khả năng cấu hình linh hoạt và có chứa nhiều tính năng. Moodle còn cho phép bạn xây dựng và cấu hình Moodle với hàng trăm plugin Moodle để nó hoạt động theo đúng những gì mà bạn mong muốn, trong đó một trong những plugin rất tốt cho việc học tập lập trình online là VPL.



Hình 7 Giao diện của hệ quản trị học tập Moodle

VPL

Giới thiệu về VPL

VPL (Virtual Programming Lab) là một plugin được ra mắt vào 2014, giúp quản lý các bài tập liên quan tới lập trình trong hệ thống quản lý học tập Moodle.

Các tính năng nổi bật của VPL có thể kể đến như:

- Có thể chỉnh sửa source code của các chương trình ngay trên trình duyệt
- Có thể chạy các bộ test để kiểm tra chương trình
- Có thể kiểm tra tính giống nhau giữa các file
- Có thể giới hạn quyền chỉnh sửa và ngăn chặn copy paste từ bên ngoài

Đặc biệt, VPL cung cấp nhiều tính năng lớn được tinh chỉnh cho bài tập lập trình:
Kiểm soát submit code:

- Cho phép giới hạn thời gian nộp
- Cho phép số lượng file tải lên tối đa
- Cho phép xác định kích thước tối đa của mỗi file được tải lên
- Cho phép đặt tên cho các file cần tải lên
- Xác thực mật khẩu khi duyệt, chỉnh sửa và tải lên các file riêng tư

Quản lý tệp:

- Cho phép chỉnh sửa các file từ trình duyệt bằng cách sử dụng trình soạn thảo code được tích hợp của VPL
- Làm nổi bật cú pháp cho các file được tải lên

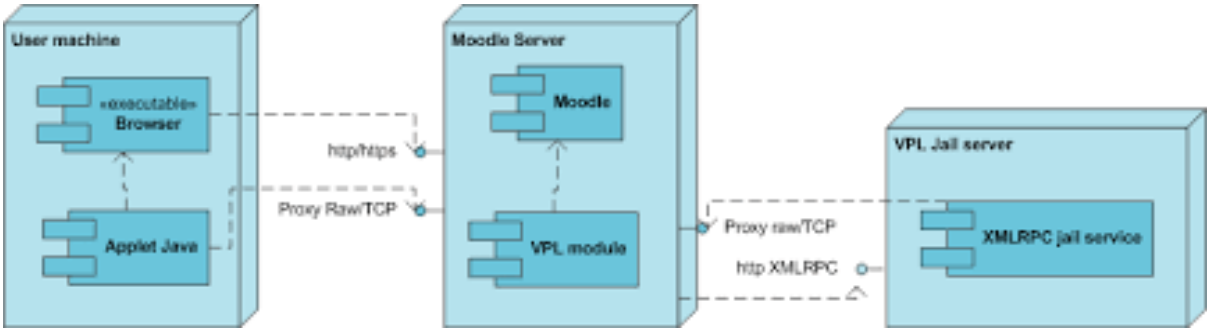
Thực thi và đánh giá submission

- Các kịch bản (script) run và debug hỗ trợ cho nhiều ngôn ngữ lập trình hiện nay (Ada, C, C++, C#, Fortran, Haskell, Java, Matlab/Octave, Pascal, Perl, PHP, Prolog, Python, Ruby, Scheme, SQL và VHDL)
- Các chương trình có thể được thực thi với đầu ra/đầu vào (input/output) trong một bảng điều khiển (console) của trình soạn thảo.
- Có thể giới hạn tài nguyên thực thi chương trình như giới hạn về thời gian, bộ nhớ, kích thước file và số tiến trình thực thi.

Kiến trúc của VPL

Các thành phần chính

VPL bao gồm ba yếu tố chính: một module của Moodle, một trình soạn thảo mã nguồn dựa trên trình duyệt (browser code editor) và một máy chủ jail (jail server) (Hình 2).



Hình 8 Kiến trúc của VPL

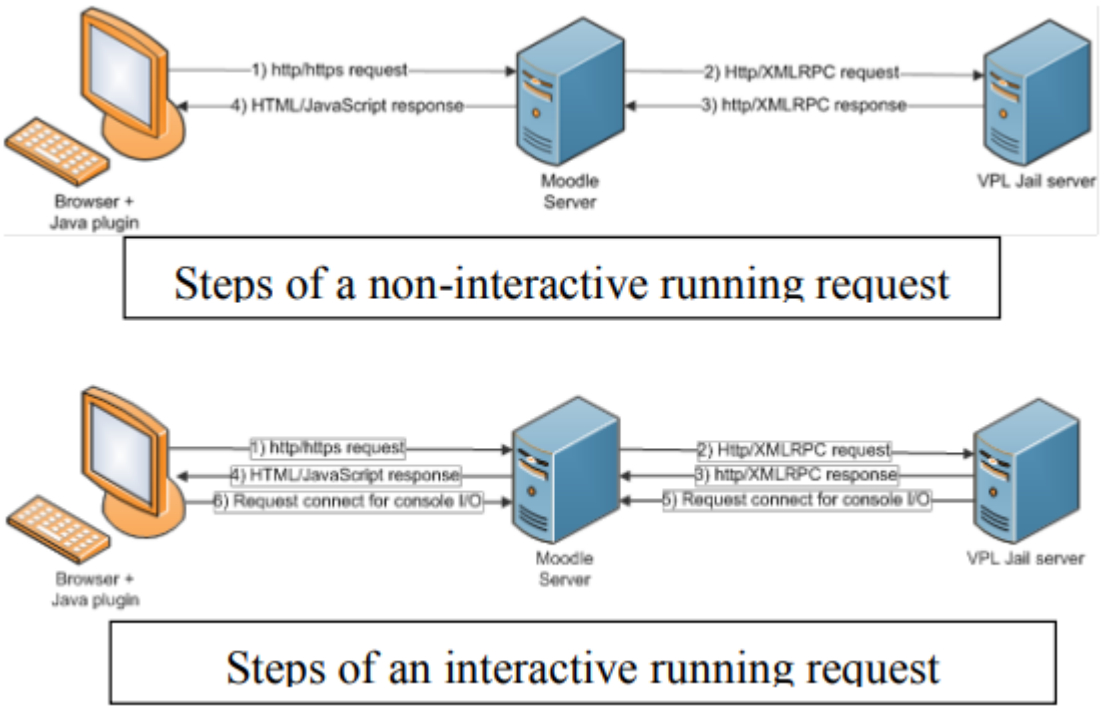
Trình soạn thảo mã nguồn là một ứng dụng Java Applet cung cấp các tính năng cơ bản để chỉnh sửa, chạy, gỡ lỗi và đánh giá mã chương trình trong một môi trường phát triển mã nguồn rất đơn giản.

Module của Moodle cung cấp các tính năng điển hình của loại thành phần này (sao lưu và khôi phục, tích hợp với sổ điểm, đặt lại khóa học, kiểm soát sự kiện, quyền truy cập dựa trên vai trò,...), nhưng cũng có các tính năng cụ thể như quản lý nộp bài, hỗ trợ đánh giá và tính năng chống đạo văn.

Máy chủ jail là máy chủ chịu trách nhiệm biên dịch và thực thi mã nguồn mà sinh viên submit trong một môi trường bảo mật và tách biệt. Nó chạy một lệnh linux chroot để cung cấp một phiên bản hạn chế của hệ thống file máy chủ với hạn chế read-only. Để chạy hoặc đánh giá một submission yêu cầu có ít nhất một máy chủ jail. Dịch vụ jail cần có bản phân phối Linux tương thích với Ubuntu hoặc Red Hat.

Các kiểu thực thi của jail server

Jail server chấp nhận cho cả yêu cầu thực thi có tương tác (interactive requests) và không tương tác (non-interactive requests). Điểm khác biệt là yêu cầu thứ 2 cần dữ liệu gửi đi bao gồm một khoá bảo mật (key), một server và một cổng giao tiếp (communication port) để điều hướng đầu vào và kết quả thực thi (input/output) thông qua một bảng điều khiển (I/O console).



Hình 9 Hai yêu cầu thực thi tương tác và không tương tác

Công nghệ chroot jail được server jail của VPL sử dụng để tạo ra môi trường thực thi cho source code. Với mỗi source code được submit, jail server sẽ tạo ra một chroot jail tương ứng với request id của submission đó. Do vậy, mỗi submission sẽ được thực thi trong các môi trường hoàn toàn độc lập nhau và độc lập với thư mục root của hệ điều hành

Các bước cài đặt và ứng dụng thực tế của Jail server với VPL trong Moodle

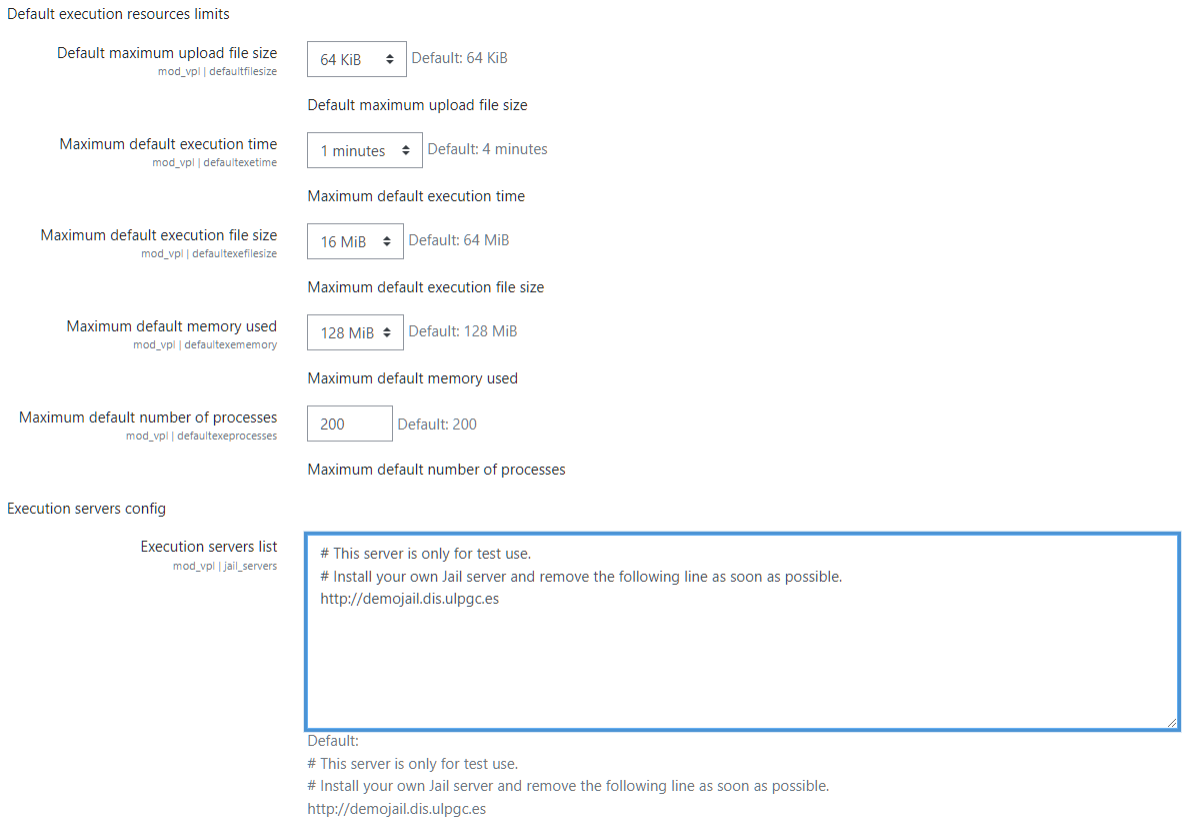
Bước 1: Lựa chọn phần cứng

Tùy chọn được khuyến nghị là sử dụng máy chủ độc lập dành riêng cho jail server. Nếu chúng ta không thể sử dụng máy tính riêng thì có thể sử dụng máy ảo trên máy chủ, ví dụ sử dụng VirtualBox hoặc VMware. Phương pháp này sẽ cung cấp sự cô lập và giới hạn tài nguyên sử dụng bởi jail server. Nếu cài đặt jail server trên máy với nhiều mục đích sử dụng khác, lưu ý rằng việc sử dụng tài nguyên bởi VPL-Jail-System có thể làm giảm hiệu suất của các dịch vụ khác. Mặc dù chưa có báo cáo về việc vi phạm bảo mật, ví dụ như thực thi mã nguồn từ xa, việc này mang theo một mối đe doạ tiềm ẩn.

Bước 2: Chuẩn bị cho hệ thống và cài đặt jail server

Cài đặt VPL jail server trên một trong những hệ điều hành Linux (Ubuntu, Fedora, CentOs…). Cài đặng ngôn ngữ hệ thống (locale) “en_US.UTF-8” dùng chung cho VPL và Moodle.

Bước 3: Cấu hình VPL jail server



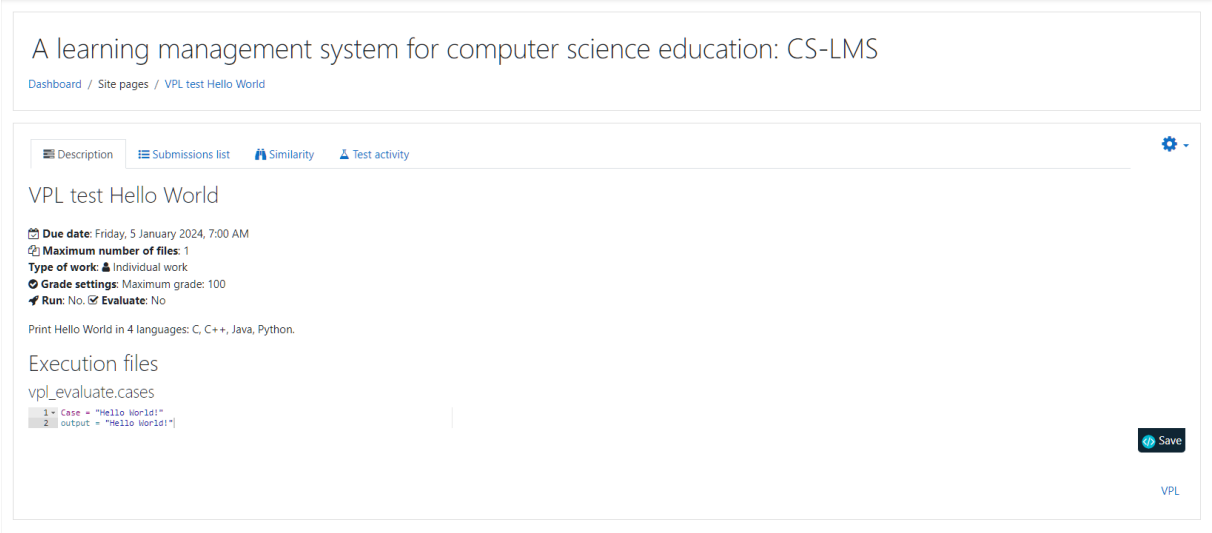
Hình 10 Cấu hình VPL jail server

Sau khi cài đặt đã hoàn tất, ta cần cấu hình cho jail server để giới hạn tài nguyên hệ thống, với các thông số:

- Default maximum upload file size: Giới hạn dung lượng file được nộp lên
- Maximum default execution time: Giới hạn thời gian thực thi của một tiến trình
- Maximum default execution file size: Giới hạn dung lượng file thực thi
- Maximum default memory used: Giới hạn tài nguyên bộ nhớ RAM cho một tiến trình
- Maximum default number of processes: Giới hạn số tiến trình trong một khoảng thời gian nhất định
- Execution servers list: Địa chỉ của jail server

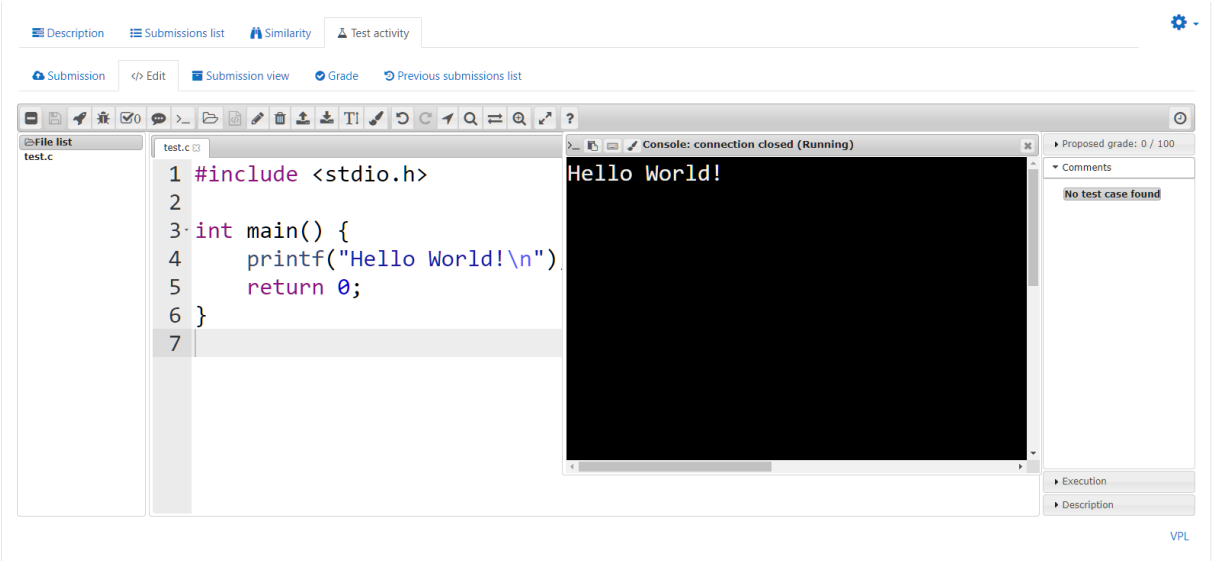
Ứng dụng VPL trong Moodle cho các bài tập lập trình:

- Giao diện mô tả câu hỏi: Chứa thông tin hạn nộp, điểm, mô tả và test case của câu hỏi.

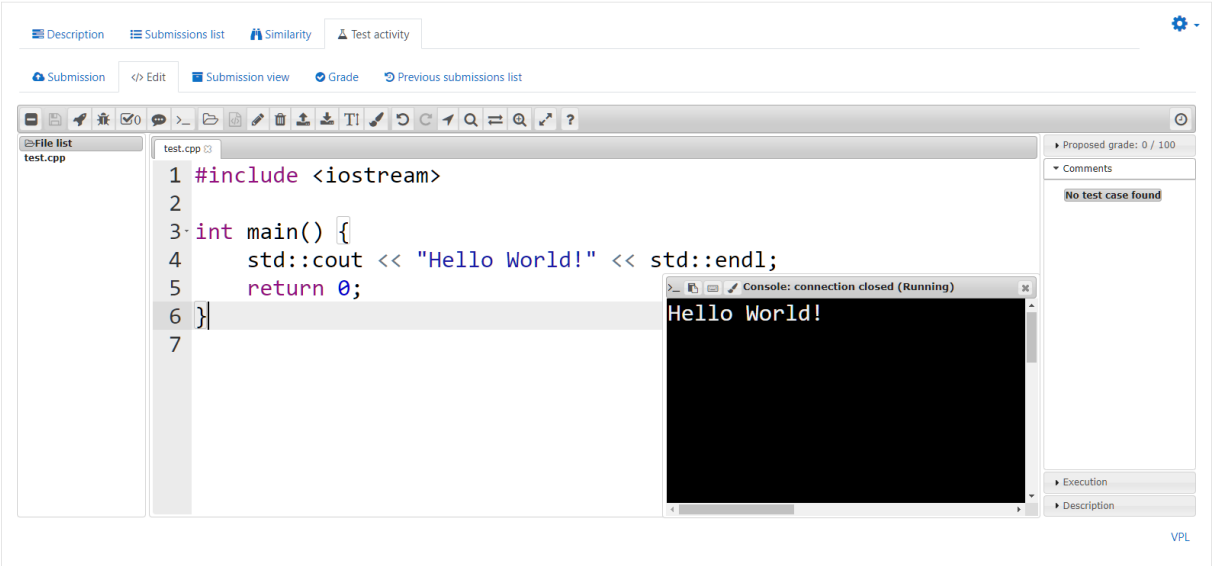


Hình 11 Giao diện mô tả câu hỏi của VPL

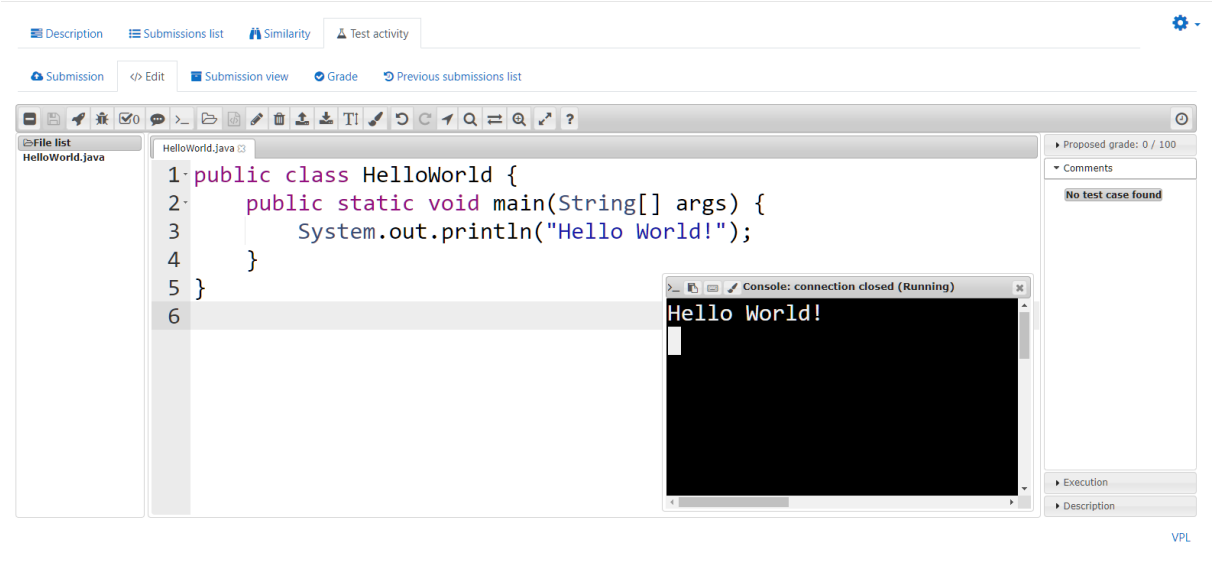
- Giao diện trình soạn thảo code và chạy code cho các chương trình C, C++, Java, Python: Mỗi khi chạy code, trình soạn thảo code sẽ nổi lên một bảng điều khiển in ra kết quả thực thi của chương trình.



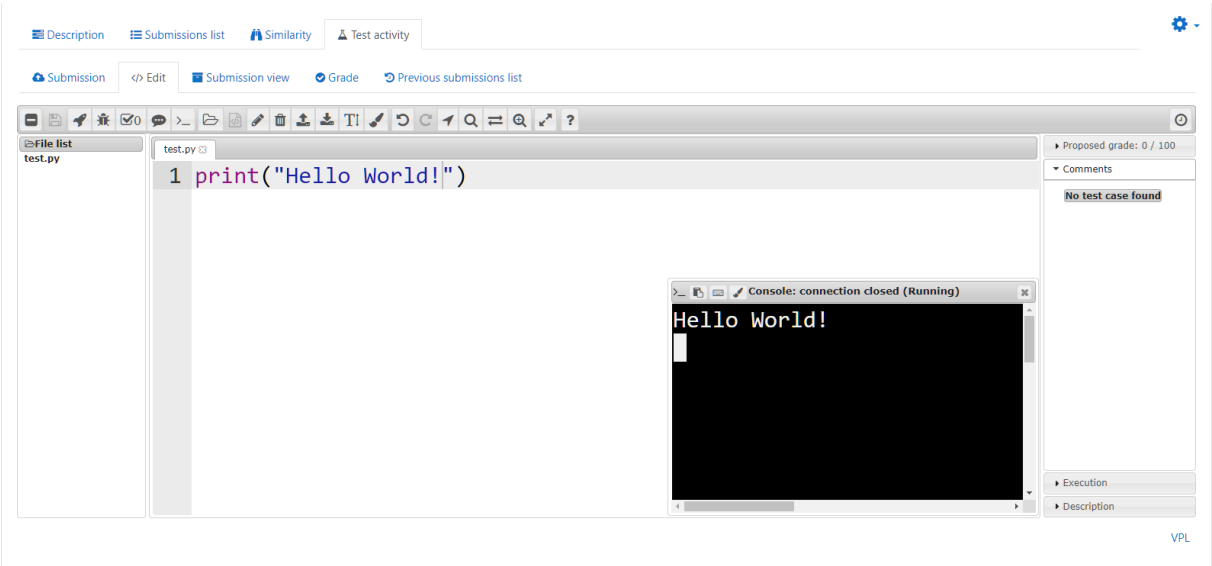
Hình 12 Giao diện VPL chạy chương trình C



Hình 13 Giao diện VPL chạy chương trình C++

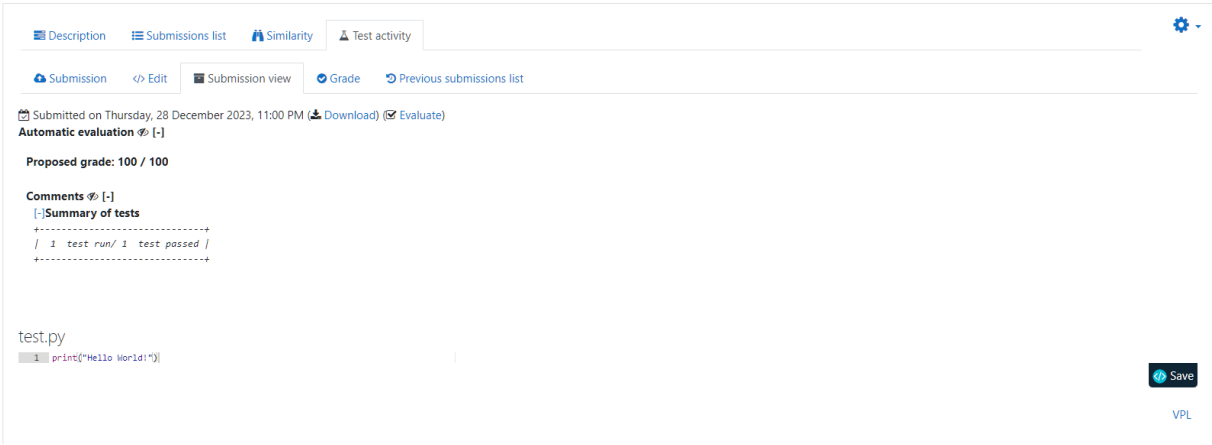


Hình 14 Giao diện VPL chạy chương trình Java



Hình 15 Giao diện VPL chạy chương trình Python

- Giao diện lịch sử submit code và chấm điểm: VPL sẽ hiển thị ra kết quả của các test case, tổng điểm của bài dựa trên các test case đúng.



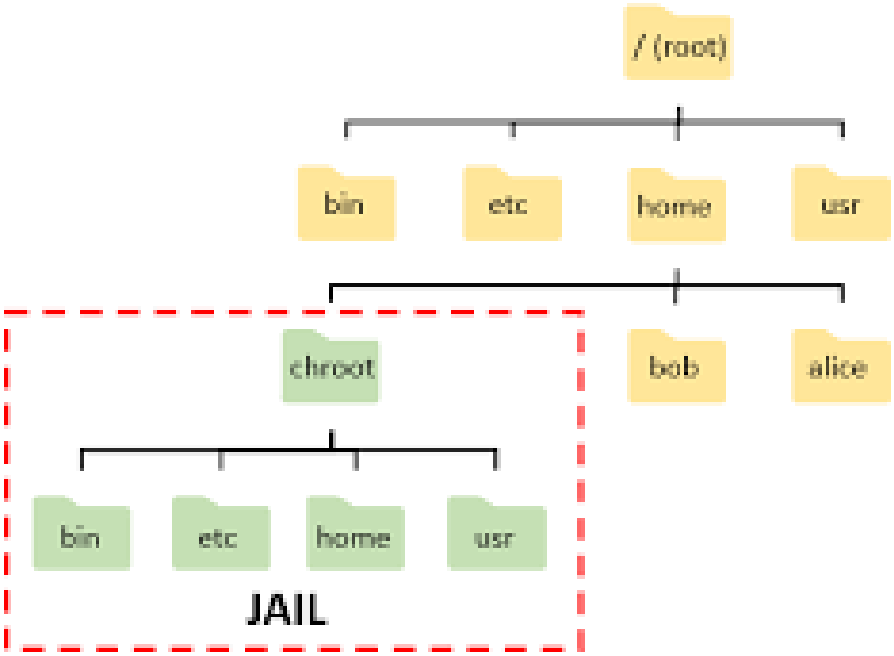
Hình 16 Giao diện lịch sử submit code và chấm điểm VPL

Các vấn đề liên quan tới chroot jail

Mặc dù hỗ trợ các tính năng về giao diện rất tiện lợi cho các bài tập lập trình, việc VPL sử dụng môi trường biên dịch và thực thi chroot jail có thể mang lại các rủi ro khó lường về bảo mật.

Một chroot (hay còn được hiểu là change root - thay đổi quyền cao nhất của một hệ điều hành) là một hoạt động UNIX nhằm thay đổi thư mục root thành một thư mục được định nghĩa bởi user của hệ điều hành. Bất kỳ tiến trình nào chúng chạy sau khi một hoạt động chroot chỉ có quyền truy cập vào thư mục root mới được định nghĩa và các

thư mục con của nó. Hoạt động này thông thường được gọi là một "chroot jail" bởi vì những tiến trình này không thể đọc hoặc ghi ra ngoài thư mục gốc mới. Rõ ràng hơn, khi chúng ta thực hiện hàm gọi này, chúng ta đơn giản là thay đổi "root" của tiến trình, trong đó root ở đây là root của hệ thống tệp (file system). Nếu chúng có một bố cục hệ thống tệp điển hình và chúng ta gọi câu lệnh chroot("/tmp"), thì / (gốc của hệ thống tệp) của chúng ta giờ đây thực sự là /tmp/. Điều này đi kèm với một số vấn đề tiềm ẩn vì tiến trình của chúng ta giờ đây không thể nhìn thấy bất kỳ tệp nhị phân (binaries) hoặc thư viện (libraries) nào khác. Hệ thống giờ đây chỉ nhìn thấy những gì có trong /tmp/ như là gốc hệ thống tệp mới.



Hình 17 Cách hoạt động chroot tạo ra một chroot jail

Chroot và người dùng root

Hàm gọi hệ thống chroot chỉ có sẵn cho người dùng root. Người dùng không phải là root không thể thực hiện hàm gọi chroot. Nếu chúng ta có thể gọi chroot thì chính chúng ta có thể thoát khỏi chế độ chroot này. Root có quyền truy cập vào hệ thống và tất cả các tài nguyên. Không có gì ngăn cản root từ việc sửa đổi trực tiếp bộ nhớ của tiến trình để thay đổi vị trí chroot trở lại '/'. Do đó quyền của người dùng root là tuyệt đối.

Chroot và người dùng bình thường (non-root user)

Chúng ta biết rằng chỉ có root mới có thể thực hiện hàm gọi hệ thống chroot. Nếu chúng ta khóa một người dùng thông thường trong một chroot, nó có thể an toàn hơn. Điều này gần như đúng, cho đến khi chúng ta nhìn vào một bức tranh lớn hơn. Khi chúng ta xem xét toàn bộ hệ thống, chúng ta không thấy được bảo mật thực sự từ chroot.

Khoá một người dùng thông thường trong một chroot sẽ ngăn họ truy cập vào phần còn lại của hệ thống. Việc sử dụng chroot có thể không làm hệ thống kém bảo mật đi, nhưng cũng không làm hệ thống bảo mật hơn nhiều. Cho dù chúng ta đã cấu hình đúng quyền cho các người dùng trên hệ thống của mình, chúng ta cũng sẽ không an toàn hơn khi ở trong một chroot so với việc phụ thuộc vào quyền hệ thống để kiểm soát một người dùng. Tất nhiên, quá trình phân quyền có thể có lỗ hổng, vì vậy việc chạy các tiến trình trong một chroot an toàn hơn so với việc chạy tiến trình ngoài chroot, nơi có thể xảy ra sai cấu hình phân quyền hệ thống. Điều này có thể đúng, nhưng việc thiết lập một chroot có thể phức tạp hơn nhiều so với việc cấu hình một hệ thống. Những sai sót trong thiết lập môi trường chroot có thể dẫn đến việc môi trường chroot ít an toàn hơn so với môi trường không chroot. Chroot không phải là không có ích, mà việc sử dụng nó có nghĩa là chúng ta an toàn tuyệt đối. Nó có thể làm cho công việc của hacker khó khăn hơn một chút nhưng sẽ không ngăn cản được một hacker cao cấp.

Tấn công chroot

Một dịch vụ daemon (dịch vụ ngầm) đang chạy trong một chroot có thể có một lỗ hổng mà cho phép hacker thực hiện các câu lệnh với đặc quyền của người dùng chạy dịch vụ daemon (tấn công thực hiện lệnh tùy ý). Kể cả khi dịch vụ daemon đã giới hạn quyền và các lệnh không được phép thực thi với đặc quyền root, có thể hacker không thể thoát khỏi chroot, nhưng hacker vẫn có thể sử dụng tài nguyên hệ thống, chẳng hạn như gửi thư rác, chiếm quyền quyền truy cập mạng cục bộ, tham gia vào một botnet, ….

Nếu một hacker muốn thực thi một ứng dụng cụ thể, nhưng ứng dụng đó không được liên kết cứng (hard link) vào chroot, vẫn có khả năng hacker có thể thực thi nó: dịch vụ daemon chạy trong chroot có thể có một lỗ hổng mà cho phép hacker kiểm soát luồng thực thực thi (execution flow) của dịch vụ. Lỗ hổng này cho phép hacker thực thi mã lệnh độc tùy ý (Ví dụ một bản sao của ứng dụng mà hacker ban đầu muốn chạy, nhưng không có trong chroot).

Trong hai ví dụ trên, phân quyền hệ thống đã có cùng hiệu quả bảo mật như chroot jail (hacker có thể thực hiện các lệnh hoặc mã lệnh tùy ý trong cả hai trường hợp). Nguy hiểm hơn nữa, một cuộc tấn công khác có thể xảy ra nếu hacker có thể đạt được quyền root. Ngày nay, hầu hết các lỗ hổng nâng cao đặc quyền đều xuất phát từ các lỗ hổng kernel. Những lỗ hổng này có thể tận dụng từ bên trong chroot.

Kết luận, không khó để xem hàm gọi hệ thống chroot như một tính năng bảo mật. Về mặt lý thuyết thì nghe có vẻ tuyệt vời, nhưng nếu chúng thực sự dành thời gian để tìm hiểu rõ hơn thì đó không phải là một tính năng bảo mật, mà nó gần như chỉ là một tính năng tăng cường. Chroot có thể làm chậm một hacker, nhưng trong hầu hết các tình huống, chroot không thể làm được điều này.

Ngoài ra, việc sử dụng chroot cũng mang lại nhiều nhược điểm:

- Quản lý môi trường thực thi gặp nhiều khó khăn

- Dependencies và thư viện: Một trong những thách thức lớn khi sử dụng chroot jail là quản lý dependencies và thư viện. Các ứng dụng thường phụ thuộc vào các thư viện cụ thể, và chúng cần được cài đặt hoặc sao chép vào môi trường chroot. Quản lý các dependencies và thư viện này có thể gặp phải khó khăn, đặc biệt là khi có sự phụ thuộc phức tạp.
- Không đồng bộ dữ liệu toàn hệ thống: Chroot không đồng bộ mọi dữ liệu của hệ thống. Các tài nguyên như biến môi trường, ngày và giờ hệ thống có thể không được chroot đồng bộ với hệ thống root. Điều này có thể tạo ra những sai lệch không mong muốn nếu ứng dụng yêu cầu một môi trường hoàn chỉnh.
- Khó khăn trong việc tạo và duy trì chroot jail: Việc tạo và duy trì chroot jail có thể đòi hỏi kiến thức kỹ thuật sâu rộng về hệ thống và Linux. Sự phức tạp tăng lên khi cần phải xử lý các cấu hình đặc biệt hoặc khi triển khai chroot jail trên nhiều hệ điều hành hoặc phiên bản Linux khác nhau.

- An toàn nhưng không tuyệt đối

- Bảo mật gia tăng: Mặc dù chroot cung cấp một lớp bảo mật gia tăng bằng cách hạn chế quyền truy cập của tiến trình, nó không phải là một biện pháp bảo mật

tuyệt đối. Có những kỹ thuật và công cụ có thể vượt qua chroot, chẳng hạn như các kỹ thuật đặc biệt của hacker hoặc malware.

- Không an toàn mọi loại tấn công: Chroot chủ yếu là một biện pháp bảo mật ở mức độ cơ bản và không thể chống lại những phương pháp phức tạp hơn như tấn công kernel hoặc các lỗ hổng bảo mật cấp cao. Hiện nay các hacker đã tìm ra cách để "vượt ngục" (break jail) để chiếm quyền kiểm soát root của hệ điều hành.
- Chỉ cung cấp bảo vệ tạm thời: Các chuyên gia đã khuyến nghị chỉ sử dụng chroot như một giải pháp bảo mật tạm thời chứ không phải giải pháp bảo mật hoàn hảo. Trong một số trường hợp, các biện pháp bảo mật mạnh mẽ hơn như containers hoặc các giải pháp ảo hóa có thể được ưa chuộng để đạt được mức độ an toàn cao hơn.

Như chúng ta đã thấy, công nghệ jail server này của Moodle có nhưng không đủ mức độ bảo mật tuyệt đối cho hệ thống cũng như khả năng mở rộng linh hoạt. Vì vậy chúng ta cần phải tìm một công nghệ khác có thể đáp ứng được vấn đề này.

2.2. Ý tưởng xây dựng hệ thống biên dịch và thực thi

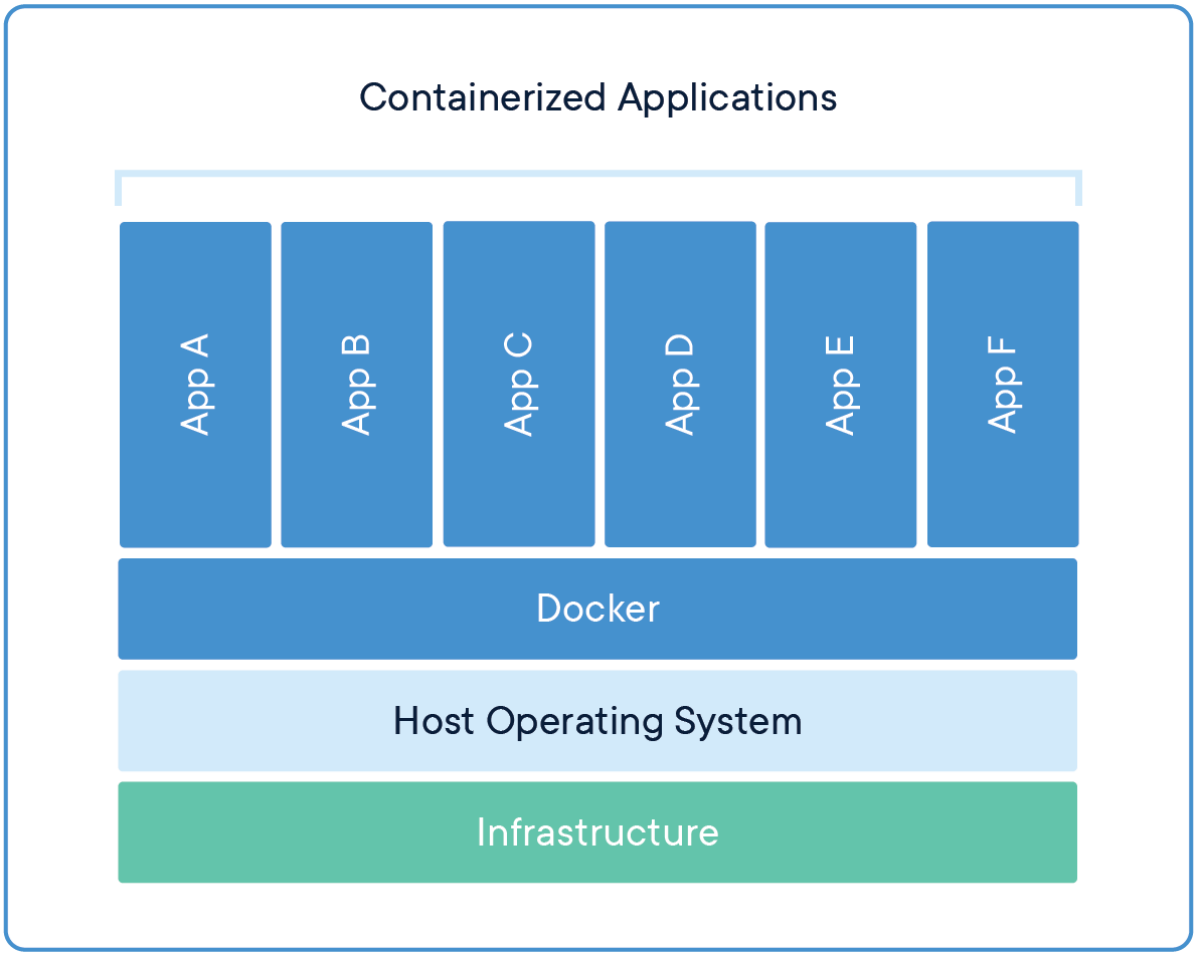
Trong điều kiện lý tưởng, giải pháp đơn giản nhất là một server biên dịch code và trả về output. Tiếp theo, server này phải hỗ trợ nhiều ngôn ngữ. Do đó, server phải được cài đặt nhiều trình biên dịch cho các ngôn ngữ bậc cao hiện nay và xuất ra một API để client có thể gửi code và nhận lại kết quả chạy. Nhưng tất nhiên chúng ta không thể hoàn toàn tin tưởng bất kỳ đoạn code nào mà người dùng submit lên. Lý do chính là những vấn đề bảo mật đã được nêu ra ở bên trên. Sau khi vấn đề bảo mật được giải quyết, cần hạn chế tài nguyên và thời gian thực thi cho từng submission để nâng cao hiệu suất hệ thống. Với yêu cầu hệ thống cần giới hạn về CPU, bộ nhớ RAM và thời gian thực thi máy ảo (VMware) hoặc Docker sẽ là một sự lựa chọn hợp lý.

Với máy ảo, nó có thể đáp ứng những yêu cầu trên rất tốt. Nhưng trong trường hợp này, máy ảo sẽ không đáp ứng được hiệu năng do việc tạo một môi trường ảo chỉ để chạy 1 submission là quá nặng cho hệ thống. Chúng ta cần một môi trường có khả năng khởi nhanh chóng với dung lượng nhẹ (lightweight), và Docker sẽ đáp ứng được tốt việc này.

2.3. Docker

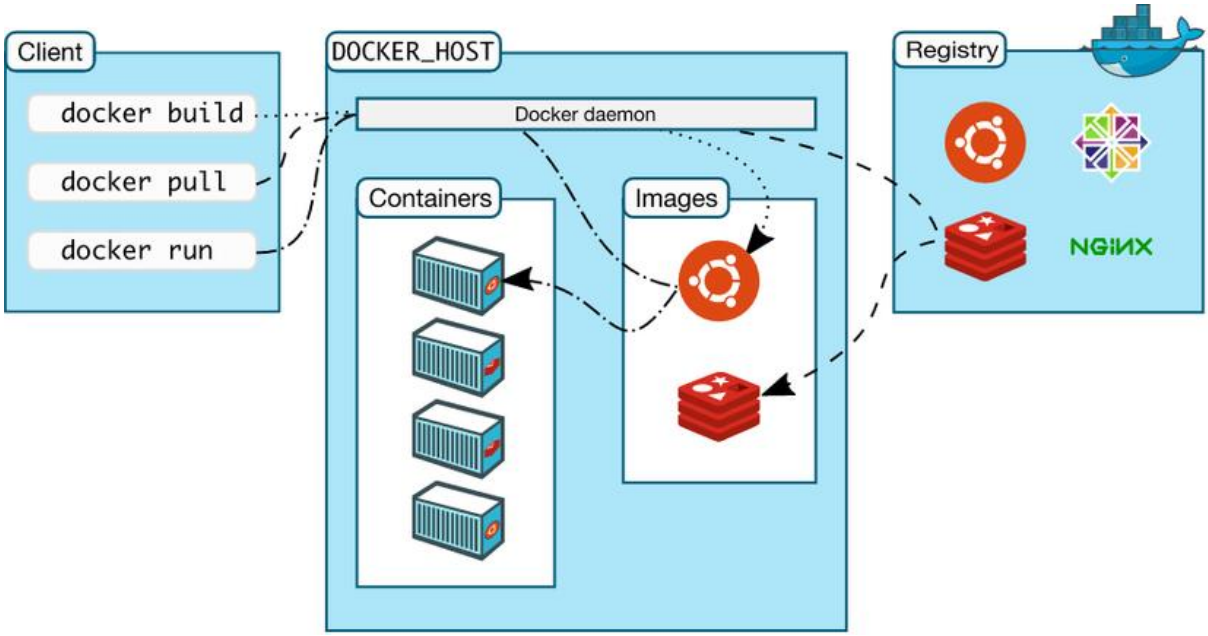
Giới thiệu về Docker

Docker là một nền tảng ứng dụng được thiết kế để tạo, triển khai và chạy các ứng dụng trong các môi trường ảo được gọi là "containers". Containers là một cách để đóng gói và di động ứng dụng cùng với tất cả các thư viện và dependencies cần thiết để chúng chạy đúng cách, đảm bảo tính nhất quán và đồng nhất giữa môi trường phát triển và môi trường triển khai.



Hình 18 Docker và các ứng dụng được container hoá

2.3.1. Kiến trúc của Docker



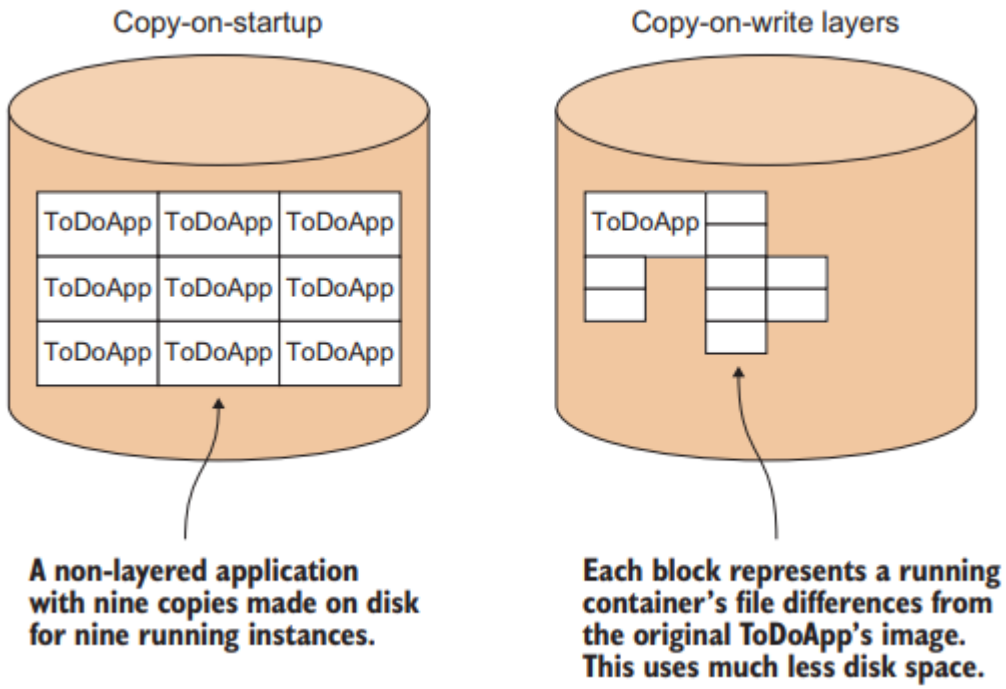
Hình 19 Mô hình kiến trúc của Docker

- Docker Engine: là thành phần chính của Docker, như một công cụ để đóng gói ứng dụng
- Docker Hub: là một "github for docker images". Trên DockerHub có hàng ngàn public images được tạo bởi cộng đồng cho phép chúng ta dễ dàng tìm thấy những image mà chúng ta cần. Và chỉ cần pull về và sử dụng với một số config mà chúng ta mong muốn.
- Images: là một khuôn mẫu để tạo một container. Thường thì image sẽ dựa trên 1 image có sẵn với những tùy chỉnh thêm. Ví dụ chúng ta cần build 1 image dựa trên image Centos mẫu có sẵn để chạy Nginx và những tùy chỉnh, cấu hình để ứng dụng web của chúng ta có thể chạy được. Chúng ta có thể tự build một image riêng cho mình hoặc sử dụng những image được chia sẻ từ cộng đồng Docker Hub. Một image sẽ được build dựa trên những chỉ dẫn của Dockerfile.
- Container: là một instance của một image. Bạn có thể create, start, stop, move or delete container dựa trên Docker API hoặc Docker CLI.
- Docker Client: là một công cụ giúp người dùng giao tiếp với Docker host.
- Docker Daemon: lắng nghe các yêu cầu từ Docker Client để quản lý các đối tượng như Container, Image, Network và Volumes thông qua REST API. Các Docker Daemon cũng giao tiếp với nhau để quản lý các Docker Service.

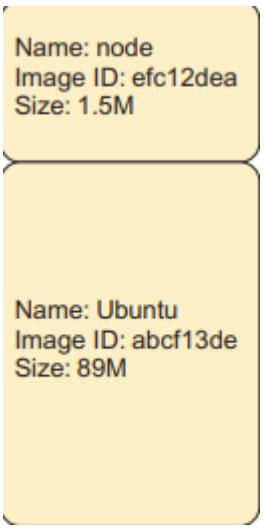
- Dockerfile: là một tập tin bao gồm các chỉ dẫn để build một image.
- Volumes: là phần dữ liệu được tạo ra khi container được khởi tạo.

Docker layering

Cơ chế xếp lớp của Docker giúp quản lý vấn đề lớn xuất hiện khi chúng ta sử dụng container ở quy mô lớn. Hãy tưởng tượng xem điều gì sẽ xảy ra nếu chúng ta khởi động hàng trăm hoặc thậm chí hàng nghìn ứng dụng, và mỗi ứng dụng đó đòi hỏi một bản sao của các tệp phải được lưu ở một nơi nào đó; disk space sẽ nhanh chóng cạn kiệt. Mặc định Docker sử dụng cơ chế copy-on-write để giảm lượng không gian đĩa cần thiết (xem hình). Khi một container đang chạy cần ghi vào một tệp, nó lưu lại thay đổi bằng cách sao chép mục đó vào một vùng mới của đĩa. Khi thực hiện một Docker commit, vùng mới của đĩa này sẽ được đóng băng và được lưu lại như một lớp với định danh riêng. Điều này làm cho các container Docker có thể khởi động nhanh chóng - chúng không cần phải copy bất kỳ dữ liệu cũ nào vì tất cả dữ liệu đã được lưu trữ thành image.



Hình 20 Cơ chế phân lớp hệ thống của Docker



Hình 21 Phân lớp hệ thống cho môi trường thực thi của ngôn ngữ NodeJS

Hình 13 minh hoạ môi trường thực thi NodeJS có 2 lớp (layer). Những lớp này là static, vì vậy nếu chúng ta cần thay đổi bất kỳ điều gì ở một lớp cao hơn, chúng ta chỉ cần xây dựng trên image mà chúng ta sử dụng làm tham chiếu. Trong trường hợp này, môi trường thực thi NodeJS đã được xây dựng trên image Ubuntu có sẵn.

Cả hai lớp này có thể được chia sẻ giữa nhiều container đang chạy, giống như cách một thư viện được chia sẻ trong bộ nhớ giữa nhiều tiến trình đang chạy. Đó là một tính năng quan trọng cho vận hành cho phép chúng ta chạy nhiều container dựa trên các image khác nhau trên các máy chủ mà không lo bị hết disk space. Với các môi trường thực thi cho các ngôn ngữ khác C, C++, Java, Python chúng ta sẽ xây dựng một image tương tự (C, C++ dùng image gcc, Java dùng image OpenJDK…)

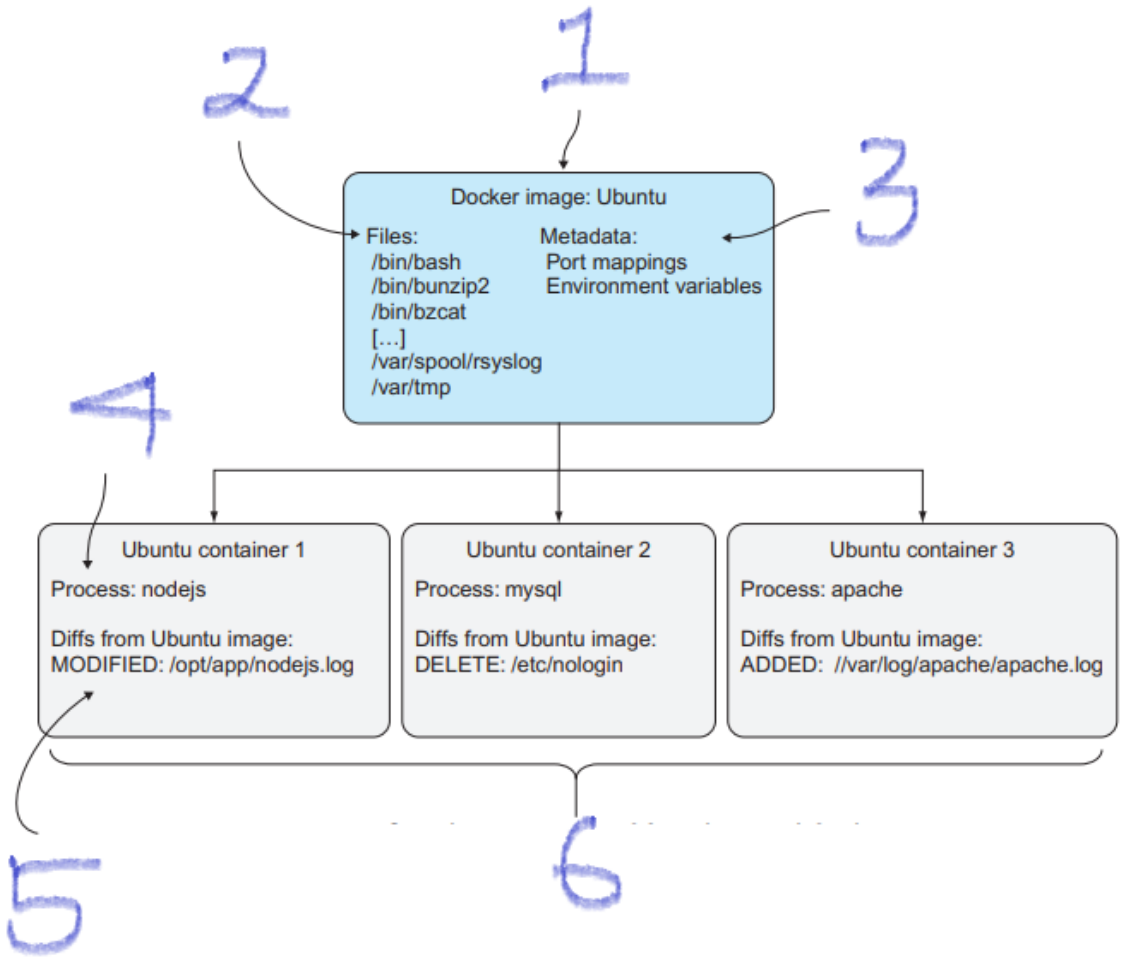
Image và container trong Docker

Image và container là hai khái niệm quan trọng nhất trong Docker.

Các containers cho phép lập trình viên đóng gói một ứng dụng với tất cả các phần cần thiết, chẳng hạn như thư viện và các phụ thuộc khác, và gói tất cả ra dưới dạng một package. Bằng cách đó, nhờ vào container, ứng dụng sẽ chạy trên mọi máy Linux khác bất kể mọi cài đặt tùy chỉnh mà máy có thể có khác với máy được sử dụng để viết code.

Một cách để nhìn vào images và container, chúng tương tự như chương trình và tiến trình. Tiến trình là một "ứng dụng được thực thi" thì Docker container có thể được nhìn như một Docker Image đang được thực thi. Gần gũi với các lập trình viên hơn nữa là về góc nhìn hướng đối tượng, có thể coi image là lớp (class) và container chính là

thực thể (object). Chúng ta có thể tạo nhiều container từ một image, và chúng hoàn toàn biệt lập với các container các như cách mà các object hoạt động. Mỗi container có thể có các cấu hình (configuration) khác nhau, nhưng cho dù chúng ta có thay đổi gì đi nữa thì cũng không ảnh hưởng đến "bản thiết kế" image.



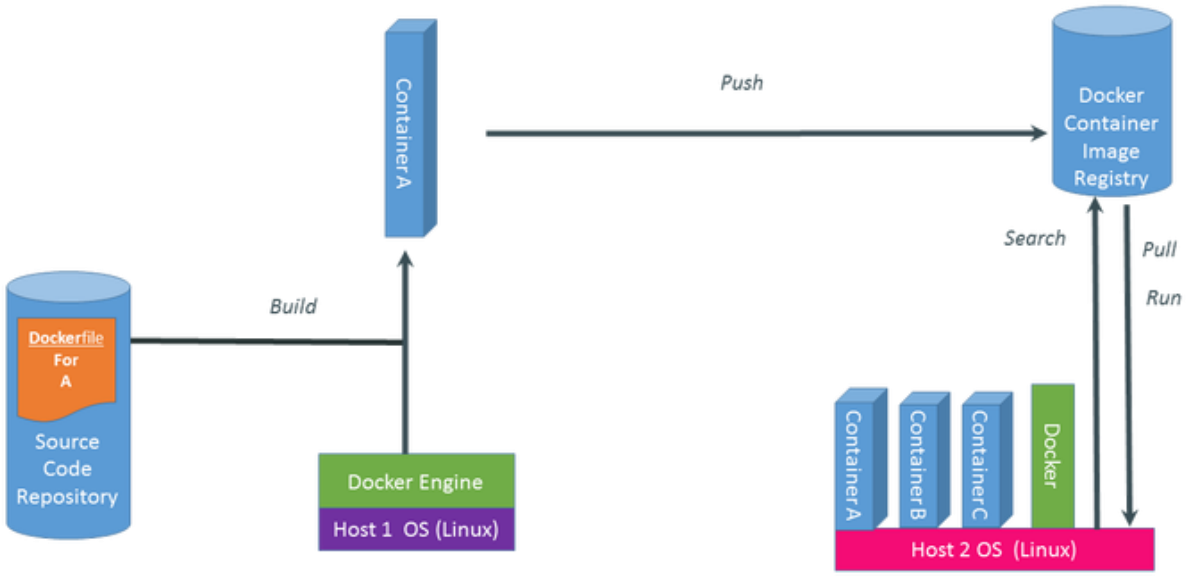
Hình 22 Các khái niệm của Docker image và container

Hình 14 mô tả các khái niệm về Docker image và container như sau:

1. Một Docker image bao gồm các file và metadata (siêu dữ liệu chứa dữ liệu về các dữ liệu). Đây là image cơ sở (base image) cho các container bên dưới.

2. Các file trong image sẽ chiếm hầu hết các không gian. Do các container hoàn toàn biệt lập nên chúng phải có các cấu hình riêng, bao gồm môi trường thực thi cho các ngôn ngữ hay thư viện riêng.
3. Metadata chứa thông tin về các biến môi trường (environment variable), port mappings (cấu hình cổng), volumes, …
4. Các container chạy một tiến trình lúc khởi động. Khi tiến trình khởi động này hoàn thành, container sẽ dừng lại. Các tiến trình khởi động này có thể sinh ra các tiến trình con khác.
5. Các thay đổi về file được lưu trong container với cơ chế copy-on-write. Khi có thay đổi, sẽ tạo ra một bản sao của image cơ sở và lưu thay đổi trên image mới đó. Do vậy, image cơ sở sẽ không bị ảnh hưởng bởi các container.
6. Các container được tạo từ các image, kế thừa file hệ thống từ chúng, và sử dụng metadata của các image này để xác định cấu hình khởi động. Các container được tách biệt nhưng chúng cũng có thể được cấu hình để có thể giao tiếp với nhau.

Xây dựng và quản lý Docker



Hình 23 Quy trình quản lý Docker

- Build

  Đầu tiên tạo một dockerfile, trong dockerfile này chính là code của chúng ta. Dockerfile này sẽ được Build tại một máy tính đã cài đặt Docker Engine. Sau khi

build ta sẽ có được Container, trong Container này chứa ứng dụng kèm bộ thư viện của chúng ta.

- Push

  Sau khi có được Container, chúng ta thực hiện push Container này lên cloud và lưu tại đó.

- Pull, Run

  Nếu một máy tính khác muốn sử dụng Container chúng ta thì bắt buộc máy phải thực hiện việc Pull container này về máy, tất nhiên máy này cũng phải cài Docker Engine. Sau đó thực hiện Run Container này.

2.3.2. So sánh Docker container và Chroot jail

Docker container không sử dụng môi trường chroot. Trong khi chroot là một cuộc gọi hệ thống Unix cho phép thay đổi thư mục gốc của một quy trình và các tiến trình con của nó, Docker container sử dụng một cơ chế khác gọi là các không gian tên (namespaces) để cách ly.

Các không gian tên trong Linux cung cấp cách ly ở cấp độ tiến trình (process-level isolation) bằng cách tạo ra các bản sao (instances) riêng biệt của các tài nguyên hệ thống như hệ thống tệp, network, process id, user id và nhiều tài nguyên khác. Docker sử dụng các không gian tên để tạo ra một môi trường cách ly cho mỗi container, cho phép các tiến trình trong container trong khuôn khổ container và ngăn chúng khỏi việc truy cập vào tài nguyên bên ngoài không gian tên đã được chỉ định của chúng.

Trong trường hợp của hệ thống tệp, Docker sử dụng sự kết hợp giữa các hệ thống tệp liên minh và cơ chế copy-on-write để cung cấp cho container một lớp hệ thống tệp cách ly của riêng mình. Điều này cho phép mỗi container có hệ thống tệp gốc riêng mà không cần thay đổi thư mục gốc của hệ thống máy chủ.

Mặc dù chroot có thể được sử dụng trong một Docker container nếu cần thiết, nhưng nó không phải là cơ chế chính để đạt được việc đóng gói container hoặc cách ly. Docker dựa vào sự kết hợp giữa các không gian tên, nhóm kiểm soát (cgroups) và các tính năng khác của hạt nhân Linux để cung cấp một giải pháp đóng gói container hiệu quả và nhẹ nhàng.

Tóm lại, các container Docker không phụ thuộc vào một môi trường chroot. Thay vào đó, chúng sử dụng các không gian tên Linux và các tính năng hạt nhân khác để cung cấp cách ly ở cấp độ quy trình và cách ly hệ thống tệp, tạo điều kiện cho việc đóng gói hiệu quả và an toàn.

Mức độ bảo mật của Docker container so với chroot jail

Docker tạo ra các "container" đang chạy, các container này phụ thuộc vào nhân của hệ thống máy chủ (host system kernel) và chỉ đóng gói các tệp nhị phân (binaries) và dependency cần thiết để chạy tệp nhị phân mong muốn. Tất nhiên, chúng ta có thể làm cho tệp nhị phân đó trở thành một quá trình khởi đầu (init process) khác mà khởi chạy các tệp nhị phân khác, nhưng điều này thường là một mô hình anti-pattern trong việc đóng gói container.

Thêm vào đó, các container được cách ly (separated) theo nhiều cách hơn so với chroot. Sự cách ly được thực hiện thông qua các không gian tên (namespaces), trong đó một không gian tên cách ly tài nguyên của nó khỏi các không gian tên khác. Ở mức độ bảo mật cao hơn, hệ thống tệp container, process ids, network stacks và người dùng đều được cách ly thông qua các không gian tên, tách biệt chúng khỏi các container khác (và khỏi máy chủ host machine). Ngoài ra, mỗi container có không gian tên uts (unix time sharing), một tên máy chủ riêng, bộ nhớ chia sẻ riêng (dedicated shared memory).

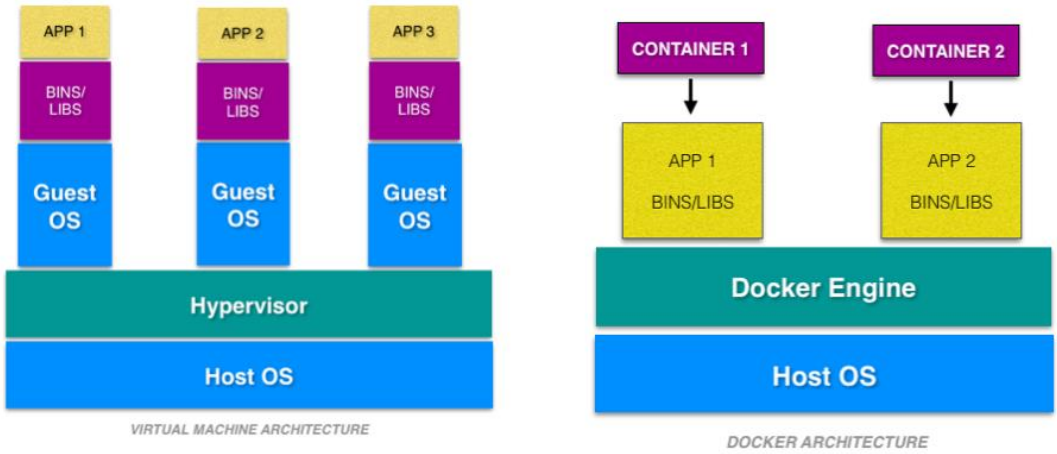
Ngoài ra, hệ thống tệp của một container được phân lớp (layered) sao cho chúng ta có thể xây dựng, thêm/đổi chức năng thành một image mới từ một image cơ sở ban đầu.

Vì vậy, Docker đã đưa khả năng bảo mật của ảo hoá (virtualization) và cách ly (isolation) lên một tầm cao mới so với chroot.

2.3.3. So sánh Docker container và máy ảo

Kiến trúc

Sự khác biệt chính giữa Docker và máy ảo (VM) nằm ở kiến trúc bên dưới.



Hình 24 Kiến trúc của máy ảo và Docker container

Các máy ảo sẽ có một hệ điều hành chủ và nhiều hệ điều hành khách trong từng máy ảo. Hệ điều hành khách này có thể là bất kỳ hệ điều hành nào như Linux và Windows, bất kể hệ điều hành chủ là gì. Ngược lại, các Docker container được host trên một server

vật lý duy nhất với một hệ điều hành chủ được chia sẻ giữa chúng. Việc chia sẻ hệ điều hành chủ giữa các container giúp chúng có dung lượng nhẹ và tăng đáng kể thời gian khởi động. Do đó, Docker container phù hợp để chạy nhiều ứng dụng trên một nhân hệ điều hành. Ngược lại, máy ảo chỉ cần thiết khi các ứng dụng hoặc dịch vụ yêu cầu phải chạy trên các hệ điều hành khác nhau.

Bảo mật

Bảo mật của máy ảo được tách biệt hoàn toàn với hệ điều hành chủ do chúng chạy hệ điều hành riêng, có nhân riêng và các tính năng bảo mật riêng. Do đó, người dùng phải rất thận trọng khi cấu hình bảo mật cho từng máy ảo.

Ngược lại với máy ảo, Docker container được chia sẻ các tính năng bảo mật với nhân của máy chủ, và chính Docker cũng đã cung cấp rất nhiều tính năng bảo mật để tránh việc một Docker container có thể chiếm quyền kiểm soát máy chủ.

Hiệu suất

Tất nhiên máy ảo sẽ chiếm nhiều tài nguyên hơn so với Docker container do phải chạy với cả một hệ điều hành để khởi động. Kiến trúc nhẹ của Docker container sẽ nhẹ hơn rất nhiều, chỉ mất 1 đến vài giây vì chúng chỉ cần khởi chạy một tiến trình gốc.

Hơn nữa, với trường hợp của máy ảo, các tài nguyên như CPU, memory, và I/O sẽ không được cấp phát vĩnh viễn cho máy ảo, khác với Docker container được cấp phát vĩnh viễn nhưng vẫn có khả năng mở rộng (scaling up) cực kỳ tốt.

2.3.4. Ưu điểm của Docker

Docker mang lại nhiều ưu điểm cho quá trình phát triển, triển khai và quản lý ứng dụng.

- Di động và tính nhất quán: Docker cho phép đóng gói tất cả dependencies và cấu hình của ứng dụng vào một container, giúp đảm bảo tính nhất quán giữa môi trường phát triển và triển khai.
- Chia sẻ và tái sử dụng: Docker images có thể được chia sẻ và tái sử dụng trên nhiều máy chủ và môi trường khác nhau. Điều này giúp giảm thời gian triển khai và đảm bảo tính nhất quán giữa các môi trường.

- Bảo trì và cập nhật dễ dàng: Docker giúp quản lý và duy trì ứng dụng một cách dễ dàng. Việc triển khai các bản cập nhật và sửa lỗi trở nên linh hoạt, không làm ảnh hưởng đến các containers đang chạy.
- Tích hợp linh hoạt: Docker có thể tích hợp dễ dàng với nhiều công nghệ khác nhau và được hỗ trợ trên nhiều nền tảng, từ on-premises đến cloud computing.
- Triển khai nhanh chóng: Docker giúp tăng tốc độ phát triển ứng dụng và giảm thời gian triển khai. Containers có thể được tạo nhanh chóng và triển khai một cách linh hoạt, đặc biệt là trong môi trường DevOps.
- Bảo mật: Containers cung cấp một mức độ cô lập giữa các ứng dụng, giảm rủi ro giao thức và tác động tiêu cực của một ứng dụng lên các ứng dụng khác. Docker cũng cung cấp các cơ chế bảo mật để giữ an toàn cho containers.
- Quản lý tài nguyên: Docker giúp quản lý tài nguyên một cách hiệu quả, từ việc quản lý CPU và bộ nhớ đến việc điều chỉnh tài nguyên dựa trên nhu cầu của ứng dụng.

Tổng kết lại, với một hệ thống biên dịch và thực thi code yêu cầu mở rộng linh hoạt, có khả năng giới hạn tài nguyên (CPU, RAM, thời gian thực thi), khởi động nhanh và dung lượng nhẹ thì Docker hoàn toàn có thể đáp ứng được điều này:

- Nếu bất kỳ một mã độc nào có ý định huỷ hoại hệ thống, phạm vi ảnh hưởng có nó chỉ có thể ở trong container mà nó hoạt động.
- Các container không cần thiết phải hoạt động mọi lúc, chúng được tạo ra để chạy một submit code của người dùng và sẽ bị huỷ (destroyed) khi container chạy xong việc của nó.
- Mỗi container là một instance của một image đã nói ở trên, và mỗi image sẽ được cài một trình biên dịch riêng. Điều này sẽ giúp việc quản lý trình biên dịch và các phiên bản của chúng trở nên dễ dàng hơn rất nhiều.

CHƯƠNG III: XÂY DỰNG HỆ THỐNG BIÊN DỊCH VÀ THỰC THI

3.1. Mục tiêu và yêu cầu khi xây dựng hệ thống biên dịch và thực thi

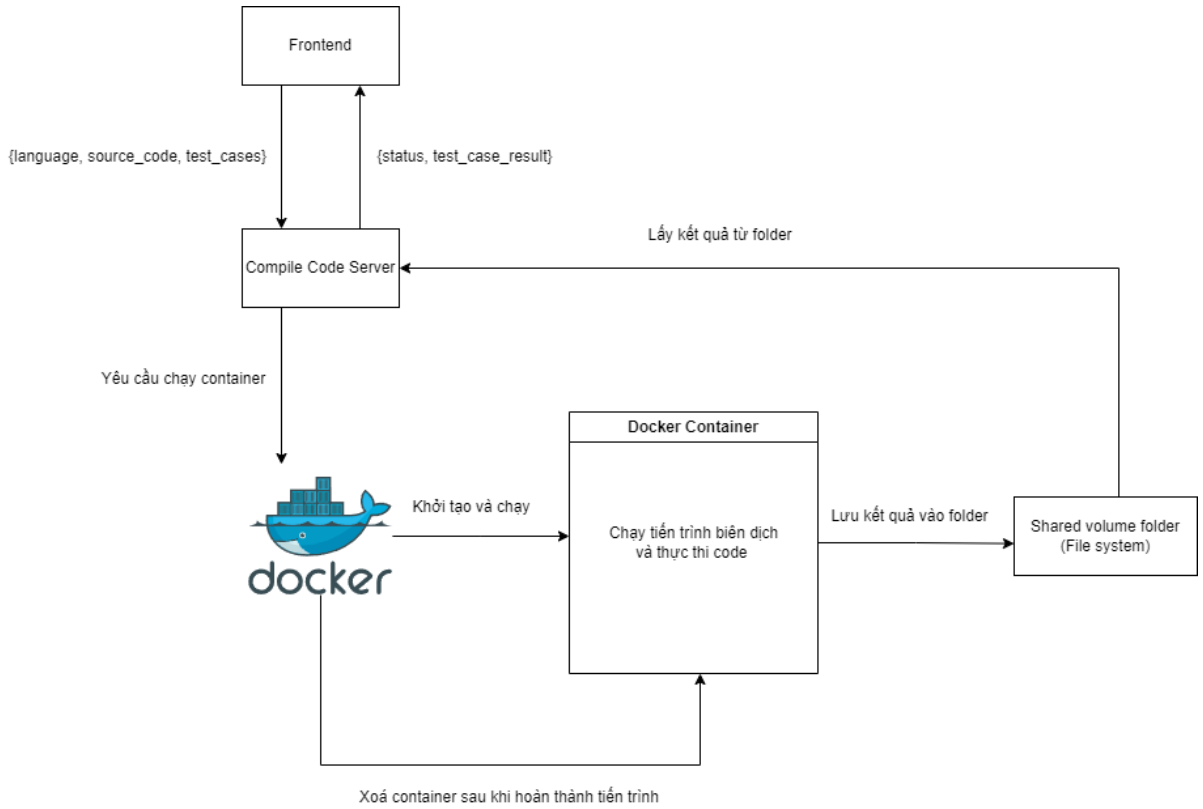
Mục tiêu xây dựng hệ thống:

- Môi trường biên dịch hiệu năng cao, ít lỗi
- Hỗ trợ nhiều ngôn ngữ lập trình phổ biến
- Bảo mật tốt, có khả năng ngăn chặn mã độc
- Tích hợp với frontend dễ dàng với API

Yêu cầu đặt ra cho hệ thống:

- Biên dịch được cho 4 ngôn ngữ phổ biến hiện nay: Java, C, C++, Python
- Giới hạn tài nguyên CPU, RAM, thời gian thực thi cho từng tiến trình biên dịch và thực thi
- Hệ thống hỗ trợ biên dịch tối đa 3 trường hợp kiểm thử cùng một lúc
- Hệ thống trả về thời gian biên dịch, tổng thời gian biên dịch, tổng thời gian thực thi cho từng trường hợp kiểm thử của submission
- Hệ thống trả về các trạng thái thực thi cho submission: Accepted (Hoàn thành), Wrong Answer (Kết quả sai), Compilation Error (Lỗi biên dịch), Out Of Memory (Hết bộ nhớ), Time Limit Exceed (Quá thời gian thực thi), Runtime Error (Lỗi thực thi hệ thống)

3.2. Kiến trúc tổng quan

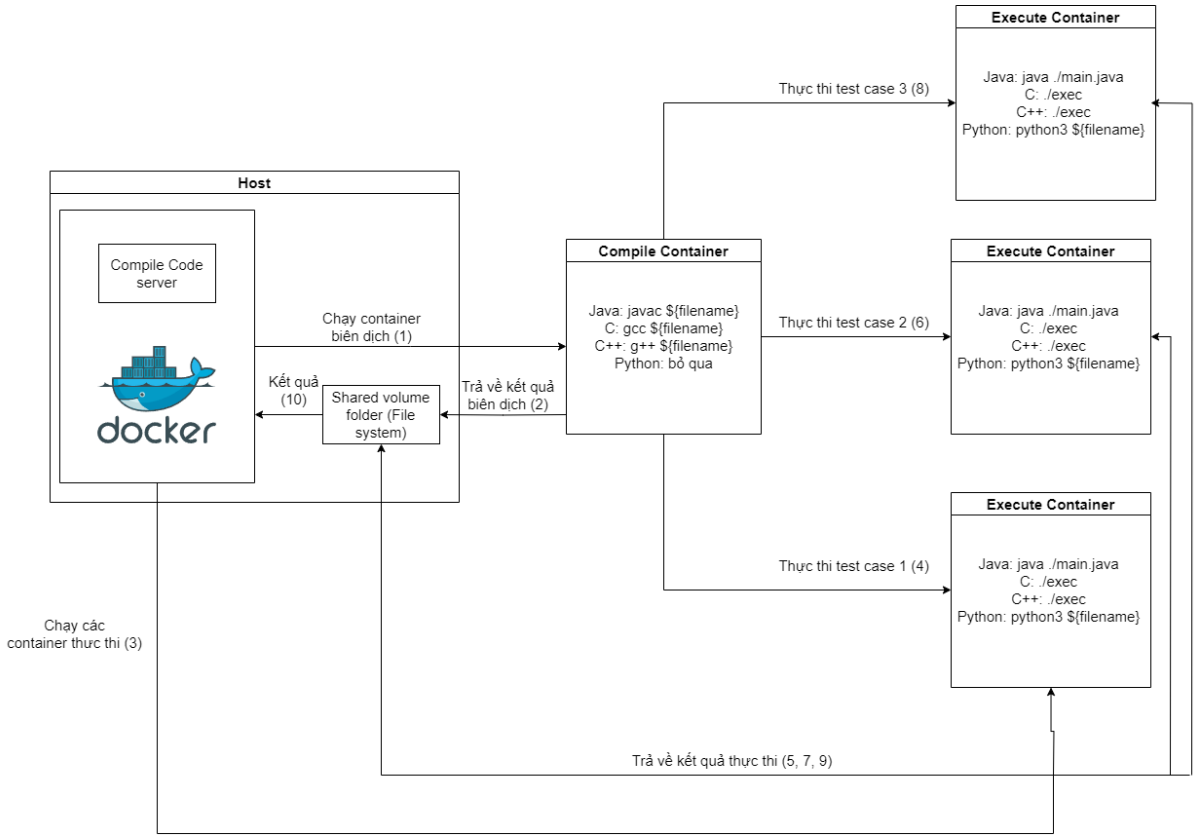


Hình 25 Kiến trúc tổng quan của hệ thống biên dịch và thực thi

Hình 17 mô tả kiến trúc tổng quan và luồng hoạt động của hệ thống biên dịch và thực thi như sau:

- Frontend gọi API của Compile Code Server với các tham số ngôn ngữ, mã nguồn, các trường hợp kiểm thử
- Compile Code Server sẽ yêu cầu Docker khởi tạo và chạy container theo lệnh `docker run`
- Docker container chạy file code của người dùng, sau đó lưu kết quả vào shared volume folder. Folder này được tạo ra khi Compile Code Server khởi chạy, và chia sẻ chung giữa các container. Đồng thời lúc này, Docker xoá container sau khi tiến trình được chạy xong, giải phóng bộ nhớ cho server.
- Compiler Code Server lấy kết quả biên dịch từ shared volume folder rồi trả kết quả trạng thái và kết quả thực thi code về cho Frontend.

3.3. Luồng hoạt động chi tiết



Hình 26 Luồng hoạt động chi tiết của hệ thống biên dịch và thực thi

1. Compile Code server căn cứ vào tham số ngôn ngữ được truyền vào từ API để tạo container biên dịch:
    - Java: Container biên dịch sẽ được khởi tạo với tiến trình khởi động (entrypoint process) là `javac ${filename}`, trong đó biến `filename` là tên của file mã nguồn.
    - C: Container biên dịch sẽ được khởi tạo với tiến trình khởi động là `gcc ${filename}`.
    - C++: Container biên dịch sẽ được khởi tạo với tiến trình khởi động là `g++ ${filename}`.
    - Python: Do python được chạy bằng trình thông dịch python, các dòng code được dịch thẳng ra mã máy và thực thi nên bước thông dịch này được bỏ qua.

2. Container biên dịch chạy tiến trình với trình biên dịch tương ứng để tạo ra file thực thi (execute file). File này được đưa vào Shared Volume folder. Nếu như trong quá trình biên dịch có lỗi, server sẽ trả về API với trạng thái Compilation Error. Sau khi tiến trình biên dịch hoàn thành, Docker xoá container biên dịch để giải phóng bộ nhớ RAM.

3. Sau khi quá trình biên dịch đã hoàn tất, Compile Code Server sẽ tạo các container biên dịch sao cho phù hợp với từng ngôn ngữ:
   - Java: Container thực thi sẽ được khởi tạo với tiến trình khởi động là `java ./main.java`.
   - C: Container biên dịch sẽ được khởi tạo với tiến trình khởi động là `./exec`.
   - C++: Container thực thi sẽ được khởi tạo với tiến trình khởi động là `./exec`.
   - Python: Container thực thi sẽ được khởi tạo với tiến trình khởi động là `python3 ${filename}`.

4. Container thực thi test case 1 chạy tiến trình với ngôn ngữ tương ứng.

5. Container thực thi test case 1 trả về kết quả thực thi qua Shared Volume folder. Nếu trạng thái của test case 1 khác với Accepted, Compile Code Server API sẽ trả về trạng thái lỗi tương ứng còn lại (Wrong Answer , Compilation Error, Out Of Memory, Time Limit Exceeded hoặc Runtime Error). Nếu trạng thái của test case 1 là Accepted, Compile Code Server sẽ tiếp tục tạo container thực thi test case 2. Sau khi tiến trình thực thi hoàn thành, Docker xoá container biên dịch để giải phóng bộ nhớ RAM.

6. Container thực thi test case 2 chạy tiến trình với ngôn ngữ tương ứng.

7. Container thực thi test case 2 trả về kết quả thực thi qua Shared Volume folder. Nếu trạng thái của test case 1 khác với Accepted, Compile Code Server API sẽ trả về trạng thái lỗi tương ứng còn lại. Nếu trạng thái của test case 1 là Accepted,

Compile Code Server sẽ tiếp tục tạo container thực thi test case 3. Sau khi tiến trình thực thi hoàn thành, Docker sẽ xoá container biên dịch để giải phóng bộ nhớ RAM.

8. Container thực thi test case 3 chạy tiến trình với ngôn ngữ tương ứng.

9. Container thực thi test case 3 trả về kết quả thực thi qua Shared Volume folder. Nếu trạng thái của test case 1 khác với Accepted, Compile Code Server API sẽ trả về trạng thái lỗi tương ứng còn lại. Nếu trạng thái của test case 1 là Accepted, Compile Code Server sẽ tiếp tục tạo container thực thi test case 3. Sau khi tiến trình thực thi hoàn thành, Docker sẽ xoá container biên dịch để giải phóng bộ nhớ RAM.

10. Kết quả thực thi của các test case được lấy từ Shared Volume folder bởi Compile Code Server.

3.4. Cấu hình tiến trình đầu vào của container thực thi

```
#!/usr/bin/env bash

ulimit -s [(${compiler.memoryLimit})]
timeout -s SIGTERM [(${compiler.timeLimit})]
[(${compiler.executionCommand})]
exit $?
```

Hình 27 Cấu hình file bash cho tiến trình đầu vào của container thực thi

Tiến trình đầu vào của container thực thi được cấu hình qua file bash của hệ điều hành Linux, với các câu lệnh:

- *ulimit*: Mang nghĩa “user limits” (Giới hạn cho người dùng). Câu lệnh này cho phép quản trị viên (administrators) quản lý và kiểm soát giới hạn tài nguyên cho các tiến trình của người dùng, cụ thể ở đây là tiến trình của container thực thi. Chúng ta sẽ truyền vào giá trị *compiler.memoryLimit* (giới hạn tài nguyên bộ nhớ RAM được cấu hình trong server biên dịch) để giới hạn tài nguyên cho tiến trình hiện tại.

- *timeout*: Câu lệnh này kiểm soát giới hạn thời gian cho tiến trình của container thực thi. Chúng ta sẽ truyền vào giá trị *compiler.timeLimit* (giới hạn thời gian thực thi được cấu hình trong server biên dịch) để giới hạn thời gian thực thi cho tiến trình hiện tại. Ngoài ra, cấu hình *-s SIGTERM* để thông báo tín hiệu ngắt cho tiến trình khi vượt quá thời gian giới hạn.
- *exit*: Trả về trạng thái khi kết thúc tiến trình. Server biên dịch sẽ lấy mã trạng thái (status code) *$?* để chuyển đổi sang trạng thái biên dịch. Các trạng thái của tiến trình có thể trả về:
+ Exit status code 0: Khi trạng thái kết thúc tiến trình bằng 0 có nghĩa là tiến trình được chạy thành công mà không có lỗi. Do đó, trạng thái của tương ứng của server biên dịch có thể là ACCEPTED hoặc WRONG_ANSWER, làm rõ trạng thái này bằng cách so sánh đầu ra của người dùng và đầu ra mong đợi của bộ testcase.
+ Exit status code 139: Khi trạng kết thúc thái tiến trình bằng 139 có nghĩa là tiến trình đã bị kết thúc do lỗi phân đoạn (segmentation fault). Lỗi phân đoạn này xảy ra khi một chương trình cố gắng truy cập một vùng nhớ bị giới hạn, hay tiến trình đã sử dụng quá lượng tài nguyên RAM cho phép. Trong trường hợp này, trạng thái tương ứng của server biên dịch sẽ là OUT_OF_MEMORY.
+ Exit status code 124: Khi trạng kết thúc thái tiến trình bằng 139 có nghĩa là tiến trình đã bị kết thúc do lỗi timeout. Trạng thái tiến trình này thường được trả về từ câu lệnh *timeout*, khi tiến trình đã chạy quá giới hạn thời gian cho phép. Trong trường hợp này, trạng thái tương ứng của server biên dịch sẽ là TIME_LIMIT_EXCEEDED.
+ Exit status code 96: Khi Khi trạng kết thúc thái tiến trình bằng 139 có nghĩa là tiến trình đã bị kết thúc do lỗi không xác định. Trong trường hợp này, trạng thái tương ứng của server biên dịch sẽ là COMPILATION_ERROR.

3.5. Xây dựng API biên dịch và thực thi

Chúng ta cần một API biên dịch thực thi, là phương tiện để frontend có thể giao tiếp được với server biên dịch code sau khi người dùng submit code lên hệ thống. Để thuận tiện cho việc xử lý request và response API, API sử dụng kiểu dữ liệu application/json.

Endpoint: `/api/compile/json`

Method: `POST`

Request Headers: Content-Type: `application/json`

Request Body:

Trường	Kiểu dữ liệu	Mô tả
sourcecode	String	Mã nguồn của người dùng
language	String	Ngôn ngữ của mã nguồn: Java, C, C++, Python
timelimit	Number	Giới hạn thời gian (s)
testCases	Object	Các test case
testCases.n	Object	Test case thứ n (n = 1, 2, 3, …)
testCases.n.input	String	Đầu vào của test case thứ n
testCases.n.expectedOutput	String	Kết quả mong đợi
memoryLimit	Number	Giới hạn bộ nhớ (MB)

Bảng 1 Bảng mô tả Request body của API biên dịch và thực thi

Ví dụ của Request Body:

```
{
     "sourcecode": "print(\"Hello World Timmy!\")",
     "language": "PYTHON",
     "timeLimit": 10,
     "testCases": {
          "1": {
               "input": "1",
               "expectedOutput": "Hello World Timmy!"
          },
          "2": {
               "input": "5",
               "expectedOutput": "Hello World Timmy!"
          }
     },
     "memoryLimit": 100
}
```

Response Body:

Trường	Kiểu dữ liệu	Mô tả
verdict	String	Trạng thái của submission: Accepted: Kết quả đúng Wrong Answer: Kết quả sai Compilation Error: Lỗi biên dịch Out of Memory: Hết bộ nhớ Time Limit Exceeded: Quá thời gian Runtime Error: Lỗi thực thi
statusCode	Number	Mã trạng thái của submission: Accepted: 100 Wrong Answer: 200 Compilation Error: 300 Out of Memory: 400 Time Limit Exceeded: 500 Runtime Error: 600
error	String	Chi tiết lỗi thực thi
testCasesResult	Object	Kết quả các test case
testCasesResult.n	Object	Kết quả test case thứ n (n = 1, 2, 3, …)
testCasesResult.n. verdict	String	Trạng thái thực thi của test case thứ n
testCasesResult.n. verdictStatusCode	String	Mã trạng thái thực thi của test case thứ n
testCasesResult.n. output	String	Kết quả thực thi của test case thứ n
testCasesResult.n. expectedOutput	String	Kết quả thực thi
testCasesResult.n. error	String	Chi tiết lỗi thực thi
testCasesResult.n. executionDuration	Number	Tổng thời gian thực thi (ms)
compilationDuration	Number	Thời gian biên dịch (ms) (Với ngôn ngữ Python, thời gian biên dịch = 0)
averageExecutionDuration	Number	Thời gian biên dịch trung bình của các

		test case
timeLimit	Number	Giới hạn thời gian (s)
memoryLimit	Number	Giới hạn bộ nhớ (MB)
language	Number	Ngôn ngữ của mã nguồn
dateTime	Date	Thời gian thực thi

Bảng 2 Bảng mô tả Response Body của API biên dịch và thực thi

Ví dụ của Response Body:


```
{
    "verdict": "Accepted",
    "statusCode": 100,
    "error": "",
    "testCasesResult": {
        "1": {
            "verdict": "Accepted",
            "verdictStatusCode": 100,
            "output": "Hello World Timmy!\n",
            "error": "",
            "expectedOutput": "Hello World Timmy!",
            "executionDuration": 233
        },
        "2": {
            "verdict": "Accepted",
            "verdictStatusCode": 100,
            "output": "Hello World Timmy!\n",
            "error": "",
            "expectedOutput": "Hello World Timmy!",
            "executionDuration": 202
        }
    },
    "compilationDuration": 0,
    "averageExecutionDuration": 217.5,
    "timeLimit": 10,
    "memoryLimit": 100,
    "language": "PYTHON",
    "dateTime": "2023-12-23T03:23:47.895633"
}
```

CHƯƠNG IV: CÀI ĐẶT HỆ THỐNG WEBSITE HỌC TẬP LẬP TRÌNH

4.1. Yêu cầu hệ thống

- Hệ điều hành Ubuntu 20.04 trở lên
- Node 18.15
- ReactJS 18.12
- MongoDB 7.0
- Java 11
- Spring Boot 2.3.x

4.2. Cài đặt hệ thống biên dịch và thực thi

Bước 1: Cài Docker qua các lệnh:

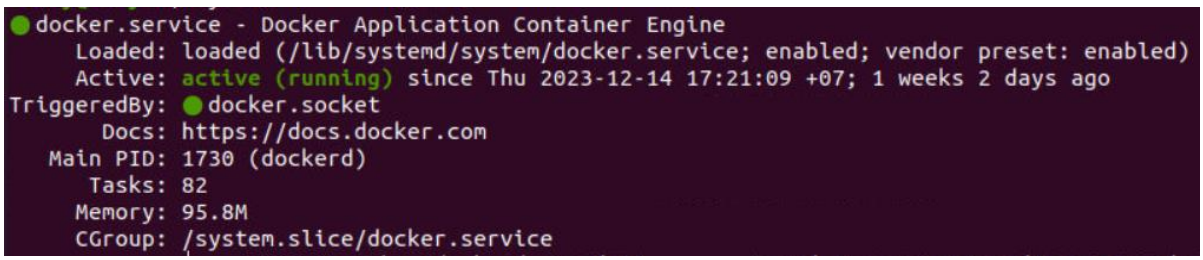
```
sudo apt update
sudo apt install apt-transport-https ca-certificates curl
software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo apt-key add -

sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu focal stable"
```

Hình 28 Các lệnh cài đặt Docker

Kiểm tra trạng thái hoạt động của Docker qua lệnh: `service docker status`



Hình 29 Dịch vụ Docker đang hoạt động

Bước 2: Tạo các file *Dockerfile.lang.compilation* cấu hình trình biên dịch cho các container biên dịch:

- Với ngôn ngữ Java:

```
FROM openjdk:11.0.6-jdk-slim

WORKDIR /app

ENTRYPOINT ["/bin/sh", "-c", "javac -d $EXECUTION_PATH
$EXECUTION_PATH/$SOURCE_CODE_FILE_NAME && rm
$EXECUTION_PATH/$SOURCE_CODE_FILE_NAME"]
```

Hình 30 Cấu hình Dockerfile cho container biên dịch Java

- Với ngôn ngữ C:

```
FROM gcc

WORKDIR /app

ENTRYPOINT ["/bin/sh", "-c", "gcc
$EXECUTION_PATH/$SOURCE_CODE_FILE_NAME -o
$EXECUTION_PATH/exec && rm
$EXECUTION_PATH/$SOURCE_CODE_FILE_NAME"]
```

Hình 31 Cấu hình Dockerfile cho container biên dịch C

- Với ngôn ngữ C++:

```
FROM gcc

WORKDIR /app

ENTRYPOINT ["/bin/sh", "-c", "g++
$EXECUTION_PATH/$SOURCE_CODE_FILE_NAME -o
$EXECUTION_PATH/exec && rm
$EXECUTION_PATH/$SOURCE_CODE_FILE_NAME"]
```

Hình 32 Cấu hình Dockerfile cho container biên dịch C++

Lưu ý: Với bước này, chúng ta không cần tạo Dockerfile cho ngôn ngữ Python vì Python dùng trình thông dịch (interpreter) để dịch thành mã máy.

Bước 3: Tạo file Docker.execution cấu hình môi trường thực thi:

```
FROM language_image

WORKDIR /app

USER root

RUN groupadd -r user -g 111 && \
    useradd -u 111 -r -g user -s /sbin/nologin -c "Docker
image user" user

ADD . .

RUN chmod a+x entrypoint-*.sh

USER user

ENTRYPOINT ["/bin/sh", "-c", "./entrypoint-$TEST_CASE_ID.sh"]
```

Hình 33 Cấu hình Dockerfile cho container thực thi

Tạo file *entrypoint-$TEST_CASE_ID.sh* cấu hình tiến trình khởi đầu cho các container thực thi test case:

```
#!/usr/bin/env bash

ulimit -s [(${compiler.memoryLimit})]
timeout -s SIGTERM [(${compiler.timeLimit})]
[(${compiler.executionCommand})]
exit $?
```

File này được viết bằng ngôn ngữ Bash của Linux, cho phép chúng ta giới hạn được bộ nhớ RAM và thời gian chạy của tiến trình.

Bước 4: Tạo file Dockerfile cho server biên dịch code để deploy server lên Docker:

```
# Build stage
FROM maven:3.6.0 AS BUILD_STAGE
WORKDIR /compiler
COPY . .
RUN ["mvn", "clean", "install", "-Dmaven.test.skip=true"]

# Run stage
FROM openjdk:11.0.6-jre-slim
WORKDIR /compiler

USER root

COPY --from=BUILD_STAGE /compiler/target/*.jar
../compiler.jar

RUN apt update && apt install -y docker.io

ADD executions ../executions

ADD entrypoint.sh ../entrypoint.sh

RUN chmod a+x ../entrypoint.sh

EXPOSE 8082

ENTRYPOINT ["../entrypoint.sh"]
```

Hình 34 Cấu hình Dockerfile cho server biên dịch code

Ở bước này, chúng ta đã cấu hình để thêm các file được tạo ở bước 2 và 3 qua câu lệnh ADD.

Bước 5: Cấu hình giới hạn tài nguyên cho server biên dịch code:

```
compiler:
  max-test-cases: ${MAX_TEST_CASES:20} # maximum number of test cases a request should handle
  compilation-container:
    volume: ${COMPILATION_CONTAINER_VOLUME:} # only when running the app inside a container
  execution:
    max-cpus: ${MAX_EXECUTION_CPUS:0}
  docker:
    image:
      delete: ${DELETE_DOCKER_IMAGE:true} # delete the docker image at the end of the execution of the container, by default it is set to true.
  execution-memory:
    max: ${EXECUTION_MEMORY_MAX:100}
    min: ${EXECUTION_MEMORY_MIN:0}
  execution-time:
    max: ${EXECUTION_TIME_MAX:10}
    min: ${EXECUTION_TIME_MIN:0}
  max-requests: ${MAX_REQUESTS:1000}
```

Hình 35 Cấu hình giới hạn tài nguyên cho server biên dịch code

Các cấu hình giới hạn tài nguyên của server biên dịch code:

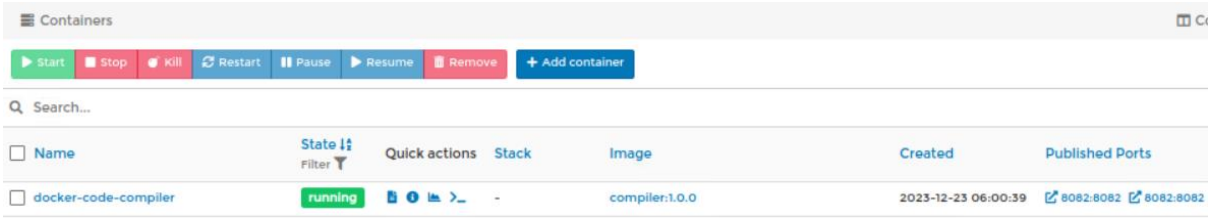
- MAX_TEST_CASES: Giới hạn test case trong một request
- MAX_EXECUTION_CPUS: Giới hạn số core CPU biên dịch và thực thi
- EXECUTION_MEMORY_MAX: Giới hạn bộ nhớ RAM cho một request
- EXECUTION_TIME_MAX: Giới hạn thời gian thực thi cho một request
- MAX_REQUESTS: Giới hạn số lượng request trong một phút

Bước 6: Deploy server biên dịch code lên Docker qua lệnh:

```
docker container run -p 8082:8082 compiler:/compiler
COMPILATION_CONTAINER_VOLUME=compiler -t compiler:1.0.0
```

Hình 36 Câu lệnh deploy server biên dịch lên Docker

Server đã được deploy lên Docker với phiên bản 1.0.0, và chạy qua port 8082, sẵn sàng nhận các API request từ frontend.



Hình 37 Server biên dịch được deploy lên Docker

4.3. Cài đặt website học tập lập trình

Bước 1: Crawl (cào dữ liệu) câu hỏi từ Leetcode

LeetCode là một nền tảng trực tuyến giúp rèn luyện kỹ năng lập trình và giải quyết bài tập cấu trúc dữ liệu và giải thuật. Nó cung cấp khoảng 2000 bài tập lập trình đa dạng, phân loại theo chủ đề và độ khó. Các bài tập này có thể được crawl từ API GraphQL public của Leetcode cung cấp.

```
data = {
    'operationName': 'problemsetQuestionList',
    'query': 'query problemsetQuestionList($categorySlug:
String, $limit: Int, $skip: Int, $filters:
QuestionListFilterInput) {\n  problemsetQuestionList:
questionList(\n    categorySlug: $categorySlug\n    limit:
$limit\n    skip: $skip\n    filters: $filters\n  ) {\n
total: totalNum\n    questions: data {\n      acRate\n
```

```
difficulty\n      freqBar\n      frontendQuestionId:
questionFrontendId\n      isFavor\n      paidOnly:
isPaidOnly\n      status\n      title\n      titleSlug\n
topicTags {\n        name\n        id\n        slug\n
}\n      hasSolution\n      hasVideoSolution\n    }\n  }\n}\n
',
    'variables': {'categorySlug': '', 'skip': 0, 'limit':
100000, 'filters': {}}
}

r = requests.post('https://leetcode.com/graphql', json =
data)
response_body = r.json()

csv_columns = list(leetcode_problemsetQuestionList[0].keys())
csv_file = 'leetcode_all_questions.csv'
try:
    with open(csv_file, 'w') as csvfile:
        writer = csv.DictWriter(csvfile,
fieldnames=csv_columns)
        writer.writeheader()
        for data in leetcode_problemsetQuestionList:
            writer.writerow(data)
except IOError:
    print("I/O error")
```

Hình 38 Crawl dữ liệu của tất cả các câu hỏi từ Leetcode

Các raw data (dữ liệu thô) của tất cả các câu hỏi được lưu vào file leetcode_all_questions.csv. Tiếp theo, chúng ta sẽ crawl mô tả chi tiết của từng câu hỏi.

```
data =
{"operationName":"questionData","variables":{"titleSlug":"add
-two-numbers"},"query":"query questionData($titleSlug:
String!) {\n  question(titleSlug: $titleSlug) {\n
questionId\n    questionFrontendId\n    boundTopicId\n
title\n    titleSlug\n    content\n    translatedTitle\n
translatedContent\n    isPaidOnly\n    difficulty\n
likes\n    dislikes\n    isLiked\n    similarQuestions\n
```

```
contributors {\n      username\n      profileUrl\n
avatarUrl\n      __typename\n    }\n
langToValidPlayground\n    topicTags {\n      name\n
slug\n      translatedName\n      __typename\n    }\n
companyTagStats\n    codeSnippets {\n      lang\n
langSlug\n      code\n      __typename\n    }\n    stats\n
hints\n    solution {\n      id\n      canSeeDetail\n
__typename\n    }\n    status\n    sampleTestCase\n
metaData\n    judgerAvailable\n    judgeType\n
mysqlSchemas\n    enableRunCode\n    enableTestMode\n
envInfo\n    libraryUrl\n    __typename\n  }\n}\n"}

r = requests.post('https://leetcode.com/graphql', json =
data)

leetcode_specific_question_infor = []
for i, title in enumerate(all_title_slug):
    while True:
        try:
            r = requests.post('https://leetcode.com/graphql',
json = data)
            if r.status_code == 200:
                print('Crawling question', i, 'done')

leetcode_specific_question_infor.append(r.json()['data']['que
stion'])
                break
        except:
            print('Something wrong with question', i, ',
trying again')

csv_columns = ['questionId', 'questionFrontendId',
'boundTopicId', 'title', 'titleSlug', 'content',
'translatedTitle', 'translatedContent', 'isPaidOnly',
'difficulty', 'likes', 'dislikes', 'isLiked',
'similarQuestions', 'contributors', 'langToValidPlayground',
'topicTags', 'companyTagStats', 'codeSnippets', 'stats',
'hints', 'solution', 'status', 'sampleTestCase', 'metaData',
'judgerAvailable', 'judgeType', 'mysqlSchemas',
'enableRunCode', 'enableTestMode', 'envInfo', 'libraryUrl',
```

```
'__typename']

csv_file = 'leetcode_specific_question_infor.csv'

try:
    with open(csv_file, 'w') as csvfile:
        writer = csv.DictWriter(csvfile,
fieldnames=csv_columns)
        writer.writeheader()
        for data in leetcode_specific_question_infor:
            writer.writerow(data)
except IOError:
    print("I/O error")
```

Hình 39 Crawl dữ liệu mô tả chi tiết của tất cả các câu hỏi từ Leetcode

Các raw data (dữ liệu thô) mô tả chi tiết của tất cả các câu hỏi được lưu vào file leetcode_specific_question_info.csv.

Bước 2: Xử lý raw data từ bộ câu hỏi của Leetcode và thêm dữ liệu vào cơ sở dữ liệu MongoDB

Với raw data từ file leetcode_all_questions.csv, chúng ta sẽ xử lý và thêm vào cơ sở dữ liệu với các trường:

+ id: Id của câu hỏi
+ name: Tên đầy đủ của câu hỏi
+ name_slug: Tên đầy đủ của câu hỏi dưới dạng slug (Ví dụ: two-sum)
+ difficulty: Độ khó của câu hỏi
+ like_count: Số lượng like của câu hỏi
+ dislike_count: Số lượng dislike của câu hỏi
+ acceptance_rate_count: Tỉ lệ giải thành công câu hỏi
+ topic_tags: Các chủ đề liên quan của câu hỏi
+ status: Trạng thái xử lý câu hỏi của người dùng

```
client =
MongoClient("mongodb://admin:admin@localhost:37017/?serverSel
ectionTimeoutMS=5000&connectTimeoutMS=10000&authSource=admin&
authMechanism=SCRAM-SHA-1")
db = client["code-programming"]
collection = db["problems"]
```

```
csv_file_path = "leetcode_all_questions.csv"
df = pd.read_csv(csv_file_path)

for _, row in df.iterrows():
    document = {
        "_id": str(ObjectId()),
        "main": {
            "id": row["frontendQuestionId"],
            'name': row['title'],
            "name_slug": row["titleSlug"],
            "difficulty": row["difficulty"].lower(),
            "like_count": row["likes"],
            "dislike_count": row["dislikes"],
            "acceptance_rate_count": row["acRate"],
            'topic_tags': row['topicTags'].replace('None',
'\'None\'').replace('\'', '\"'),
            "status": "" if row["status"] else "unattempted",
        }
    }
    collection.insert_one(document)

client.close()
```

Hình 40 Xử lý và thêm dữ liệu của tất cả các câu hỏi vào MongoDB

Với raw data từ file leetcode_specific_question_info.csv, chúng ta sẽ xử lý và thêm vào cơ sở dữ liệu với các trường:

+ id: Id của câu hỏi
+ name_slug: Tên đầy đủ của câu hỏi dưới dạng slug (Ví dụ: two-sum)
+ description_body: Mô tả của câu hỏi
+ similar_questions: Các câu hỏi tương tự
+ topic_tags: Các chủ đề liên quan của câu hỏi
+ status: Trạng thái xử lý câu hỏi của người dùng
+ input: Đầu vào của test case
+ output: Đầu ra mong đợi của test case
+ hints: Các gợi ý cho câu hỏi

```
csv_file = 'leetcode_specific_question_infor.csv'
df = pd.read_csv(csv_file)
```

```
client = MongoClient(

'mongodb://admin:admin@localhost:37017/?serverSelectionTimeou
tMS=5000&connectTimeoutMS=10000&authSource=admin&authMechanis
m=SCRAM-SHA-1')
db = client['code-programming']
collection = db['spec_problems']

for index, row in df.iterrows():
    if not isinstance(row['content'], str) or not
row['content']:
        continue

    main_data = {
        'id': int(row['questionId']),
        'name_slug': row['titleSlug'],
        'description_body': row['content'],
        'similar_questions':
row['similarQuestions'].replace('\'', '\"'),
    }

    test_data = []

    content_lines = str(row['content'])

    res =
re.findall(r'<strong>Input:<\/strong>\s*(.*?)\s*<strong>Outpu
t:<\/strong>\s*(.*?)\s*(<strong>|<\/pre>)',
                       content_lines)

    for tup in res:
        test_data.append({
            'input': tup[0].replace('&quot;', '\"'),
            'output': tup[1].replace('&quot;', '\"'),
        })

    document = {
        'main': main_data,
        'hints': row['hints'].replace('it\'s', 'it
is').replace('It\'s', 'It is').replace('\'', '\"'),
```

```
        'test': test_data,
    }

    collection.insert_one(document)

client.close()
```

Hình 41 Xử lý và thêm dữ liệu mô tả chi tiết của tất cả các câu hỏi vào MongoDB

Bước 3: Tạo file Dockerfile cho backend và frontend của website học tập lập trình để deploy website lên Docker:

```
FROM node:18.19.0

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

RUN cp .env.prod .env

EXPOSE 8085

CMD ["npm", "start"]
```

Hình 42 Cấu hình Dockerfile cho backend website học tập lập trình

```
FROM node:18.19.0

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install
```

```
COPY . .

RUN npm run build

RUN cp .env.prod .env

EXPOSE 3000

CMD ["npm", "start"]
```

Hình 43 Cấu hình Dockerfile cho frontend website học tập lập trình

Bước 3: Deploy hệ thống website học tập lập trình lên Docker qua lệnh:

```
docker container run -p 8085:8085 --name compro-server -t
compro-server:1.0.0
docker container run -p 3000:3000 --name compro-client -t
compro-client:1.0.0
```

Hình 44 Các lệnh deploy website học tập lập trình lên Docker

Server đã được deploy lên Docker với phiên bản 1.0.0, backend chạy qua port 8085 và frontend chạy qua port, sẵn sàng được sử dụng thực tế.

KẾT LUẬN

Kết quả đạt được

Sau thời gian học tập, nghiên cứu và phát triển hệ thống, đồ án của em về cơ bản đã đạt được các mục tiêu ban đầu đặt ra. Đồ án đã đưa ra được cái giải pháp mới rồi đi sâu vào phân tích, thiết kế và cuối cùng đã tạo ra được một website hoàn chỉnh.

Về hệ thống biên dịch và thực thi:

- Hệ thống biên dịch và thực thi hoạt động trơn tru, ổn định, hiệu năng tốt
- Hoàn thiện tốt các chức năng bảo mật cho hệ thống
- API hoạt động, tương tác tốt với frontend

Về mặt giao diện website:

- Màu sắc và thiết kế bắt mắt cho website học tập lập trình
- Bố cục giao diện website cân đối, hài hoà giúp người dùng có thể thao tác dễ dàng

Về chức năng:

- Đáp ứng được các chức năng lớn của website học tập lập trình
- Các chức năng bổ trợ đã hoạt động tốt

Là người phát triển hệ thống, em thấy rằng việc xây dựng website học tập lập trình đã giúp em hiểu được toàn cảnh một hệ thống lớn hoạt động từ bước phát triển tới bước vận hành. Ngoài ra, em còn được nắm rõ hơn cách Docker và hệ điều hành Linux hoạt động, cách xây dựng website bằng ReactJS.

Hạn chế

Đồ án còn một số hạn chế như:

- Do vấn đề về thời gian nên còn nhiều chức năng có thể bổ sung giúp hệ thống có thể hoàn thiện hơn nữa
- Giao diện chưa responsive
- Website thiếu chức năng hướng cộng đồng cho người dùng: chat, thảo luận về bài giải

- Hệ thống biên dịch và thực thi vẫn chưa đo được chính xác lượng tài nguyên tiêu thụ của một tiến trình
- Hệ thống biên dịch còn hạn chế về việc hỗ trợ nhiều ngôn ngữ lập trình

Hướng phát triển sau này

- Bổ sung các tính năng hướng cộng đồng
- Bổ sung thêm các ngôn ngữ lập trình khác cho hệ thống biên dịch và thực thi
- Nâng cấp hiệu năng cho hệ thống biên dịch và thực thi

TÀI LIỆU THAM KHẢO

[1] Prof. Douglas Thain, University of Notre Dame. *"Introduction to Compilers and Language Design*"

[2] Juan C. Rodríguez-del-Pino, Enrique Rubio-Royo, Zenón J. Hernández-Figuero. "*A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features*"

[3] Ian Miell, Aidan Hobson Sayers. "*Docker in practice*"

[4] GeeksForGeeks, "Introduction to Compilers", https://www.geeksforgeeks.org/introduction-to-compilers/

[5] GeeksForGeeks, "History of Compiler", https://www.geeksforgeeks.org/history-of-compiler

[6] RedHat, "Is chroot a security feature?". https://www.redhat.com/en/blog/chroot-security-feature

[7] Medium, "How does the computer understands programming languages", https://medium.com/the-ui-girl/some-computer-science-questions-7c1056909757

[8] Medium, "How we used Docker to compile and run untrusted code", https://blog.remoteinterview.io/how-we-used-docker-to-compile-and-run-untrusted-code-2fafbffe2ad5

[9] Medium, "System Design — Online Coding / Judge Platform", https://medium.com/@jnu_saurav/system-design-online-coding-judge-platform-5b39380818fc