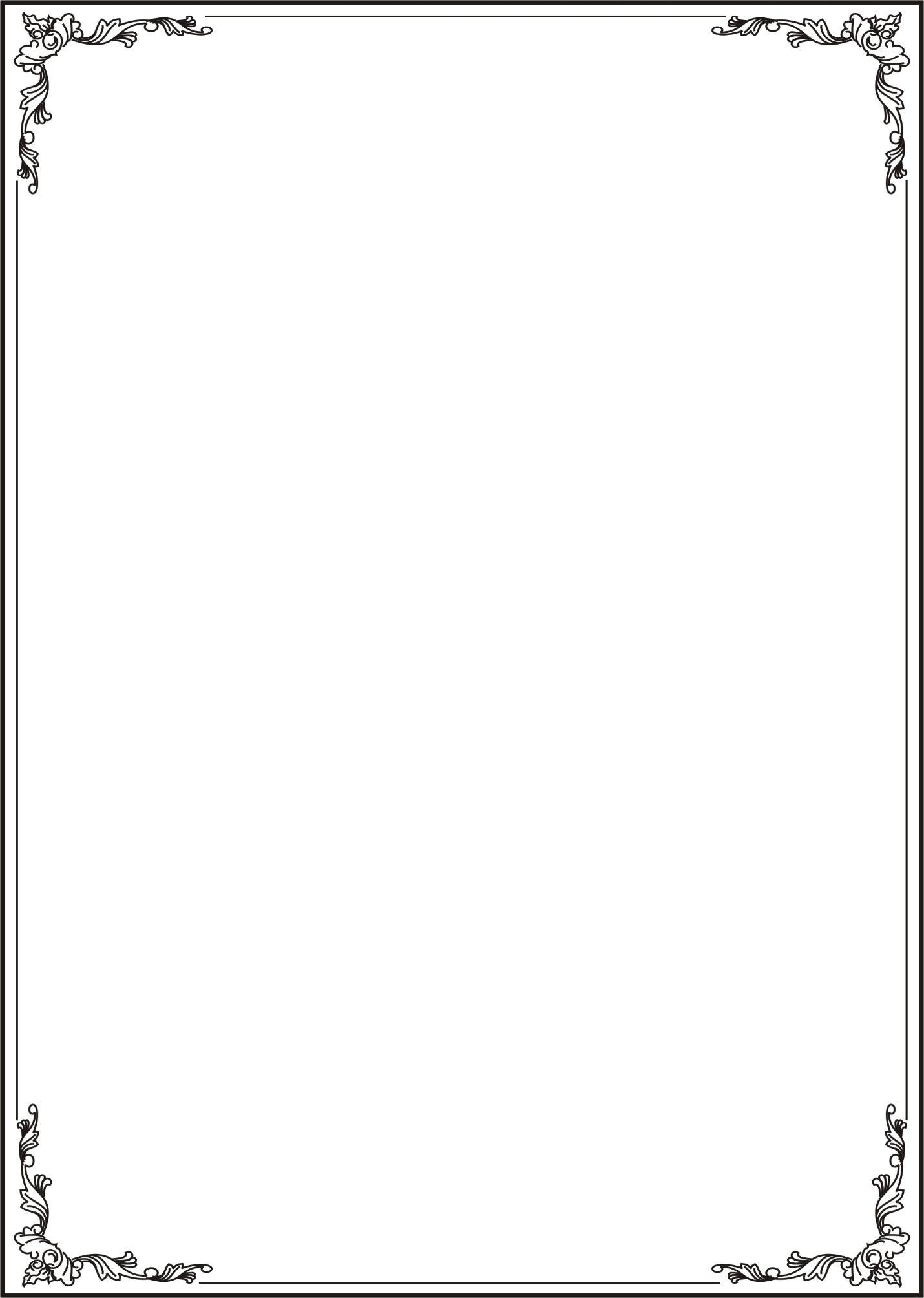
****ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

A blue logo with a black background

Description automatically generated with low confidence

**BÁO CÁO CUỐI KỲ**

**SO SÁNH HIỆU SUẤT GIỮA GIN (GO) VÀ SPRING BOOT (JAVA)**

**SE400.P11.PMCL**

**SEMINAR CÁC VẤN ĐỀ HIỆN ĐẠI CỦA**

**CÔNG NGHỆ PHẦN MỀM**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn:** | **ThS. Đinh Nguyễn Anh Dũng** |
| **Thành viên:** | * **Nguyễn Tiến Vĩ - 21522788** * **Nguyễn Đình Khoa - 21520997** |

Thành phố Hồ Chí Minh, ngày 30 tháng 12, năm 2024

**DANH SÁCH THÀNH VIÊN**

|  |  |  |
| --- | --- | --- |
| **STT** | **Họ và tên** | **Mã số sinh viên** |
| **1** | **Nguyễn Tiến Vĩ** | **21522788** |
| **2** | **Nguyễn Đình Khoa** | **21520997** |

Mục Lục

[1. Tổng quan 5](#_Toc186652314)

[1.1. Tổng quan về đề tài 5](#_Toc186652315)

[1.2. Lý do chọn đề tài 5](#_Toc186652316)

[2. Cơ sở lý thuyết 5](#_Toc186652317)

[2.1. Gin 5](#_Toc186652318)

[2.1.1. Đặc trưng 6](#_Toc186652319)

[2.1.2. Ưu điểm 6](#_Toc186652320)

[2.1.3. Nhược điểm 7](#_Toc186652321)

[2.2. Spring Boot 7](#_Toc186652322)

[2.2.1. Đặc trưng 7](#_Toc186652323)

[2.2.2. Ưu điểm 8](#_Toc186652324)

[2.2.3. Nhược điểm 8](#_Toc186652325)

[2.3. PostgresSQL 9](#_Toc186652326)

[2.3.1. Đặc trưng 9](#_Toc186652327)

[2.3.2. Ưu điểm 9](#_Toc186652328)

[2.3.3. Nhược điểm 10](#_Toc186652329)

[2.4. Docker 10](#_Toc186652330)

[2.4.1. Đặc trưng 10](#_Toc186652331)

[2.4.2. Ưu điểm 11](#_Toc186652332)

[2.4.3. Nhược điểm 11](#_Toc186652333)

[2.5. cAdvisor 11](#_Toc186652334)

[2.5.1. Đặc trưng 12](#_Toc186652335)

[2.5.2. Ưu điểm 12](#_Toc186652336)

[2.5.3. Nhược điểm 12](#_Toc186652337)

[2.6. Prometheus 13](#_Toc186652338)

[2.6.1. Đặc trưng 13](#_Toc186652339)

[2.6.2. Ưu điểm 13](#_Toc186652340)

[2.6.3. Nhược điểm 14](#_Toc186652341)

[2.7. Grafana OSS 14](#_Toc186652342)

[2.7.1. Đặc trưng 14](#_Toc186652343)

[2.7.2. Ưu điểm 15](#_Toc186652344)

[2.7.3. Nhược điểm 15](#_Toc186652345)

[3. Nội dung và phương pháp 15](#_Toc186652346)

[3.1. Nội dung 15](#_Toc186652347)

[3.1.1. Trường hợp hệ thống idle 16](#_Toc186652348)

[3.1.2. Request trả về dữ liệu JSON tĩnh 16](#_Toc186652349)

[3.1.3. Request trả về dữ liệu từ cơ sở dữ liệu 16](#_Toc186652350)

[3.2. Thiết kế, cài đặt hệ thống 18](#_Toc186652351)

[3.2.1. demo-java 18](#_Toc186652352)

[3.2.2. demo-gin 19](#_Toc186652353)

[3.2.3. postgres\_go 19](#_Toc186652354)

[3.2.4. postgres\_java 19](#_Toc186652355)

[3.2.5. prometheus 19](#_Toc186652356)

[3.2.6. grafana 20](#_Toc186652357)

[3.2.7. Client Machine 20](#_Toc186652358)

[4. Hiện trạng 20](#_Toc186652359)

[4.1. Các kết quả đã đạt được 20](#_Toc186652360)

[4.2. Kết quả sau khi thực nghiệm: 21](#_Toc186652361)

[4.2.1. Ở trạng thái idle 21](#_Toc186652362)

[4.2.2. GET 1 static json 22](#_Toc186652363)

[4.2.3. GET 1 order record 23](#_Toc186652364)

[4.2.4. POST create 1 order 24](#_Toc186652365)

[4.2.5. GET all products 25](#_Toc186652366)

[4.2.6. POST chuyển đổi ảnh màu sang đơn sắc 26](#_Toc186652367)

[4.3. Tài liệu kết quả 26](#_Toc186652368)

[5. Kết luận 26](#_Toc186652369)

[5.1. Thuận lợi 27](#_Toc186652370)

[5.2. Khó khăn 27](#_Toc186652371)

[5.2.1. Khó khăn trong việc làm quen với Grafana 27](#_Toc186652372)

[5.2.2. Giới hạn tần suất gửi request 28](#_Toc186652373)

[6. Tài liệu tham khảo 28](#_Toc186652374)

1. **Tổng quan**
   1. **Tổng quan về đề tài**

Đề tài tập trung vào việc so sánh hiệu suất giữa hai Framework phát triển API phổ biến hiện nay là Gin (một Framework của Go) và Spring Boot (một Framework lâu đời và phổ biến của hệ sinh thái Java). Việc so sánh hiệu suất giữa 2 Framework này sẽ giúp xác định các điểm mạnh và yếu của từng Framework trong ứng dụng xử lý API hiệu quả cao, từ đó hỗ trợ quyết định lựa chọn công nghệ phù hợp cho hệ thống.

* 1. **Lý do chọn đề tài**

- **Độ phổ biến của Framework:** Gin và Spring Boot là hai trong số những lựa chọn hàng đầu hiện nay khi triển khai các ứng dụng backend. Gin nổi bật vì sự tối giản, trong khi vẫn đảm bảo tốc độ nhanh nhờ cách tiếp cận hướng hiệu suất của Go. Spring Boot thì mạnh mẽ nhờ các tính năng và số lượng thư viện phong phú của Java với lịch sử phát triển lâu đời. So sánh này giúp làm rõ các lợi ích cũng như hạn chế của mỗi Framework trong bối cảnh yêu cầu thực tế.

**- Khả năng tối ưu hóa hiệu suất:** Trong bối cảnh hiện đại ngày nay, phần lớn các hệ thống API đều cần xử lý số lượng request lớn và đảm bảo tốc độ phản hồi nhanh. Gin và Spring Boot đều có khả năng xây dựng API hiệu quả, tuy nhiên mỗi Framework có cách quản lý tài nguyên, tối ưu hóa bộ nhớ và xử lý các tác vụ bất đồng bộ khác biệt. Việc kiểm tra hiệu suất sẽ cho góc nhìn rõ ràng hơn về hiệu quả và khả năng đáp ứng của mỗi Framework.

1. **Cơ sở lý thuyết**
   1. **Gin**

Gin là một Web Framework được viết bằng ngôn ngữ lập trình Go (Golang). Nó hỗ trợ API tương tự Martini (một Framework khác dựa trên Go nhưng đã bị ngừng phát triển) với hiệu suất được cải thiện (lên đến 40 lần) nhờ httprouter (một thư viện router cho Go, hỗ trợ việc định tuyến). Framework này được xây dựng với triết lý tối giản, giúp phát triển các ứng dụng API nhanh chóng và tiết kiệm tài nguyên.

* + 1. **Đặc trưng**

**- Nhanh:** Gin sử dụng cấu trúc Radix tree để xử lý routing, tiêu thụ ít bộ nhớ, không sử dụng reflection.

**-** **Hỗ trợ Middleware:** Một request HTTP được xử lý qua chuỗi các Middleware.

**- Không crash:** Gin có thể bắt các lỗi xảy ra trong quá trình xử lý HTTP và phục hồi lại hệ thống. Nhờ vậy, server sẽ luôn sẵn sàng.

**- Xác thực JSON:** Gin có thể phân tích và xác thực JSON của request. Ví dụ: kiểm tra sự tồn tại của các giá trị bắt buộc.

**- Nhóm các route:** Phân chia các route cần và không cần ủy quyền, các phiên bản API khác nhau... Ngoài ra, các nhóm này có thể được lồng vào nhau không giới hạn nhưng không làm ảnh hưởng hiệu suất.

**- Quản lý lỗi:** Gin cung cấp các cách tiện lợi để thu thập tất cả các lỗi xảy ra trong quá trình xử lý một request HTTP.

**- Tích hợp sẵn các format rendering:** Gin cung cấp API dễ sử dụng cho việc render JSON, XML hay HTML...

**- Có tính mở rộng:** Dễ dàng tạo Middleware mới.

* + 1. **Ưu điểm**

**- Hiệu suất cao:** Gin được xây dựng với ưu tiên về tốc độ. Gin phù hợp xử lý lượng truy cập lớn cho các hệ thống backend và API.

**-** **Tối giản và gọn nhẹ**: Triết lý thiết kế của Gin là giữ mọi thứ đơn giản và tinh gọn. Nó sử dụng ít bộ nhớ và không sử dụng các lớp trừu tượng thừa thãi, giúp các lập trình viên dễ dàng làm quen.

**- Định tuyến nhanh chóng:** Router của Gin được tối ưu và có thể xử lý các tác vụ định tuyến nhanh chóng và hiệu quả kể cả với những yêu cầu phức tạp.

**- Hỗ trợ Middleware:** Gin cung cấp một hệ thống Middleware phong phú, cho phép các lập trình viên mở rộng hệ thống với các chức năng như authentication, logging, rate-limiting... dưới dạng các module.

**- Cộng đồng người dùng sôi nổi:** Gin có một cộng đồng người dùng lớn và sôi động đang đóng góp cho sự phát triển của Framework và hỗ trợ lẫn nhau thông qua các diễn đàn và đóng góp vào repository open-source của Gin.

* + 1. **Nhược điểm**

**- Không tích hợp sẵn các tính năng thông dụng:** Vì Gin là một Framework hướng đến sự gọn gàng, tinh giản nên nó thiếu đi một vài các tính năng phổ biến với các Framework khác như: ORM (Object Relational Mapping), quản lý cookie và session... Tuy nhiên điều này cũng cho phép các lập trình viên được linh hoạt trong việc lựa chọn các thực hiện các tính năng đó (sử dụng thư viện hoặc tự phát triển).

**- Cộng đồng người dùng nhỏ hơn các Framework lâu đời khác:** Tuy có số lượng người dùng lớn cũng như cộng đồng sôi nổi nhưng vì ra đời sau các Framework lớn như Spring Boot, Node.js, .NET... việc tìm hướng dẫn sử dụng cũng như trợ giúp khi sử dụng Gin có thể khó khăn hơn.

* 1. **Spring Boot**

Spring Boot là một Framework dựa trên Java, phát triển từ hệ sinh thái Spring. Nó cung cấp tất cả các tính năng của Spring nhưng dễ sử dụng hơn. Spring Boot là một Framework dựa trên Microservice, cho phép xây dựng ứng dụng trong thời gian ngắn hơn nhiều so với Spring vì gần như mọi thứ đều được tự động cấu hình.

* + 1. **Đặc trưng**

- Tạo các ứng dụng Spring độc lập.

- Tích hợp trực tiếp Tomcat, Jetty hoặc Undertow (không cần triển khai các file WAR).

- Cung cấp các dependency "starter" có tính opinionated để đơn giản hóa việc cấu hình.

- Tự động cấu hình Spring và các thư viện bên thứ ba khi có thể.

- Cung cấp các tính năng sẵn sàng cho sản xuất như đo lường, kiểm tra tình trạng hệ thống và cấu hình bên ngoài.

- Không tạo code tự động và không yêu cầu cấu hình XML.

* + 1. **Ưu điểm**

**-** Phát triển ứng dụng nhanh chóng sử dụng Spring.

- Tự động cấu hình tất cả các thành phần cho một ứng dụng Spring sẵn sàng cho production.

- Máy chủ tích hợp sẵn (Tomcat, Jetty, Undertow) giúp triển khai ứng dụng nhanh chóng và hiệu quả hơn.

- Không cần cấu hình XML.

- Hệ thống Plugin phong phú.

- Truy cập dễ dàng vào các cơ sở dữ liệu như MySQL, Oracle, MongoDB, Redis, ActiveMQ...

- Tích hợp dễ dàng với hệ sinh thái Spring.

- Cộng đồng lớn và lâu đời.

* + 1. **Nhược điểm**

- **Thiếu sự kiểm soát:** Spring Boot tạo rất nhiều dependencies, kể cả khi chúng không được sử dụng, dẫn đến kích thước file deploy lớn.

- **Thiếu sự linh hoạt:** Spring Boot sử dụng hướng tiếp cận opinionated. Điều này có nghĩa là nó thường hướng đến một giải pháp duy nhất cho mỗi vấn đề. Nó giúp lập trình viên đưa ra quyết định dễ dàng nhưng cũng làm nó kém linh hoạt hơn so với Spring.

- **Kích thước ưng dụng lớn:** Spring Boot thường tạo ra các file JAR/WAR với kích thước lớn do tích hợp sẵn các thư viện và server nhúng.

- **Khởi động chậm:** Spring Boot có tốc độ khơi động khá chậm so với các framework khác, bên cạnh đó các request khi server mới khởi chạy được lý chậm vì JIT chưa được tối ưu.

* 1. **PostgresSQL**

PostgresSQL là một cơ sở dữ liệu quan hệ mạnh mẽ, mã nguồn mở với hơn 35 năm lịch sử phát triển, cùng danh tiếng về sự tin cậy, tính năng phong phú và hiệu suất cao.

* + 1. **Đặc trưng**

**- Hệ quản trị cơ sở dữ liệu quan hệ.**

**- Tuân thủ ACID:** Postgres tuân thủ 4 đặc tính ACID để đảm bảo các transaction trên cơ sở dữ liệu được xử lý một cách đáng tin cậy.

**- Tính mở rộng:** Cơ sở dữ liệu hỗ trợ nhiều kiểu dữ liệu và có thể dễ dàng được mở rộng với các hàm, toán tử...

**- Tuân thủ chuẩn SQL:** PostgresSQL tuân theo tiêu chuẩn của SQL, giúp việc sử dụng cơ sở dữ liệu trở nên dễ dàng cho người dùng đã có kinh nghiệm với SQL.

**- Kiểm soát Concurrency:** Postgres hỗ trợ multi-version concurrency control (MVCC), cho phép truy cập đồng thời vào cùng một dữ liệu bởi nhiều transaction mà không gây xung đột

* + 1. **Ưu điểm**

**- Hiệu quả về chi phí:** PostgreSQL là phần mềm mã nguồn mở, không có chi phí bản quyền.

**- Tính mở rộng cao:** PostgreSQL cung cấp khả năng mở rộng theo chiều ngang với các read replica và connection pooling, giúp nó phù hợp với các ứng dụng có lưu lượng truy cập cao và đồng thời cao.

**- Cộng đồng mạnh mẽ:** PostgreSQL có một cộng đồng người dùng và nhà phát triển lớn, năng động, giúp việc tìm kiếm hỗ trợ và tài nguyên trở nên dễ dàng hơn.

**- Tương thích tốt với các nền tảng đám mây:** PostgreSQL có kiến trúc thân thiện và phù hợp với việc triển khai trên nền tảng đám mây hiện đại.

* + 1. **Nhược điểm**

- **Thiếu công cụ tích hợp sẵn:** So với một số hệ thống cơ sở dữ liệu khác, PostgreSQL không có các công cụ tích hợp sẵn để sao lưu, giám sát và quản lý. Tuy nhiên, có rất nhiều công cụ bên thứ ba có sẵn để cung cấp các tính năng này.

**- Không được "chống lưng" bởi một tổ chức uy tín:** Không như các hệ quản trị cơ sở dữ liệu phổ biến hiện nay (Oracle, SQLServer), Postgres là một dự án open source. Nhiều người dùng vẫn có hoài nghi về độ tin cậy của cơ sở dữ liệu này.

* 1. **Docker**

Docker là một nền tảng mở, phục vụ việc phát triển, triển khai và vận hành các ứng dụng. Docker cho phép tách biệt ứng dụng khỏi cơ sở hạ tầng của bạn để có thể nhanh chóng triển khai sản phẩm. Docker cung cấp khả năng đóng gói và chạy ứng dụng trong một môi trường cách ly gọi là Container. Sự cách ly này cho phép bạn chạy nhiều Container trên cùng một máy chủ. Các Container có kích thước nhỏ và chứa tất cả các phần mềm cần thiết để chạy ứng dụng, vì vậy chúng không phụ thuộc vào các phần mềm trên máy chủ. Các Container này có thể được chia sẻ trong khi làm việc cũng như khi triển khai ứng dụng và đảm bảo chúng đều hoạt động giống nhau.

* + 1. **Đặc trưng**

- **Triển khai úng dụng nhanh chóng, nhất quán:** Docker giúp tối ưu hóa vòng đời phát triển ứng dụng bằng cách cho phép các lập trình viên làm việc trong các môi trường chuẩn hóa.

**- Triển khai linh hoạt:** Các Container của Docker có độ linh hoạt cao, chúng có thể chạy trên máy tính cá nhân của lập trình viên, trên máy vật lý hoặc máy ảo ở data center, dịch vụ đám mây...

**- Chạy nhiều tác vụ hơn trên cùng một phần cứng:** Docker có kích thước nhỏ và nhanh. Nó cung cấp một giải pháp thay thế hiệu quả và tiết kiệm chi phí so với các máy ảo dựa trên hypervisor, giúp bạn tận dụng tối đa tài nguyên của máy chủ.

* + 1. **Ưu điểm**

**- Nhất quán giữa các nền tảng khác nhau:** Docker cho phép triển khai ứng dụng không phụ thuộc vào nền tảng máy chủ.

**- Lưu trữ Serverless:** Các Container Docker dựa trên nền tảng đám mây và không cần nhiều bộ nhớ để hoạt động

**- Triển khai nhanh chóng:** Loại bỏ các bước cài đặt, cấu hình thừa thãi, giúp việc triển khai được nhanh chóng và dễ dàng.

**- Linh hoạt và mở rộng:** Các lập trình viên có thể sử dụng bất kỳ công nghệ nào và tăng hoặc giảm tài nguyên Container sử dụng theo nhu cầu.

* + 1. **Nhược điểm**

**- Khó nắm bắt:** Lập trình viên khi chuyển từ nền tảng khác sang Docker có thể dễ dàng làm quen với các chức năng cơ bản nhưng khó để nắm bắt hoàn toàn.

**- Hạn chế về khả năng điều phối:** Dù Docker cung cấp một số tính năng tự động hóa, nhưng chúng không mạnh mẽ bằng các nền tảng Container hóa khác như Kubernetes. Việc quản lý nhiều container và môi trường cùng một lúc có thể trở nên khó khăn.

* 1. **cAdvisor**

cAdvisor (Container Advisor) được phát triển với Go, giúp người dùng Container hiểu rõ hơn về mức sử dụng tài nguyên, hiệu suất của các Containers đang chạy. Nó là một daemon có nhiệm vụ thu thập, tổng hợp, xử lý và xuất các thông tin trên về các Container đang hoạt động.

* + 1. **Đặc trưng**

**- Theo dõi tiêu thụ tài nguyên:** cAdvisor theo dõi mức sử dụng tài nguyên (CPU, bộ nhớ, mạng, disk) trong thời gian thực của tất cả các Containers trên máy chủ.

- **Tích hợp với Docker:** cAdvisor hoạt động natively với các Container Docker, giúp việc tích hợp với Docker liền mạch và dễ dàng.

**- Giao diện Web:** cAdvisor cung cấp một Web UI đơn giản cho phép theo dõi các thông tin trên.

* + 1. **Ưu điểm**

**- Kích thước nhỏ:** cAdvisor có kích thước rất nhỏ và không làm ảnh hưởng nhiều đến hiệu suất của hệ thống, cho phép theo dõi tài nguyên mà không gây giảm hiệu suất các ứng dụng khác.

**- Dễ sử dụng:** Với Web UI và cấu hình đơn giản, cAdvisor dễ triển khai và sử dụng, kể cả với các người dùng không có kinh nghiệm về giám sát hệ thống.

**- Khả năng tích hợp với Grafana và Prometheus:** cAdvisor tương thích với Prometheus, làm tác vụ thu thập và trực quan hóa dữ liệu trở nên dễ dàng.

* + 1. **Nhược điểm**

**- Không hỗ trợ quản lý log:** cAdvisor tập trung vào theo dõi mức sử dụng tài nguyên nhưng không phục vụ việc quản lý log của Container.

**- Thiếu các tùy chọn trực quan hóa phức tạp:** cAdvisor chỉ cung cấp giao diện cơ bản, không có các tùy chọn nâng cao. Để phân tích chi tiết hơn, người dùng cần kết hợp sử dụng với các công cụ khác như Grafana và Prometheus.

**- Thiếu tính mở rộng:** cAdvisor chỉ được thiết kế để theo dõi các máy chủ đơn lẻ. Vì vậy, trong các trường hợp cần theo dõi nhiều Container phân tán trên nhiều máy chủ khác nhay, cAdvisor không phải là lựa chọn tốt.

* 1. **Prometheus**

Prometheus là một và dịch vụ theo dõi hệ thống do Cloud Native Computing Foundation phát triển. Prometheus thu thập chỉ số từ các mục tiêu được cấu hình trước theo các khoảng thời gian nhất định, thực hiện các biểu thức và hiển thị kết quả. Nó còn có thể kích hoạt cảnh báo khi các điều kiện nhất định xuất hiện.

* + 1. **Đặc trưng**

- **Mô hình dự liệu đa chiều:** chuỗi thời gian được định nghĩa bởi tên chỉ số và tập hợp các chiều key/value.

**- PromQL:** Prometheus sử dụng một ngôn ngữ truy vấn mạnh mẽ và linh hoạt để tận dụng tính đa chiều của dữ liệu

**- Không phụ thuộc vào lưu trữ phân tán:** Máy chủ độc lập và tự quản lý dữ liệu.

**- Hỗ trợ nhiều chế độ vẽ đồ thị và tạo dashboard:** Prometheus hỗ trợ nhiều chế độ tạo đồ thị và dashboard cho việc hiển thị và phân tích dữ liệu.

* + 1. **Ưu điểm**

**- Khả năng mở rộng:** Prometheus có khả năng mở rộng theo chiều ngang, có nghĩa là bạn có thể thêm nhiều server Prometheus khi khối lượng dữ liệu tăng lên.

**- Sử dụng mô hình HTTP pull:** Prometheus tự động thu thập chỉ số từ các mục tiêu mỗi khoảng thời gian nhất định. Hướng tiếp cận này cho đảm bảo chỉ số sẽ được thu thập nhất quán không phụ thuộc vào vòng đời của dịch vụ mà nó theo dõi, làm Prometheus phù hợp để theo dõi các dịch vụ thay đổi thường xuyên.

**- Hiệu quả về chi phí:** Prometheus là một dự án open-source, miễn phí sử dụng. Vì vậy nó là một lựa chọn hiệu quả về chi phí cho tất cả người dùng.

* + 1. **Nhược điểm**

**- Hạn chế về lưu trữ dài hạn:** Prometheus được thiết kế chủ yếu cho giám sát thời gian thực và lưu trữ dữ liệu ngắn đến trung hạn. Nó không tối ưu cho việc lưu trữ dữ liệu lịch sử dài hạn. Điều này có thể là một hạn chế cho nhu cầu phân tích trong khoảng thời gian dài.

**- Khả năng trực quan hóa cơ bản:** Các tính năng trình duyệt biểu thức và vẽ đồ thị tích hợp sẵn của Prometheus khá cơ bản. Để tạo các bảng điều khiển và trực quan hóa nâng cao, thường cần tích hợp với các công cụ bên ngoài như Grafana hoặc SigNoz.

**- Hạn chế của mô hình pull-based:** Mặc dù mô hình pull-based có những ưu điểm nhất định, nhưng cũng có một số thách thức như: không phù hợp với các công việc ngắn hạn không sống đủ lâu để scrape...

* 1. **Grafana OSS**

Grafana Open Source Software (OSS) cho phép bạn truy vấn, trực quan hóa, cảnh báo và theo dõi các chỉ số, logs và traces bất kể chúng được lưu trữ ở đâu. Grafana sử dụng các plugin kết nối và hỗ trợ truy vấn dữ liệu từ các nguồn khác nhau (Prometheus, CloudWatch, Loki, các cơ sở dữ liệu SQL/NoSQL...). Grafana còn cung cấp cho các công cụ để hiển thị dữ liệu đó trên các bảng điều khiển với biểu đồ và hình ảnh trực quan.

* + 1. **Đặc trưng**

**- Trực quan hóa dữ liệu tại thời gian thực:** Grafana cung cấp bộ công cụ trực quan phong phú, từ biểu đồ chuỗi thời gian đến heatmaps, các biểu đồ 3D tiên tiến, giúp giải mã các dữ liệu phức tạp.

**- Xây dựng bản điều khiển mượt mà:** Với hơn 150 plugin, Grafane có thể hợp nhất tất cả các nguồn dữ liệu vào một bảng điểu khiển duy nhất, làm đơn giản hóa việc giám sát và xử lý sự cố.

**- Giải quyết vấn đề nhanh hơn với Grafana Alerting:** Grafana hỗ trợ cảnh báo từ nhiều nguồn dữ liệu khác nhau để nhanh chóng xác định vị trí lỗi và khắc phục chúng.

* + 1. **Ưu điểm**

**- Hỗ trợ nhiều kiểu trực quan hóa dữ liệu:** Grafana cung cấp nhiều loại hình biểu đồ: đồ thị, bảng, đồng hồ, heatmap... giúp người dùng thoải mái lựa chọn kiểu biểu diễn phù hợp cho dữ liệu cần theo dõi.

**- Hỗ trợ tốt nhiều nguồn dữ liệu:** Grafana có thể tích hợp với nhiều nguồn dữ liệu từ các nền tảng và môi trường khác nhau trực tiếp, giúp đơn giản hóa quá trình cài đặt và phân tích dữ liệu.

**- Cảnh báo và thông báo:** Hệ thống cảnh báo của Grafana có thể thông báo khi các nhóm dữ liệu vượt quá hoặc thấp hơn ngưỡng được cài đặt trước qua các kênh email, Slack... giúp người dùng giám sát và xử lý sự cố hiệu quả.

**- Cộng đồng, hệ sinh thái mạnh mẽ, tài liệu chi tiết:** Grafana có cộng đồng người dùng lớn và năng động. Hệ sinh thái Grafana cũng vô cùng phát triển và phong phú, với bộ tài liệu chi tiết và được cập nhật thường xuyên.

* + 1. **Nhược điểm**

**- Phức tạp để sử dụng:** Đối với người mới bắt đầu, sử dụng Grafana có nhiều thách thức. Việc thiết lập các bảng điều khiển của Grafana, tạo cảnh báo yêu cầu phải có hiểu biết về các ngôn ngữ truy vấn cụ thể như PromQL, cũng như phải làm quen với rất nhiều tùy chọn trong nên tảng.

**- Tiêu tốn tài nguyên:** Grafana có thể tiêu tốn nhiều tài nguyên, đặc biệt là khi xử lý các bảng điều khiển phức tạp hoặc xử lý số lượng truy vấn lớn.

1. **Nội dung và phương pháp**
   1. **Nội dung**

Để đảm bảo kết quả so sánh toàn diện giữa hai framework, các trường hợp thử nghiệm sẽ bao gồm ba tình huống: **hệ thống ở trạng thái idle, request trả về dữ liệu JSON tĩnh, request trả về dữ liệu từ cơ sở dữ liệu và requests thực hiện thuật toán convert ảnh.** Các tiêu chí đo lường trong quá trình thử nghiệm là: mức độ sử dụng tài nguyên hệ thống (CPU, bộ nhớ), độ trễ (latency), thông lượng (throughput)

* + 1. **Trường hợp hệ thống idle**

**- Mục tiêu:** Đo lường mức sử dụng tài nguyên hệ thống (CPU, bộ nhớ) khi hệ thống không xử lý request nào để xác định mức tài nguyên cần thiết để duy trì hệ thống trong trạng thái sẵn sàng làm việc.

**- Cách thực hiện:** Khởi động ứng dụng Gin và Spring Boot và để chúng hoạt động ở trạng thái chờ trong một khoảng thời gian mà không có request nào

**- Kết quả kỳ vọng:** Vì cả hai hệ thống đều ở trạng thái không xử lý request nên mức tiêu thụ tài nguyên ở mức tối thiểu. Kết quả sẽ giúp so sánh mức tiêu thụ tài nguyên để duy trì trạng thái sẵn sàng giữa hai Framework.

* + 1. **Request trả về dữ liệu JSON tĩnh**

**- Mục tiêu:** Đánh giá hiệu suất khi hệ thống chỉ xử lý các tác vụ trả về dữ liệu JSON đơn giản mà không cần thực hiện truy vấn cơ sở dữ liệu.

**- Cách thực hiện:**

Tạo một endpoint trả về một đối tượng JSON tĩnh, giả lập dữ liệu người dùng, ví dụ:{ "id": 1, "name": "Test User", "email": "user@example.com", "role": "admin" }

Ứng dụng client gửi một loạt request đến endpoint này với số lượng tăng dần (ví dụ: từ 100 đến 10000 request/s) để mô phỏng tải thực tế.

**- Kết quả kỳ vọng:** Vì đây là dữ liệu tĩnh nên thời gian xử lý chỉ phụ thuộc vào hiệu suất của Framework và cấu trúc JSON. Gin sẽ có lợi thế vì được tối ưu hóa cho tốc độ và các tác vụ xử lý đơn giản.

* + 1. **Request trả về dữ liệu từ cơ sở dữ liệu**

**- Mục tiêu:** Đánh giá hiệu suất khi hệ thống phải thực hiện truy vấn cơ sở dữ liệu, nhằm đánh giá khả năng xử lý của từng Framework khi làm việc với cơ sở dữ liệu.

**- Cách thực hiện:**

Thiết lập một cơ sở dữ liệu PostgreSQL chứa dữ liệu người dùng. Mỗi ứng dụng Gin và Spring Boot sẽ có một endpoint như /users, truy vấn danh sách người dùng từ cơ sở dữ liệu.

Ứng dụng client gửi một loạt request đến endpoint này với số lượng tăng dần (ví dụ: từ 100 đến 10000 request/s) để mô phỏng tải thực tế.

**- Kết quả kỳ vọng:** Spring Boot có thể tận dụng các tính năng quản lý kết nối của Java để duy trì hiệu suất ổn định trong khi Gin có thể vượt trội về tốc độ xử lý các truy vấn đơn giản nhờ khả năng tối ưu hóa của Golang. Sự khác biệt về tốc độ xử lý và mức tiêu thụ tài nguyên có thể thấy rõ khi tăng số lượng request hoặc khi truy vấn dữ liệu lớn.

* 1. **Thiết kế, cài đặt hệ thống**

**A diagram of a company

Description automatically generated**

Mỗi service đều được chứa trong một Docker Container. Các Container chứa ứng dụng Spring Boot và Gin và các Container cho cơ sở dữ liệu được cấu hình tài nguyên CPU và RAM tương đồng nhau để đảm bảo tính công bằng. Các Container còn lại có tài nguyên được cấu hình tự động.

* + 1. **demo-java**

Đây là ứng dụng được xây dựng bằng framework Spring Boot (Java). Nó sẽ chạy các API để benchmark và gửi số liệu đến hệ thống giám sát. Các endpoint được sử dụng:

* **/actuator/prometheus:** endpoint được sử dụng để expose các chỉ số của ứng dụng cho service Prometheus
* **/orders/{id}:** trả về order với id cụ thể

Port: 8091

Tài nguyên được giới hạn: 2 CPU, 2GB RAM

* + 1. **demo-gin**

Đây là ứng dụng được xây dựng bằng framework Gin (Go). Tương tự với ứng dụng Spring Boot, nó sẽ chạy các API để benchmark và gửi số liệu đến hệ thống giám sát. Các endpoint được sử dụng:

* **/metrics:** endpoint được sử dụng để expose các chỉ số của ứng dụng cho service Prometheus
* **/orders/{id}:** trả về order với id cụ thể

Port: 8090

Tài nguyên được giới hạn: 2 CPU, 2GB RAM

* + 1. **postgres\_go**

Cơ sở dữ liệu PostgreSQL phục vụ ứng dụng demo-go. Nó chứa dữ liệu thử nghiệm cần thiết để thực hiện benchmark.

Port: 15432

* + 1. **postgres\_java**

Cơ sở dữ liệu PostgreSQL phục vụ ứng dụng demo-java. Nó chứa dữ liệu thử nghiệm cần thiết để thực hiện benchmark.

Port: 25432

* + 1. **prometheus**

Là server giám sát chính của hệ thống, được dùng để thu thập và lưu trữ số liệu từ cadvisor, demo-go, demo-java. Nó cung cấp một cơ sở dữ liệu thời gian thực để lưu trữ các chỉ số và hỗ trợ query để phân tích.

Port: 9090 (để truy cập giao diện Prometheus)

* + 1. **grafana**

Server có nhiệm vụ hiển thị số liệu thu thập bởi prometheus một cách trực quan

Port: 3000 (truy cập giao diện Grafana)

username và password mặc định: admin

* + 1. **Client Machine**

Client Machine có nhiệm vụ gửi request đến demo-java và demo-go trong suốt quá trình thực hiện benchmark với tần suất tăng dần theo thời gian. Đồng thời có nhiệm vụ tính toán Request Duration của mỗi request từ khi gửi đến khi nhận được kết quả và cho phép server prometheus theo dõi chỉ số này.

1. **Hiện trạng**
   1. **Các kết quả đã đạt được**

* Nghiên cứu, tìm hiểu và triển khai thành công các service phục vụ việc theo dõi các chỉ số (CPU, RAM, Latency) của các ứng dụng.
* Xây dựng 5 APIs Spring Boot và Gin với endpoint thực hiện chức năng truy vấn dữ liệu từ cơ sở dữ liệu, chuyển đổi ảnh và trả về cho client dưới dạng JSON.
* Xây dựng được script gửi request đến server với tần suất, địa chỉ URL có thể tùy chỉnh bằng CLI.
  1. **Kết quả sau khi thực nghiệm:**
     1. **Ở trạng thái idle**

A screenshot of a computer

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Gin | Spring Boot |
| CPU | Ổn định trong khoảng 0.1% | Lúc đầu sử dụng nhiều CPU và tăng dần, sau khoảng 1 phút khởi động thì giảm dần và dao động từ 0.7% - 0.8% |
| RAM | Ổn định ở mức 1.5% | Ổn định ở mức 14.5% |

* + 1. **GET 1 static json**

A screenshot of a computer

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Gin | Spring Boot |
| CPU | Sử dụng CPU tăng dần theo số lượng request, đạt đỉnh ở khoảng 12% tại mức 800 requests/s. | Mức sử dụng CPU cao, tăng dần lên đến 20% vào lúc số lượng request chỉ khoảng 170 requests/s. Tuy nhiên, sau đó giảm dần và dao động trong khoảng 15%. |
| RAM | Ổn định ở mức 0.8% trong suốt quá trình. | Sử dụng khoảng 13.5% RAM lúc idle, khi số lượng request tăng cao thì tăng đến khoảng 14.6% và giữ ổn định |

* + 1. **GET 1 order record**

A screenshot of a computer

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Gin | Spring Boot |
| CPU | Sử dụng CPU tăng dần theo số lượng request, đạt đỉnh ở khoảng 18% tại mức 850 requests/s. | Mức sử dụng CPU cao, tăng dần lên đến 20% vào lúc số lượng request chỉ khoảng 170 requests/s. Tuy nhiên, sau đó giảm dần và dao động trong khoảng 15%. |
| RAM | Dao động trong khoảng từ 0.85-1.4%. Mức sử dụng RAM tăng khi số requests tăng cao. | Sử dụng khoảng 12.5% RAM lúc idle, tăng khi bắt đầu nhận được requests, lên đến khoảng 14.5% ở mức 850 requests/s |

* + 1. **POST create 1 order**

A screenshot of a computer screen

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Gin | Spring Boot |
| CPU | Mức sử dụng CPU tăng rất nhanh so với các bài test khác, lên đến khoảng 160% tại thời điểm database bị quá tải. | Mức sử dụng CPU cao, lên đến khoảng 50% tại 620 requests/s. |
| RAM | Ban đầu, mức sử dụng RAM thấp (0.8%). Tuy nhiên, khi database bắt đầu có hiện tượng quá tải thì tăng đột ngột đến 21% | Sử dụng khoảng 12.5% RAM lúc idle, tăng khi bắt đầu nhận được requests, lên đến khoảng 14.5%. Khi database bị quá tải thì RAM tăng đột ngột lên khoảng 19% |

* + 1. **GET all products**

A screenshot of a computer

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Gin | Spring Boot |
| CPU | Lúc bắt đầu tăng vừa phải, khi số lượng request nhiều thì tăng nhanh và vượt qua cả Spring Boot tại 800 req/s với khoảng 50% | Lúc bắt đầu tăng nhanh, đến khoảng 25% thì có khoảng dừng sau đó lại tăng nhanh, đạt mức 49% tại 800 request/s |
| RAM | Ổn định ở mức 0.8% - 0.9% trong suốt quá trình. | Tăng nhẹ từ mức 12% lúc idle đến khoảng 14% khi số lượng request tăng cao và giữ ổn định. |

* + 1. **POST chuyển đổi ảnh màu sang đơn sắc**

A screenshot of a computer

Description automatically generated

|  |  |  |
| --- | --- | --- |
|  | Gin | Spring Boot |
| CPU | Mức sử dụng CPU tăng nhanh từ lúc nhận request, đến mức 110% tại 125 requests/s. | Mức sử dụng CPU tăng rất nhanh, đạt mức 200% (sử dụng hết 2CPU) tại 125 requests/s. |
| RAM | Ổn định trong khoảng dưới 1% | Tăng nhanh từ 12% đến khoảng 34% khi nhận được requests |

* 1. **Tài liệu kết quả**
* Video demo ngắn: <https://drive.google.com/file/d/1RivrgZQwcR_21M3H0GLGCDgl2tjeIIYa/view?usp=sharing>
* Tất cả cả tài liệu, source code liên quan của dự án có thể được truy cập tại GitHub repository: <https://github.com/dkdeptrai/SE400>

1. **Kết luận**

Sau khi thực nghiệm, nhóm đã rút ra kết luận như sau:

|  |  |
| --- | --- |
|  | Tóm tắt kết quả so sánh |
| CPU | Gin có xu hướng sử dung CPU hiệu quả hơn trong các trường hợp tải thấp và trung bình. Tuy nhiên khi tải cao (800 requests/s), Gin tiêu tốn tương đương hoặc thậm chí cao hơn Spring Boot. Trong các tác vụ phức tạp như chuyển đổi ảnh, Gin sử dụng ít CPU hơn đáng kể so với Spring Boot. |
| Memory | Memory: Gin ổn định với mức sử dụng RAM rất thấp (1-1.5%) trong hầu hết bài test so với Spring Boot (12%-34%) trong các bài test, cho thấy lợi thế của Gin trong việc quản lý bộ nhớ. |
| Latency | Gin có độ trễ thấp hơn Spring Boot trong hầu hết các tình huống, đặc biệt với tốc độ xử lý nhanh hơn đáng kể trong tác vụ phức tạp như chuyển đổi ảnh (25ms so với 1.9s). Với tác vụ đọc dữ liệu dữ liệu JSON tĩnh hay từ cơ sở dữ liệu, cả hai Framework đều ổn định với độ trễ thấp sau khi hệ thống đạt trạng thái cân bằng. |
| Database Operation | Gin tỏ ra hiệu quả hơn Spring Boot trong bài test đọc dữ liệu từ database. Tuy nhiên, ở bài test ghi dữ liệu, Gin có kết quả kém hơn Java rõ rệt. Điều này cho thấy Spring Boot có độ tin cậy cao hơn khi tương tác với database |

* 1. **Thuận lợi**
* Tài liệu phong phú: vì 2 framework này là framework phổ biến nhất của Go và Java, nên việc tìm kiếm tài liệu trở nên dễ dàng.
* Có nhiều công cụ hỗ trợ đo: các framework trên đều hỗ trợ nhiều công cụ đo hiệu suất, trong đó phổ biến nhất là Prometheus. Tuy nhiên, Spring đã có Actuator hỗ trợ thực hiện việc đo đạc thông số, còn với Gin phải can thiệp vào code để có thể đo được.
  1. **Khó khăn**
     1. **Khó khăn trong việc làm quen với Grafana**

Trong quá trình triển khai giám sát hệ thống, nhóm gặp trở ngại với việc xây dựng các panel trên Grafana do đây là lần đầu tiếp cận công cụ này:

1. **Khả năng làm quen với PromQL**: PromQL mới đối với các thành viên của nhóm, có cú pháp và logic khác biệt so với các ngôn ngữ thường dùng.
2. **Giao diện phức tạp của Grafana**: Grafana cung cấp nhiều tùy chọn cấu hình và loại biểu đồ, nhưng chính sự phong phú này khiến việc tạo panel trở nên phức tạp.
3. **Thiếu kinh nghiệm với hệ thống giám sát**: Là người mới làm quen với Grafana, nhóm còn gặp khó khăn trong việc đánh giá và lựa chọn các chỉ số quan trọng cần theo dõi.
   * 1. **Giới hạn tần suất gửi request**

**Mô tả hiện tượng:** Khi client gửi request đến tốc độ khoảng 1,500 request/giây, hệ thống không thể tiếp tục gia tăng lưu lượng xử lý mặc dù tài nguyên CPU, RAM và I/O của container vẫn còn khả dụng.

Lỗi: read tcp 127.0.0.1:54096->127.0.0.1:8091: read: connection reset by peer

* **Các lý do khả thi:**
  + **Giới hạn phần mềm:** Có thể phần mềm hoặc cấu hình hệ thống đạt ngưỡng tối đa hoặc đang bị giới hạn bởi số lượng kết nối tối đa.
  + **Cấu hình mạng:** Độ trễ mạng hoặc khả năng kết nối trong container hoặc giữa client và container có thể đang bị giới hạn.
  + **Giới hạn tài nguyên Docker:** Có thể Docker đang áp dụng giới hạn tài nguyên không cho phép container sử dụng tối đa các tài nguyên hệ thống.

1. **Tài liệu tham khảo**

[1]

S. Eseme, “Gin Tutorial: The Ultimate Guide (2023),” *Mastering Backend*, Aug. 09, 2023. https://masteringbackend.com/posts/gin-framework (accessed Nov. 01, 2025).

[2]

Introduction, “Introduction,” *Gin Web Framework*, May 09, 2024. https://gin-gonic.com/docs/introduction/ (accessed Nov. 01, 2025).

[3]

“Spring | Home,” *Home*, 2024. https://spring.io/ (accessed Nov. 01, 2025).

[4]

Эллеонора Керри, “Coffee break #75. Advantages and disadvantages of using Spring Boot. Functions for Strings in Java,” *JavaRush*, Aug. 08, 2023. https://javarush.com/en/groups/posts/en.3380.coffee-break-75-advantages-and-disadvantages-of-using-spring-boot-functions-for-strings-in- (accessed Nov. 13, 2025).

[5]

“PostgreSQL 17.2 Documentation,” *PostgreSQL Documentation*, Nov. 21, 2024. https://www.postgresql.org/docs/current/ (accessed Nov. 06, 2025).

[6]

“Quest Software,” *Quest.com*, 2025. https://www.quest.com/learn/what-is-postgresql.aspx (accessed Jan. 01, 2025).

[7]

DuploCloud, “Docker Advantages and Disadvantages: What You Need to Know Before You Switch,” *DuploCloud*, Dec. 04, 2023. https://duplocloud.com/blog/docker-advantages-and-disadvantages/ (accessed Dec. 15, 2025).

[8]

“prometheus/prometheus: The Prometheus monitoring system and time series database.,” *GitHub*, Nov. 28, 2024. https://github.com/prometheus/prometheus?tab=readme-ov-file (accessed Nov. 01, 2025).

‌ [9]

F. Daniel, “What Are The Benefits of Prometheus?,” *SigNoz*, Jul. 20, 2024. https://signoz.io/guides/what-are-the-benefits-of-prometheus/ (accessed Nov. 01, 2025).

[10]

“Grafana OSS and Enterprise | Grafana documentation,” *Grafana Labs*, 2025. https://grafana.com/docs/grafana/latest/ (accessed Nov. 01, 2025).

[11]

Ozan Unlu, “Maximize the Potential: A Deep Dive into Grafana Pros and Cons,” *Edgedelta.com*, Jun. 03, 2024. https://edgedelta.com/company/blog/grafana-pros-and-cons (accessed Nov. 10, 2025).

[12]

“google/cadvisor: Analyzes resource usage and performance characteristics of running containers.,” *GitHub*, Dec. 10, 2024. https://github.com/google/cadvisor (accessed Nov. 02, 2025).

[13]

“google/cadvisor: Analyzes resource usage and performance characteristics of running containers.,” *GitHub*, Dec. 10, 2024. https://github.com/google/cadvisor (accessed Nov. 03, 2025).

[14]

*Youtube.com*, 2025. https://www.youtube.com/watch?v=PL0c-SvjSVg (accessed Nov. 01, 2025).

[15]

“tutorials/lessons/201 at main · antonputra/tutorials,” *GitHub*, 2025. https://github.com/antonputra/tutorials/tree/main/lessons/201 (accessed Nov. 01, 2025).

‌ [16]

“What is Docker?,” *Docker Documentation*, 2024. https://docs.docker.com/get-started/docker-overview/ (accessed Nov. 12, 2025).