河北工程大学信息与电气工程学院

英文参考文献原文复印件及译文

专　　业：软件工程

姓　　名：陈幼伟

学　　号：211010223

指导教师：韩院彬

完成日期　2025　年　4　月

外文文献

Scalable Micro-Service based Approach to FHIR Server with Golang and No-SQL

Framework Fahim Shariar Shoumik Department of CSE Ahsanullah University of Science and Technology (AUST) Email: fahim.shoumik@gmail.com

Md. Ibna Masum Millat Talukder Department of CSE, AUST

Ahmed Imtiaz Jami Department of CSE, AUST

Neeaz Wahed Protik Department of CSE, AUST

Md. Moinul Hoque Associate Professor Department of CSE Ahsanullah University of Science and Technology Email:moinul@aust.edu

Abstract—Fast Health Interoperability Resource (FHIR) for Electronic Health Record (EHR) not only opens a new era for health-care systems to exchange data between themselves but also allows other various systems following the same framework to communicate between them. FHIR is engaging and evolving rapidly and it is creating the need of massive data storage, retrieval and transfer requirements. For a server that implements this framework, has to be agile to cope with the growing and massive data handling. E-health-care data also seem to grow proportional to time. So, an ideal FHIR Server has to be scalable to fit this demand over time. Several different tools could help store, anonymize, analyze and extract data because there are no single tools to get all those things done. In this work, we propose a scalable, agile, and reliable Micro-service based Architecture for FHIR server that is highly available with the help of Document Oriented Database and helps to store health-care data. The proposed architecture is fast responsive. It gives our system the f lexibility to use different tools that implement FHIR Framework interface in such way that the whole system could be made vertically scalable and the system can perform better in terms of time complexity compared to a Monolithic based implementation of the framework. Experimental results show that the proposed model can be highly used as an alternative to currently available FHIR system implementation.

Keywords: FHIR; Interoperability; REST API; Health-Care system; scalability; micro-service

I. INTRODUCTION

FHIR Stands for Fast Health Interoperable Resource de veloped by HL7 [1]. FHIR can be seen as

a standard that describes two major things for sharing health care records electronically among compatible systems. The first one is the data format structure for storing all sorts of medical records and the second one is the Application Programming Interface (API) on how to access those data. Medical systems implementing the FHIR standard have to preserve their data following the standard and must have interfaces to enable data sharing with other similar systems through the API. It is a framework that guides how health-care data should be stored, manipulated and searched. This framework is still under development with health-care experts around the world. FHIR framework has components for real world concepts like Patient, Practitioner, Device, Organization, Location, Health care Service including but not limited to Allergy, Care-Plan, Observation, Diagnostics, Appointment, Medication, Task etc. FHIR has an implementation guidelines that indicates how a FHIR Server should behave over the REST API [2] Guidelines. FHIR contains a verifiable and testable syntax, a set of rules and constraints, methods and interface signatures including specifications for the implementation of a server capable of requesting and delivering FHIR business objects called 'Resource' which might be represented with JSON, XML, RDF, UML etc.

There are not too many public implementation of the FHIR standard readily available. HAPI (HL7 application program ming interface; pronounced "happy") [3] is an open-source, object-oriented, Monolithic HL7 FHIR Server implementation. The project was initiated by the University Health Network, a large multi-site teaching hospital in Toronto, Canada. By default HAPI FHIR uses Apache Derby as database for its persistence layer to save resources, however it could be configured to use various database supported by JPA 2. FHIRBase[4] is another open-source relational storage for FHIR with document-API based on PostgreSQL. FHIRBase takes the best parts of Relational and Document Databases for persistence of FHIR resources. FHIRBase stores resources relationally and gives the power of SQL for querying and aggregating including a set of SQL procedures and views to persist and retrieve resources as JSON documents. In this work, we have implemented the FHIR standard in terms of Micro-service structure rather than the monolithic based approaches. This enables our system to become agile and fast responsive and also flexible to adapt to various tools.

Micro-service [5] is a way to divide a coupled service into smaller services based on their functions which communicate with each other and gives output as a single unit. Some might consider it as Service Oriented Architecture (SOA) [6] but Micro-service is a Subset of SOA rather than SOA itself, where SOA could be monolithic as well. Micro-service gives agility in development and independent from technological stack 978-1-5386-1150-0/17/$31.00 ©2017 point where each micro-service could be built with different language and database as needed but still functions like a Monolithic System architecture [7]. There is no particular rule when thinking of micro-services from monolithic, its rather application specific . To help us design the micro-service based health-care information management system, we consulted the already defined FHIR[8] components like Administration, Clinical, Financial, Security etc.

II. REASONS FOR MOVING TOWARDS MICRO-SERVICE BASED ARCHITECTURE

Scalability

– A server is marked as scalable if the system is capable of handling the increased load. Scalability of a server nowadays is most important especially for large systems like health-care. A single monolith sys- tem is only capable of scaling through vertically[9] by increasing CPU resource for that particular system and if any portion of the system goes down then the entire application will not function. Whereas with the Micro-service architecture we have the opportunity to horizontally scale the whole system, which means we can connect more CPU that will act as a single unit but may be distributed over multiple networks. Also, we can assign more resource to a particular service handling more load and leave the others as it is.

Service Deployment

– Services should be compiled and containerize into a small container for deployment into clusters. With the help of Golang [1] Cross compilation we can compile each services into small static binaries and couple them with small Docker[10] container for deployment. Docker tools could help scaling each micro-services into large Docker Swarm Cluster which manages the internal communication network and service discovery with load balancer for the containers .

Loose Coupling

– When services are not coupled with each other for functioning and independent it becomes much more easier for developer to develop, maintain and deploy those services. On a Micro-service, a pattern follows decoupled interconnected services.

Agility and Development

– Micro-services reinforce modular structure which is highly independent, having members of all roles and skills that are required to build and maintain, which is particularly important for larger teams. Decoupling teams are as relevant as decoupling software mod- ules.

Highly Availability and API Gateway

– Connecting to various services directly is a bad practice and if that directly connected service goes offline then the system will lose its potential to work as a single unit. To maintain a good service each micro-service should be highly available[11] with time-outs connecting to a load balancer. An API Gateway is required to manages connections to different services if one fails or responses with much delayed time.

Challenges

– Modular Micro-service pattern has some challenges, First one is decoupling the monolithic into micro- services based on business objects. After separa-
tion, an inner service communication mechanism is needed which could be through Remote

Procedure Call (RPC) or a Message Bus to help exchange of messages between services. An API Gateway can help manage the client request for the whole system and forward request into separate micro-services and process the response back to the client in a meaningful format like JSON or XML.

III. PPROPOSED SERVER ARCHITECTURE

Monolithic Systems like the large electronic health record systems eventually tangled into unreasonably large systems. The effect is that problems quickly get out of control thought the monolithic design is comparatively easier to debug and test if there is a need for simple changes to be made in multiple locations. The problem increases with various health care systems across different health-care ecosystems running different versions of services and not connected systems.

Breaking up a monolithic system into smaller services helps simplify the individual build for business objects. While the overall complexity of the system may increase but the savings in developer time and pain is well worth the cost. Each individual Micro-service will be much simpler to understand in isolation. In theory, this decreases the amount of time it takes to on-board new developers into a system.

A. Design Decisions

To build a Micro-service one should identify their business objects. In the health-care domain, there are no fixed sets of 'business objects', but there is a notional and ongoing evolutionary, consensus-based process for identifying some common set of business objects including terms like 'patient', 'procedure', 'observation', 'order', etc. The FHIR specification provides a framework for defining these health-care business objects known as 'Resource'. These business objects are grouped into a single category based on their similarity. For instance, Patient and Practitioner, Location, and Organization resource fall into Administration Category. Allergy, CarePlan, RiskAssessment falls into Clinical Similarly resources like Task, Appointment, Schedule fall into Workflow Category and so on. These grouped business objects have the potential to become separate services

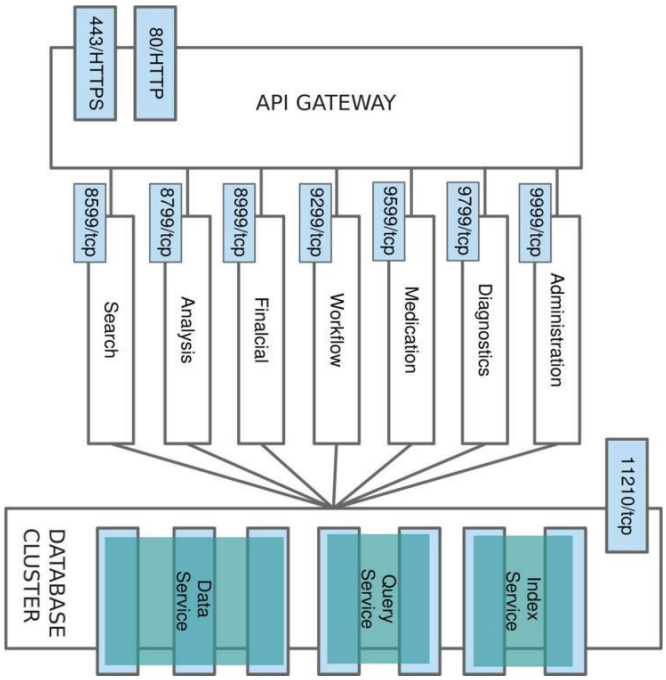

Fig. 1. Micro Service Based FHIR Server (Modularized by FHIR Compo- nents)

Figure 1 illustrates these business objects as separate micro- service in a FHIR Server for our system.

B. API Gateway

Individual micro-services are connected to the API Gateway via Googles RPC(gRPC) on different ports. The Gateway handles the routing and decides which services to talk to. It has the ability to provide the required abstractions at the gateway level for existing micro-services, e.g. rather than providing a one-size-fits-all style API, the API Gateway can expose a different API for each client.

Advantages of having an API Gateway

- Lightweight message routing at the gateway level could be used to do basic message filtering, routing, and trans- formation at the gateway layer.
- Central place to apply non-functional capabilities, such as authentication, security, monitoring and throttling rather implementing these services for each micro layer.

C. Independent Scaling

For our proposed model we choose Couchbase[12], which is a Document Based Storage with Multi Dimensional Scal- ing[13] Capability and SQL like Document Query Language called N1QL[14] which is used in FHIR Search. All those separate

micro-services are connected to a Single Couchbase Database Cluster.

The database cluster in figure 1 demonstrates a deployment topology that can be achieved with Multi Dimensional Scaling Database Service. In this topology, each service is deployed

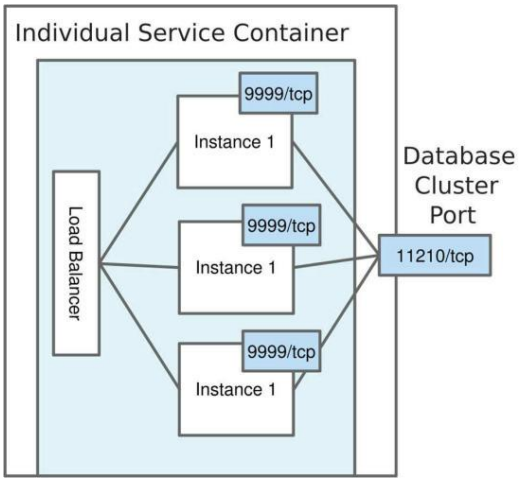

Fig. 2. Scaled Service through a load balancer (Administration)

to an independent zone within the cluster. Each service zone within a cluster (data, query, and index services) can now scale independently so that the best computational capacity is provided for each of them. Each micro-service could also be scaled up to multiple instances for high availability. Figure 2 shows highly available single service containers scaled up to multiple instances in a node managed through a load balancer with the help of docker. This ensures that if one instance of the service panics then the load balancer instantly switches to other instance for request processing

D. Working Procedure of the architecture

• Communication between the client and API Gateway is done via the REST API. • The API Gateway handles the validity of the request via Authentication and then it passes the request to the router.

• The Router decodes the request and passes the message to Service Discovery module where all the micro-services are registered. E.g, a single request might want informa tion about a patient and diagnostic report. Those services belong to different micro-services and requests will get redirected to those service nodes via the gRPC.

• Service Nodes communicate with Database Cluster and returns the result back to the Router.

• The Router then combines the results in a Bundle Re source[15] and returns the

result to the client.

IV. EXPERIMENTALVERIFICATION

Experimental Setup

In order to evaluate the standpoint of our design, we have compared our system with monolithic based imple mentation with same sets data from Synthea (Synthetic Patient Population Simulator) 2. Both of the benchmark is performed with Apache Benchmarking Tool(ab). The benchmark parameter was defined by 1000 as the total request with 100 concurrent request.

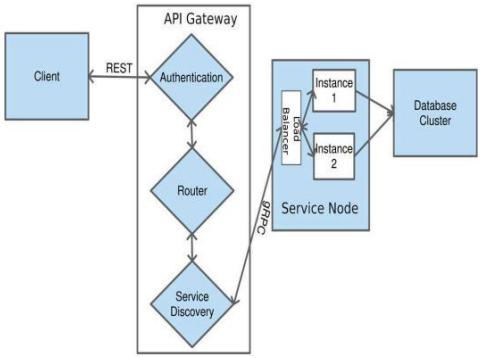

Fig. 3. Worflow Diagram of the entire system

– The Benchmark 1: Focuses on getting a Single

Resource from both the server. [GET:http://localhost:8888/Patient/26d9e7b1-f633 4f85-8f1e-243455c7bc2c]

[GET:http://localhost:80/Patient/26d9e7b1-f633 4f85-8f1e-243455c7bc2c]

– The Benchmark 2:

Fetches a bundle resource that contains all the Ob servation results of a patient using his/her unique identifier calls [GET] were made to both server located in different port using

[GET:http://localhost:8888/Observation?subject=
Patient/26d9e7b1-f633-4f85-8f1e-243455c7bc2c]

[GET:http://localhost:80/Observation?subject=
Patient/26d9e7b1-f633-4f85-8f1e-243455c7bc2c]

Both the benchmarking are performed in a virtual machine (vagrant) with two separate instance each containing (2 logical cores) having 2 GB of Ram. Both

the Monolithic and The Micro-service Server were run via docker. Monolithic server used for benchmarking was a docker image of a HAPI FHIR Server exposing on port 8888 and assem bled micro-service based FHIR Server with an API Gateway exposing to port 80. Figure 4 and Figure 5 show the results in terms of number of requests made vs response time. In the results (figure 4 and figure 5) Monolithic system is marked with a purple colour line and our Micro-service based architecture is marked with a green coloured smooth line. We can see from the result that, as the number of requests increases, the monolith system seem to loose performance taking longer time to response and our Micro-service based system seems to keep up with the performance over the time because the API Gateway only connects to ‘Administration‘ service when we requested a single Patient information same thing happens for benchmark 2, it only communicates with the Diagnostics Micro-service because that is where all the models and business objects are stored for observation results.

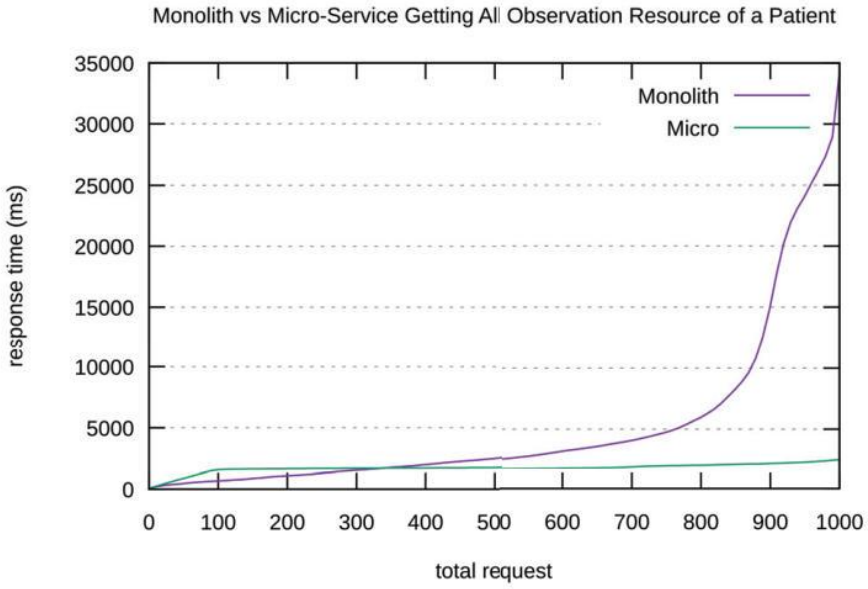

Fig. 4. Response Time for the benchmark 1 scenari

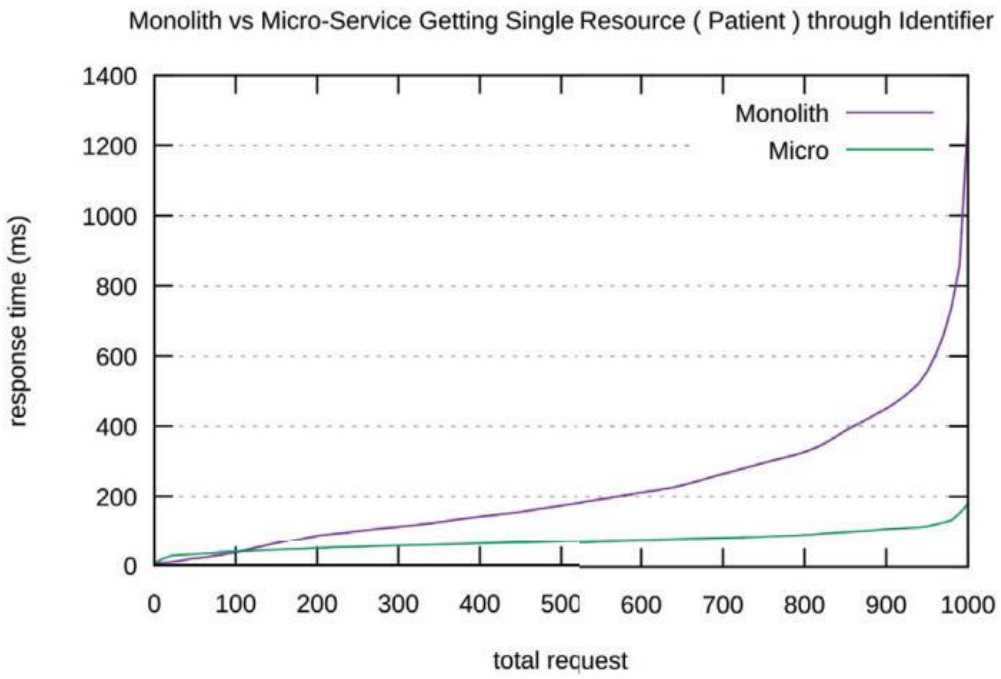

Fig. 5. Response Time for the benchmark 2 scenario

V. DISCUSSIONANDFUTUREWORK

Our proposed model achieve scalability and performance by separating Gigantic Monolith based health-care application architecture into Micro-services with Go language and Couch base. Go language helps us build a smaller static compiled binary that is easy to deploy and scale via docker and using Couch-base's Multidimensional Scaling, we can achieve performance and stability by vertically scaling the components that handle the data. Virtualization will increase the scalability features to this proposed model. Then we can deploy each containerized micro service in different virtualization balanced through a load balancer which might have multiple cores inside it and the load balancer will choose to scale its services based on heart beat/ health check of each virtualization environment. Adding more cores may not increase the scalability but it will definitely help the load balancer to choose a better virtualization or cpu during the scaling of individual services. This work was presented as an outcome of a continuous research work and in the next iteration of our research, we are working on other plans that include techniques to caching documents in the case of a Database failure and replication of database into multiple clusters. This will

allow the system to be available in the face of known or unknown failures. Moreover, we are also working on data representation and data semantics for extraction of electronic health-care data for a complex analytical purpose which is difficult to attain in the monolithic based implementation.

VI. REFERENCES

1. Joel Rodrigues. Health Information Systems: Concepts, Methodologies, Tools, and Applications, Volume 1. IGI Global, ISBN 978-1-60566-988-5, 2010

2. David Booth et al., Web Services Architecture. World Wide Web Consortium, 11 February 2004. Section 3.1.3: Relationship to the World Wide Web and REST Architectures.

3. http://hapifhir.io/doc intro.html, Accessed on the July 16, 2017.

4. http://fhirbase.github.io/docs.html, Accessed on the July 16, 2017.

5. Richardson, Chris. 'Microservice architecture pattern'. microservices.io. Retrieved on 2017-03-19.

6. 'Chapter 1: Service Oriented Architecture (SOA)'. msdn.microsoft.com. Retrieved on 2016-09-21.

7. Rod Stephens, 'Beginning Software Engineering', John Wiley and Sons, pp. 94. ISBN 978-1-118-96916-8, 2015

8. SMART on FHIR: a standards-based, interoperable apps platform for electronic health records.

9. Hesham El-Rewini and Mostafa Abd-El-Barr, Advanced Computer Architecture and Parallel Processing. John Wiley and Sons. p. 66. ISBN 978-0-471-47839-3, 2005.

10. Vivek Ratan, Docker: A Favourite in the DevOps World. Open Source Forum. Retrieved July 14, 2017.

11. Floyd Piedad, Michael Hawkins (2001). High Availability: Design, Techniques, and Processes. Prentice Hall. ISBN 978 0-130-96288-1.

12.https://developer.couchbase.com/documentation/server /4.6/introduction/intro.html, Accessed on the July 16 ,2017

13.Borg, I., Groenen, P., Modern Multidimensional Scaling: theory and applications (2nd ed.). New York: Springer-Verlag. pp. 207212. ISBN 0-387-94845-7, 2005

14. Andrew Slater., Ssssh! dont tell anyone but Couchbase is a serious contender:

Couchbase Live Europe 2015, March 24, 2015

15. https://www.hl7.org/fhir/bundle.html, the July 16, 2017

中文翻译

基基于 Golang 和非 SQL 的可扩展微服务方法构建 Contoso 服务器

摘要:

快速医疗互操作资源(Fast Healthcare Interoperability Resources，FHIR)为电子健康记录（Electronic Health Record，EHR）系统之间的数据交换开启了新时代，不仅促进了医疗系统之间的数据共享，也允许其他遵循相同框架的各种系统进行通信。 FHIR 正在迅速发展，引发了对海量数据存储、检索和传输的需求。 实现该框架的服务器必须具备敏捷性，以应对不断增长的大规模数据处理。 电子医疗数据的增长也与时间成正比，因此，理想的 FHIR 服务器必须具有可扩展性，以适应随时间增加的需求。 由于没有单一工具能够完成所有这些任务，需要多种不同的工具来帮助存储、匿名化、分析和提取数据。 在本研究中，我们提出了一种基于微服务架构的可扩展、敏捷且可靠的 FHIR 服务器，该服务器结合了面向文档的数据库，具有高可用性，能够存储医疗数据。 所提出的架构响应迅速，为我们的系统提供了灵活性，使其能够使用不同的工具来实现 FHIR 框架接口，从而使整个系统能够进行垂直扩展，并在时间复杂度方面相比于基于单体架构的实现表现更优。 实验结果表明，该模型可以作为当前可用的 FHIR 系统实现的替代方案。

关键词：FHIR；互操作性；REST API；医疗系统；可扩展性；微服务

PC, PDA，平板电脑等。

1.引言

快速医疗互操作资源（Fast Health Interoperability Resources，FHIR）是由 HL7 组织开发的标准，旨在规范医疗系统间电子健康记录（EHR）的共享。

FHIR 标准主要定义了两方面内容：一是用于存储各类医疗记录的数据格式结构，二是访问这些数据的应用程序接口（API）。 采用 FHIR 标准的医疗系统需按照该标准存储数据，并通过 API 接口与其他遵循相同标准的系统共享数据。该框架指导医疗数据的存储、操作和检索方式，目前正由全球医疗专家共同开发完善。

FHIR 框架包含了对现实世界概念的组件，如患者（Patient）、从业者（Practitioner）、设备（Device）、组织（Organization）、地点（Location）、医疗服务(Healthcare Service)，以及过敏(Allergy)、护理计划(Care-Plan)、观察（Observation）、诊断（Diagnostics）、预约（Appointment）、用药

（Medication）、任务（Task）等。 FHIR 的实施指南规定了 FHIR 服务器在 REST API 上的行为规范。 FHIR 具有可验证和可测试的语法、一套规则和约束、方法和接口签名，包括实现能够请求和提供 FHIR 业务对象（称为“资源”）的服务器的规范，这些资源可以用 JSON、XML、RDF、UML 等表示。

目前，公开可用的 FHIR 标准实现并不多。 HAPI（HL7 应用程序接口，发音为“happy”）是一个开源的、面向对象的、基于单体架构的 HL7 FHIR 服务器实现。 该项目由加拿大多伦多大学健康网络医院发起。 默认情况下，HAPI FHIR 使用 Apache Derby 作为其持久层数据库来保存资源，但也可以配置为使用 JPA 2 支持的各种数据库。 FHIRBase 是另一个基于 PostgreSQL 的开源关系型 FHIR 存储，具有基于文档的 API。 FHIRBase 结合了关系型和文档型数据库的优点，用于持久化 FHIR 资源。 它以关系型方式存储资源，并利用 SQL 的强大功能进行查询和聚合，包括一组 SQL 过程和视图，以 JSON 文档的形式持久化和检索资源。

在本研究中，我们采用微服务架构实现了 FHIR 标准，而非传统的单体架构方法。 这种方式使我们的系统更加敏捷、响应迅速，并具有适应各种工具的灵活性。 微服务是一种将紧密耦合的服务根据其功能划分为更小的服务的方法，这些小型服务相互通信，并作为一个整体输出结果。 有些人可能认为这类似于面向服务的架构（SOA），但微服务是 SOA 的一个子集，SOA 也可能是单体式的。

微服务在开发中具有敏捷性，并且独立于技术栈，每个微服务可以根据需要使用不同的编程语言和数据库，但仍然像单体系统架构一样运行。 从单体架构转向微服务架构并没有固定的规则，而是取决于具体的应用。 为了设计基于微服务的医疗信息管理系统，我们参考了 FHIR 已定义的组件，如行政管理、临床、财务、安全等。

2.转向微服务架构的原因

在现代医疗系统等大型应用中，微服务架构因其灵活性和高效性，逐渐成为主流选择。以下是采用微服务架构的主要原因：

可扩展性

微服务架构支持水平扩展，即通过增加更多服务实例来处理更高的负载。这使得系统能够根据需求动态调整资源分配，提升性能和响应速度。 与传统的单体架构相比，微服务架构在扩展性方面具有明显优势。

服务部署

微服务架构允许将每个服务独立编译并容器化，便于在集群环境中部署和管理。 利用容器化技术，如 Docker，可以将服务打包成轻量级、可移植的容器，简化部署流程。 此外，容器编排工具（如 Kubernetes）能够管理服务的内部

通信、服务发现和负载均衡，进一步提升部署效率和系统稳定性。

松耦合

微服务架构强调服务之间的独立性和低耦合度。 这种设计使得开发人员可以更容易地开发、维护和部署各个服务。 服务之间通过明确的接口进行通信，减少了相互依赖，提高了系统的灵活性和可维护性。

敏捷开发

微服务架构支持敏捷开发模式，允许小型团队专注于特定服务的开发和维护。这种方式减少了沟通成本，提高了开发效率，并使团队能够快速响应市场变化，持续交付高质量的软件产品。

高可用性与 API 网关

微服务架构通过服务隔离提高了系统的容错能力，即使某个服务出现故障，也不会影响整个系统的运行。 引入 API 网关可以统一管理客户端请求，处理服务路由、负载均衡和故障转移等功能，确保系统的高可用性和稳定性。

挑战

尽管微服务架构带来了诸多优势，但也伴随着一些挑战。 首先，需要将单体应用拆分为基于业务对象的微服务，这要求对业务领域有深入的理解。 其次，服务间的通信需要可靠的机制，如远程过程调用（RPC）或消息总线，以确保服务之间的有效交互。 此外，引入 API 网关可以帮助管理客户端请求，将其分发到相应的微服务，并以 JSON 或 XML 等格式返回处理结果。

3.提议的服务器架构

传统的单体式电子健康记录系统（EHR-S）往往演变成过于庞大的系统，导致问题变得难以控制。 尽管单体式设计在需要对多个位置进行简单更改时相对容易调试和测试，但随着不同医疗生态系统中运行不同版本服务且未连接的系统的增加，问题也随之加剧。

将单体系统拆分为更小的服务有助于简化各业务对象的构建。 虽然系统的整体复杂性可能增加，但在开发时间和降低难度方面的收益是值得的。 每个独立的微服务在孤立情况下更易于理解。 理论上，这减少了新开发人员熟悉系统所需的时间。

A. 设计决策

构建微服务时，首先需要识别其业务对象。 在医疗领域，并没有固定的一组“业务对象”，但通过共识和持续的演进，确定了一些常见的业务对象，如“患者”（Patient）、“手术”（Procedure）、“观察”（Observation）、“订单”（Order）等。 FHIR 规范提供了一个框架，用于定义这些被称为“资源”（Resource）的医疗业务对象。 这些业务对象根据其相似性被归类到单一类别

中。 例如，患者（Patient）、从业者（Practitioner）、地点（Location）和组织(Organization)资源属于管理类别；过敏(Allergy)、护理计划(CarePlan)、风险评估（RiskAssessment）属于临床类别；任务（Task）、预约（Appointment）、日程安排（Schedule）属于工作流程类别，等等。 这些分组的业务对象有潜力成为独立的服务。

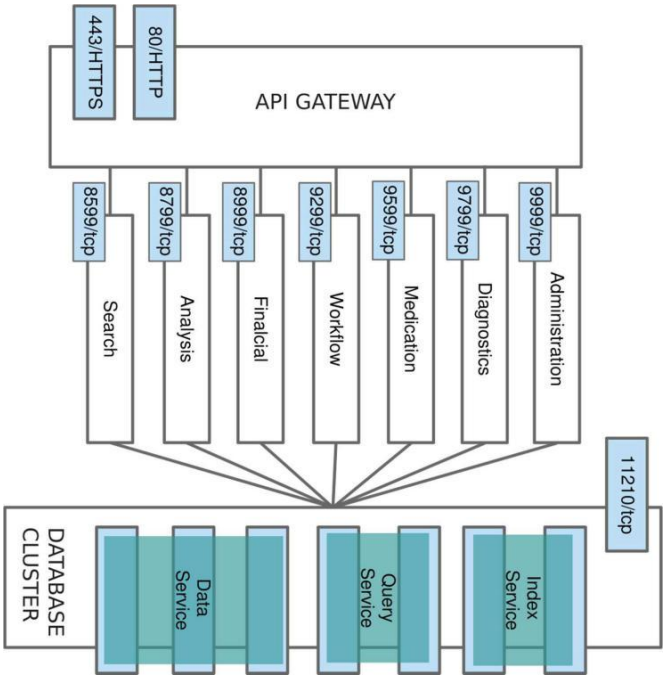

图 1 基于微服务的 FHIR 服务器（按 FHIR 组件模块化）

B. API 网关

各个微服务通过 Google 的 RPC（gRPC）连接到 API 网关，使用不同的端口。网关处理路由并决定与哪些服务进行通信。它能够为现有的微服务提供所需的抽象，例如，API 网关可以为每个客户端暴露不同的 API，而不是提供一种统一的 API。API 网关的优点：

- 在网关层进行轻量级的消息路由，可以用于进行基本的消息过滤、路由和转换。
- 在中心位置应用非功能性能力，如认证、安全性、监控和流量限制，而不是为每个微服务层单独实现这些服务。

C. 独立扩展

对于我们提出的模型，我们选择了 Couchbase[12]，它是一种具有多维扩展[13]能力的文档型存储，并且具有类似 SQL 的文档查询语言 N1QL[14]，该语言用于

FHIR 搜索。所有这些独立的微服务都连接到一个单一的 Couchbase 数据库集群。图 1 中的数据库集群展示了一种可以通过多维扩展数据库服务实现的部署拓扑结构。在这种拓扑中，每个服务都被部署。

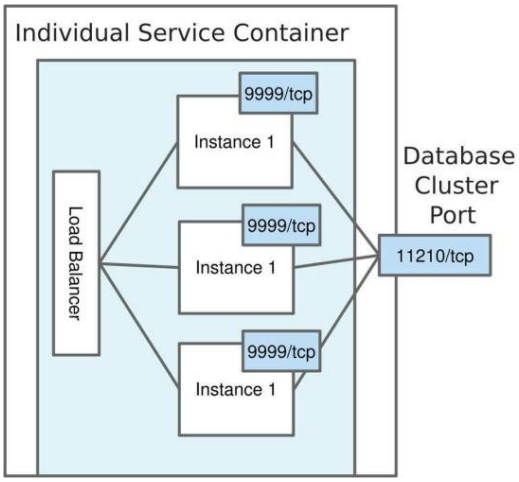

图 2. 通过负载均衡器扩展的服务（管理）

在集群内的独立区域中，每个服务区域（数据、查询和索引服务）现在可以独立扩展，以便为每个服务提供最佳的计算能力。每个微服务也可以扩展为多个实例，以确保高可用性。图 2 展示了通过负载均衡器将单个服务容器扩展为多个实例，并通过 Docker 管理的节点。这确保了如果某个服务实例出现故障，负载均衡器会立即切换到其他实例进行请求处理。

D. 架构的工作流程

- 客户端与 API 网关之间的通信通过 REST API 进行。
- API 网关通过身份验证处理请求的有效性，然后将请求传递给路由器。
- 路由器解码请求并将消息传递给服务发现模块，在该模块中所有微服务都有注册。例如，一个请求可能需要有关患者和诊断报告的信息。这些服务属于不同的微服务，请求将通过 gRPC 重定向到相应的服务节点。
- 服务节点与数据库集群通信并将结果返回给路由器。
- 路由器将结果合并成一个 Bundle 资源，并将结果返回给客户端。

4.实验工作

为了评估我们设计的可行性，我们将我们的系统与基于单体架构的实现进行了比较，使用相同的数据集，数据来自 Synthea（合成患者人口模拟器）。两者的基准测试都使用了 Apache 基准测试工具（ab）。基准测试参数定义为 1000 个总请求，100 个并发请求。

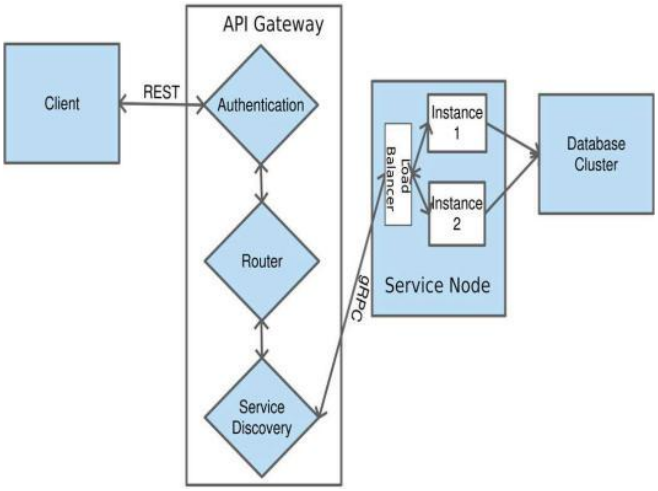

图 3. 整个系统的工作流程图

- 基准测试 1：聚焦于从两个服务器获取单一资源。[GET:http://localhost:8888/Patient/26d9e7b1-f633-4f85-8f1e-243455c7bc2c]

[GET:http://localhost:80/Patient/26d9e7b1-f633-4f85-8f1e-243455c7bc2c] - 基准测试 2：通过患者的唯一标识符获取包含所有观察结果的资源包，通过 [GET] 请求发送到位于不同端口的两个服务器：[GET:http://localhost:8888/Observation?subject=Patient/26d9e7b1-f633-4f85-8f1e-243455c7bc2c]

[GET:http://localhost:80/Observation?subject=Patient/26d9e7b1-f633-4f85-8f1e-243455c7bc2c] 两个基准测试都在虚拟机（vagrant）中进行，虚拟机有两个独立的实例，每个实例包含 2 个逻辑核心和 2GB 内存。单体架构和微服务架构的服务器都通过 Docker 运行。用于基准测试的单体服务器是一个暴露在 8888 端口的 HAPI FHIR 服务器的 Docker 镜像，而基于微服务的 FHIR 服务器则有一个 API 网关，暴露在 80 端口。图 4 和图 5 显示了每个请求与响应时间的关系。从结果（图 4 和图 5）可以看到，单体系统随着请求数量的增加，性能逐渐下降，响应时间变长。而我们的微服务架构则保持较好的性能，因为 API 网关只在请求单一患者信息时连接到“管理”服务；对于基准测试 2，当请求诊断信息时，它只与诊断微服务通信，因为所有模型和业务对象都存储在该微服务中。

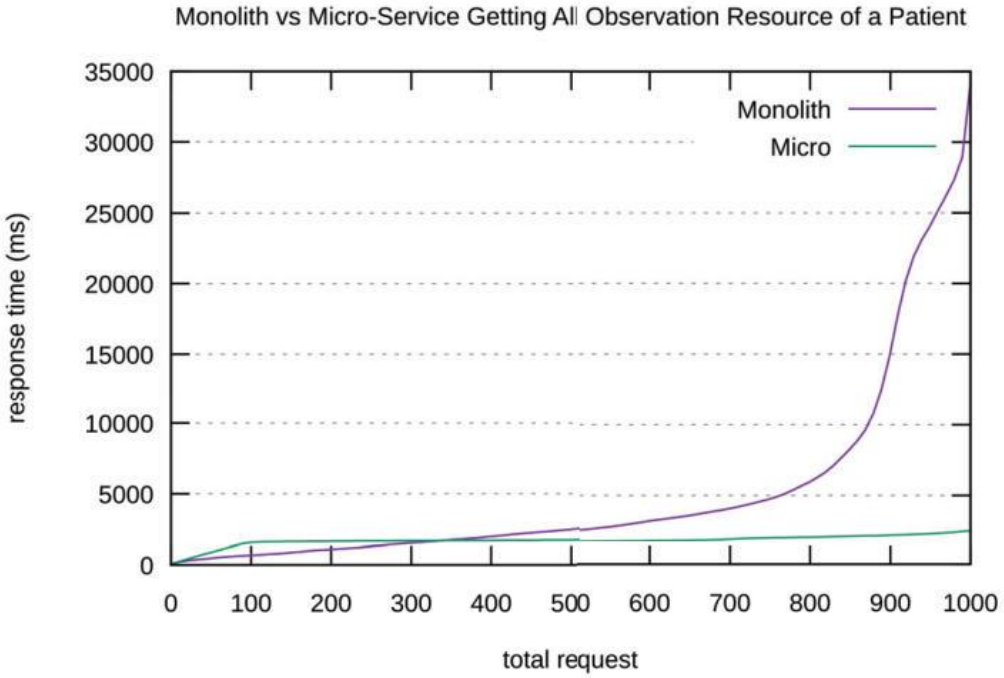

图 4. 基准测试 1 场景的响应时间

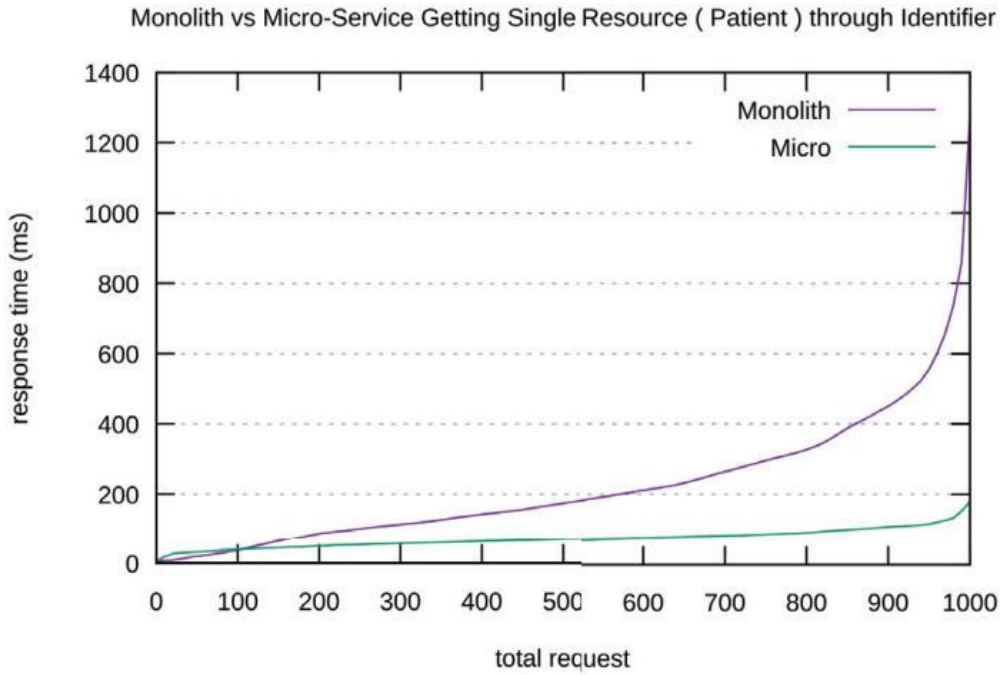

图 5. 基准测试 2 场景的响应时间 b

5.伪代码

我们提出的模型通过将庞大的基于单体的医疗保健应用架构拆分为微服务，结合 Go 语言和 Couchbase，实现了可扩展性和性能。Go 语言帮助我们构建了一个较小的静态编译二进制文件，易于通过 Docker 部署和扩展，而使用 Couchbase 的多维扩展功能，我们可以通过垂直扩展处理数据的组件来实现性能和稳定性。虚拟化将增强这个模型的可扩展性特性。接下来，我们可以将每个容器化的微服务部署在不同的虚拟化环境中，并通过负载均衡器进行平衡，这些虚拟化环境可能具有多个内核，负载均衡器将在每个虚拟化环境的心跳/健康检查基础上决定是否扩展其服务。增加更多的核心可能不会提高可扩展性，但它肯定会帮助负载均衡器在扩展单个服务时选择更合适的虚拟化或 CPU。此项工作是我们持续研究工作的成果之一，在下一轮研究中，我们将关注包括在数据库故障情况下缓存文档的技术和将数据库复制到多个集群的工作。这将使系统能够应对已知或未知的故障。此外，我们还在致力于电子健康数据的提取、数据表示和数据语义分析，这在基于单体的实现中是难以实现的，尤其是在复杂的分析任务中。

6.结果

1.Joel Rodrigues. 《健康信息系统：概念、方法论、工具与应用，第 1 卷》。IGI Global，ISBN 978-1-60566-988-5，2010 年

2.David Booth 等人，《Web 服务架构》。万维网联盟，2004 年 2 月 11 日，第 3.1.3 节：与万维网和 REST 架构的关系。

3.http://hapifhir.io/doc_intro.html，访问时间：2017 年 7 月 16 日

4.http://fhirbase.github.io/docs.html，访问时间：2017 年 7 月 16 日

5.Richardson, Chris. “微服务架构模式”。microservices.io，2017 年 3 月 19 日检索。

6.“第 1 章：面向服务的架构（SOA）”。msdn.microsoft.com，2016 年 9 月 21 日检索。

7.Rod Stephens, 《软件工程入门》，John Wiley & Sons，第 94 页。ISBN 978-1-118-96916-8，2015 年

8.SMART on FHIR：一个基于标准的、互操作的应用平台，用于电子健康记录。

9.Hesham El-Rewini 和 Mostafa Abd-El-Barr, 《先进的计算机架构与并行处理》。John Wiley & Sons，第 66 页。ISBN 978-0-471-47839-3，2005 年

10.Vivek Ratan, 《Docker：DevOps 世界中的宠儿》。开源论坛，2017 年 7 月 14 日检索。

11.Floyd Piedad, Michael Hawkins（2001）。《高可用性：设计、技术与流

程》。Prentice Hall。ISBN 978 0-130-96288-1。

12.https://developer.couchbase.com/documentation/server/4.6/introduction/intro.html，2017 年 7 月 16 日访问

13.Borg, I., Groenen, P.,《现代多维扩展：理论与应用（第 2 版）》纽约：Springer-Verlag，第 207-212 页。ISBN 0-387-94845-7，2005 年

14.Andrew Slater,“嘘！别告诉别人，Couchbase 是一个严肃的竞争者：Couchbase Live Europe 2015”，2015 年 3 月 24 日

15.https://www.hl7.org/fhir/bundle.html，2017 年 7 月 16 日访问

评阅表

1. 英文资料选择合理，与毕业设计论文相关度高。 〇符合　　〇较符合　　〇基本符合　　〇不符合 2. 专业术语、词汇翻译的准确度较高，体现了较强的专业英语应用水平。 〇符合　　〇较符合　　〇基本符合　　〇不符合 3.翻译材料能与原文能保持一致，能正确表达出原文意思。 〇符合　　〇较符合　　〇基本符合　　〇不符合 4. 翻译材料语句通顺，符合中文的表达习惯。 〇符合　　〇较符合　　〇基本符合　　〇不符合 5. 翻译字、词数满足要求。 〇符合　　〇较符合　　〇基本符合　　〇不符合
指导教师签字： 年　月　日

注：此表必须在同一页面。