УДК 004.414.3

ЭФФЕКТИВНАЯ РАЗРАБОТКА ПРИЛОЖЕНИЙ ПРИ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ

А. Э. Порфильева[1], Р. Ф. Шайхутдинов[2], Г. А. Нуриева[3], М. Р. Сидиков[4], М. М. Абрамский[5], А. И. Карпов[6], Д. И. Раимов[7], Р. Р. Новиков[8]

[1-8] *Высшая школа информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета*

[1]porfileva.anastasia@gmail.com, [2]rus.shaikhut@gmail.com, [3]nurievag97@gmail.com, [4]sidikov.marsel@gmail.com, [5]ma@it.kfu.ru, [6]artik100313@gmail.com, [7]dinar88raimov@gmail.com, [8]ruslandia996@gmail.com

Аннотация

Рассмотрены особенности внедрения микросервисной архитектуры в процесс разработки. Проиллюстрированы преимущества данного подхода по сравнению с традиционным монолитным подходом. Показана связь использования микросервисной архитектуры с возможностью работы команды по гибким методологиям разработки.

***Ключевые слова**: микросервисы, микросервисная архитектура, эффективная разработка, гибкие методологии*

ВВЕДЕНИЕ

Моделирование архитектуры программного продукта является важной частью процесса разработки. Выбор конкретного архитектурного решения зависит от многих факторов, в частности, назначения разрабатываемой системы. Несмотря на разнообразие подходов к архитектуре приложений, в рамках современных проектов, нуждающихся в легкой масштабируемости, в последнее время предпочтение отдается микросервисам [1].

Термин Microservice Architecture описывает технологию разработки программного обеспечения, при которой приложение представляет собой совокупность слабосвязанных сервисов. Сервисы строятся вокруг бизнес-логики и имеют возможность независимого развертывания. Каждый сервис работает в своем

процессе и взаимодействует с другими сервисами при помощи легковесных механизмов, таких, например, как HTTP [2]. При этом разработка каждого сервиса может вестись независимо от других. Это позволяет организовать процесс разработки гибко и распределенно.

Основная цель данной работы заключается в определении и описании условий, необходимых для осуществления эффективной разработки на основе микросервисной архитектуры.

Статья организована следующим образом: в первом разделе описан подход сервис-ориентированной архитектуры (SOA), во втором разделе дано описание преимуществ микросервисной архитектуры по сравнению с монолитными архитектурами, в третьем разделе описаны предлагаемые принципы организации эффективной разработки в проектной команде, разрабатывающей приложения в микросервисной архитектуре.

1. СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА

Сервис-ориентированная архитектура (Service-oriented architecture, SOA) [3] – это подход к разработке приложений, который осуществляет публикацию сервисами своих интерфейсов, позволяя другим сервисам обращаться к ним посредством стандартных интернет-протоколов, которые не зависят от языков программирования и платформ.

Далее будет проведен сравнительный анализ нескольких архитектурных патернов, реализующих принципы SOA.

Общая архитектура брокера объектных запросов (CORBA)

CORBA [4] – это архитектура, предназначенная для интеграции различных систем с помощью вызова удаленных процедур, что позволяет не зависеть от операционных систем, языков программирования и оборудования. Для взаимодействия используется абстрактный протокол GIOP, а интерфейсы описаны на языке IDL [5].

Отметим, что стандарт CORBA имел ряд недостатков, среди которых:

- независимость от местоположения – клиентский код не обладает информацией, локальным или удаленным был вызов;
- использование нестандартных протоколов и портов; корпоративные политики безопасности часто не допускают их использования.

Были необходимы надежный канал связи и простой обмен сообщениями, а также было необходимо уменьшить количество удаленных вызовов для повышения производительности системы. В 1990 г. появились веб-сервисы [6] – это приложение с определенным URL, которое предоставляет свой интерфейс для взаимодействия с помощью сообщений:

- приложения взаимодействуют посредством сообщений, поэтому сократилось число удаленных вызовов;
- сообщения отправляются по HTTP-соединению;
- используются простые спецификации для обмена сообщениями (SOAP, REST, GraphQL);
- используются форматы сообщений, которые не зависят от платформ (XML, JSON и т. д).

Однако и веб-сервисы обладали рядом недостатков:

- возможна перегрузка системы из-за синхронного обмена сообщениями;
- по причине различий форматов сообщений трудно интегрировать одни службы с другими.

Через некоторое время появилась **очередь сообщений** – шаблон проектирования, обеспечивающий независимость компонентов системы, которые используются для обмена информацией с заранее определенным форматом сообщений. Очереди сообщений обеспечивают масштабируемость, отказоустойчивость и гарантированную доставку.

В роли брокеров сообщений используются такие инструменты, как RabbitMQ [7], Kafka [8], Beanstalkd [9], Amazon MQ [10] и т. д.

Одним из недостатков этого шаблона является то, что очереди сообщений создают дополнительную операционную сложность. Каждая очередь должна быть создана, настроена и должна контролироваться. Каждому отправителю и получателю необходимо настроить местоположение и имя каждой очереди.

Данный подход решил проблему синхронного обмена сообщениями, но проблема трудной интеграции различных сервисов осталась. Еще одной попыткой решить проблему стал шаблон проектирования «Сервисная шина предприятия» (ESB), который работает как очередь сообщений, но конвертирует сообщения в

нужный формат, тем самым решая проблему интеграции сервисов, использующих разные форматы сообщений [11]. Недостатки данного шаблона:

- снижается скорость связи из-за конвертации сообщений;
- общая точка отказа;
- высокая сложность конфигурирования.

2. МИКРОСЕРВИСНАЯ АРХИТЕКТУРА

Микросервисная архитектура [2] является современным представлением подхода SOA. В микросервисной архитектуре сервисы имеют меньший размер и обмениваются сообщениями внутри приложения. Программный интерфейс сервиса публикуется через шлюз, который транслирует запросы извне в нужные сервисы. Таким образом, микросервисы могут быть разработаны в широком технологическом стеке.

Сервисы возможно реализовать на различных языках программирования с применение разных баз данных, в зависимости от того, что лучше подходит для решения конкретных задач [12].

Микросервисы обычно противопоставляются монолитному подходу – когда приложение построено как единое целое с общей базой данных. Автор книг и статей по архитектуре программного обеспечения Мартин Фаулер определяет несколько преимуществ монолитной и микросервисной архитектур [2]:

Преимущества монолитной архитектуры:

- *простота;*
- *согласованность;*
- *межмодульный рефакторинг* – при необходимости можно переместить классы из одного модуля в другой.

Преимущества микросервисной архитектуры:

- *частичное развертывание* – возможность обновлять отдельные сервисы приложения независимо от других;
- *доступность* – если какие-то из модулей прекращают свое функционирование, то работа других модулей и приложения в целом не прерывается;
- *сохранение модульности* – исключается наличие общих состояний между несколькими модулями, что дает возможность отключать/изымать одни модули, не нарушая работу других;

- *гетерогенность* – возможность реализации модулей системы на разных языках; гетерогенные информационные системы помогают избегать наличия тесных связей между модулями системы благодаря использованию разных языков программирования;
- *независимая масштабируемость* – способность модулей справляться с увеличением рабочей нагрузки при добавлении ресурсов независимо от других модулей, размещаемых на других серверных узлах [13].

Следует учитывать, что многие приложения начинают разрабатываться с помощью монолитного подхода, в силу своих преимуществ зарекомендовавшего себя для небольших команд. Однако со временем монолитные приложения начинают обладать массивной кодовой базой, что означает необходимость развернуть весь стек технологий и собрать целиком проект при разработке сравнительно небольших новых задач.

3. ОРГАНИЗАЦИЯ ГИБКОЙ РАЗРАБОТКИ

В своем блоге «Coding the Architecture» в 2013 году архитектор программного обеспечения Саймон Браун предположил, как будет выглядеть гибкая архитектура программного обеспечения [14]. Его описание гибкой архитектуры достаточно хорошо ложится на микросервисную архитектуру. По его мнению, предоставить гибкость поможет стиль архитектуры, построенный с использованием небольших слабосвязанных компонентов (сервисов), которые взаимодействуют друг с другом для ответа на запрос пользователя. Сервисы могут быть модифицированы и протестированы изолированно или даже вырваны и заменены в зависимости от изменения требований, новые компоненты могут быть добавлены и масштабированы, если необходимо.

Согласно манифесту гибкой разработки (agile manifesto, [15]), успешные и эффективные команды принимают необходимые решения совместно, а рекомендованный размер команды составляет не менее 3 и не более 9 человек [16]. Таким образом, Agile обеспечивает поддержку эффективной совместной разработки в архитектуре микросервисов.

Ведущие технологические организации, такие, как Amazon, Netflix и Apple, используют микросервисную архитектуру для обеспечения гибкости быстрой адаптации и реагирования на запросы своих клиентов [17]. Согласно ресурсам

NGINX, 68% организаций проводят исследования или внедряют микросервисы [18]. Такой рост значимости микросервисов отражает переход от крупных приложений с кодовым содержанием (монолитов) к более легким приложениям.

Однако, несмотря на то что ведущие компании уже делают переход от монолитной архитектуры к микросервисной, существует ряд сложностей, с которыми можно столкнуться при разработке и поддержке и которые надо учитывать при переходе на микросервисную архитектуру:

1. Необходимо предусматривать автономность каждого микросервиса от других компонентов. Если каждый сервис вызывает несколько удаленных сервисов, стоит продумать способы по уменьшению общего времени задержки. Помимо этого, следует учитывать проблемы с надежностью, так как удаленный вызов может отказать в любой момент, таким образом, с увеличением количества сервисов потенциальных точек отказа становится больше.
2. Требуется хранить единые компоненты системы в единой библиотеке, однако в этом случае при изменении данной библиотеки необходимо перезапускать все связанные микросервисы. Другой вариант — хранить единый код во всех микросервисах, однако такой код сложнее поддерживать.
3. Стратегия тестирования, которая применяется к монолитным приложениям, должна меняться при переходе на микросервисы. Учитывая, что приложения, построенные в архитектуре микросервисов, обеспечивают высокую функциональность и производительность, тестирование должно охватывать каждый уровень сервисов и их взаимодействие. Следовательно, увеличивается время, необходимое для тестирования всего приложения.
4. Для внедрения больших систем микросервисов требуются затраты на квалифицированных специалистов, хорошая инфраструктура и ресурсы, необходимые для организации взаимодействия сервисов.

Можно выделить следующие случаи, когда микросервисная архитектура будет поддерживать модель эффективной разработки.

1. *Микрокоманды.* В большом проекте с небольшими командами разработчиков, каждая из которых реализует собственный функционал, изменения одной группы так или иначе затрагивают другие группы — такой подход в команде замедляет процесс разработки. Наличие разных групп разработ-

чиков с общей кодовой базой также приводит к проблемам – по мере увеличения объема кода становится сложно следить за тем, чтобы все части были организованными и оптимально сочетались друг с другом. С архитектурой микросервисов возможно разделить код так, чтобы различные команды разработчиков могли полностью владеть им. Это позволит командам внедрять новшества намного быстрее, без утомительных процессов проектирования, просмотра и развертывания.

2. *Опытный менеджер проекта.* При использовании микросервисов роль руководителя проекта выходит на первый план, так как над различными частями приложения работает множество разных команд. Чтобы синхронизировать работу микрокоманд и избежать проблем, следует нанимать грамотных проектных менеджеров.
3. *Длительный период жизни и поддержки проекта.* Если проект стремительно развивается в своих масштабах и годы разработки только будут его увеличивать, то есть смысл разделять кодовую базу на отдельные компоненты. Внедрение микросервисов помогает при длительной разработке. Если проект необходимо выполнить в кратчайшие сроки, то внедрение новых технологий будет лишь замедлять процесс разработки.
4. *Знания.* Не следует использовать микросервисную архитектуру в том случае, если в команде отсутствуют специалисты с пониманием технологий микросервисов.

Рассмотрим, какие технические знания потребуются java-разработчику для работы с микросервисами. Микросервисы – это распределенная система, каждый сервис должен решать только одну проблему, чтобы не перекомплементировать систему. Проектная группа должна быть готова к реорганизации программных элементов по мере необходимости. Знание контейнеров, таких, как Docker, может быть полезным. Поскольку микросервисы Java обычно связывают REST через HTTP, требуются также знания REST / HTTP / RAML / Swagger. Упростить работу с микросервисами помогут такие фреймворки, как Spring, Spark, Dropwizard, Play Framework и так далее [19].

ЗАКЛЮЧЕНИЕ

При разработке программного продукта необходимо серьезно подойти к выбору архитектурного решения. Микросервисы представляются подходящим выбором для реализации больших систем, предполагающих длительную разработку. Однако микросервисная архитектура несет в себе достаточно много рисков. В этом случае успех проекта во многом зависит от наличия опытного системного архитектора и проектного менеджера.

Команда, использующая микросервисную архитектуру, должна быть сформирована на основе бизнес-возможностей. Одним из обязательных условий успешного внедрения в проект микросервисной архитектуры является компетентность команды.

СПИСОК ЛИТЕРАТУРЫ

1. *Chris Richardson* (2017). Pattern: Microservice Architecture. URL: http://microservices.io/patterns/microservices.html

2. *James Lewis, Martin Fowler* (25 марта 2014). Microservices. URL: https://martinfowler.com/articles/microservices.html

3. Microsoft – Understanding Service-Oriented Architecture. URL: https://msdn.microsoft.com/en-us/library/aa480021.aspx

4. Сайт Corba. URL: http://www.corba.org/

5. *Steve Vinoski.* CORBA: Integrating Diverse Applications Within Distributed Heterogenerous Environments / Steve Vinoski, 1997.

6. *Алекс Родригес* (16.09.2015). Web-сервисы RESTful: основы. URL: https://www.ibm.com/developerworks/ru/library/ws-restfu/index.html

7. Сайт RabbitMQ. URL: https://www.rabbitmq.com/

8. Сайт Apache Kafka. URL: https://kafka.apache.org/intro

9. Сайт Beanstalkd. URL: http://kr.github.io/beanstalkd/

10. Сайт AmazonMQ. URL: https://aws.amazon.com/ru/amazon-mq/

11. Microsoft Message Bus (2004). URL: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff647328(v=pandp.10)

12. *Sam Newman*. Principles of Microservices, 2015. URL: https://vimeo.com/131632250

13. *Michael Hofmann, Erin Schnabel and Katherine Stanley*. Microservices Best Practices for Java / URL: http://www.redbooks.ibm.com/abstracts/sg248357.html

14. *Simon Brown*. Coding the Architecture. URL: http://www. codingthear-chitecture.com/2013/09/03/what_is_agile_software_architecture.html

15. Agile Mnifesto. URL: http://agilemanifesto.org/

16. The Scrum Guide ("The Development Team" chapter) URL: https://www. scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf

17. Leanix. Why Netflix, Amazon, and Apple Care About Microservices. URL: https://blog.leanix.net/en/why-netflix-amazon-and-apple-care-about-microservices

18. NGINX. The Future of Application Development and Delivery Is Now. URL: https://www.nginx.com/resources/library/app-dev-survey/

19. Leanix. Developing Microservices with Java. URL: https://blog.leanix.net/ en/developing-microservices-with-java

EFFECTIVE APPLICATION DEVELOPMENT WITHIN MICROSERVICE ARCHITECTURE

A. E. Porfileva[1], R. F. Shaikhutdinov[2], G. A. Nurieva[3], M. R. Sidikov[4], M. M. Abramskiy[5], A. I. Karpov[6], D. I. Raimov[7], R. R. Novikov[8]

[1-8] *Higher School of Information Technologies and Intelligent Systems, Kazan (Volga Region) Federal University*

[1]porfileva.anastasia@gmail.com, [2]rus.shaikhut@gmail.com, [3]nurievag97@gmail.com, [4]sidikov.marsel@gmail.com, [5]ma@it.kfu.ru, [6]artik100313@gmail.com, [7]dinar88raimov@gmail.com, [8]ruslandia996@gmail.com

Abstract

The paper presents the features of the using the microservice architecture in the development process. The advantages of this approach are illustrated in comparison with the traditional monolithic approach. The connection between the use of the microservice architecture and the ability of the team to work within agile development methodologies is shown.

***Keywords**: microservices, microservice architecture, efficient development, agile methodologies*

REFERENCES

1. *Chris Richardson* (2017). Pattern: Microservice Architecture. URL: http://microservices.io/patterns/microservices.html

2. *James Lewis, Martin Fowler* (25 March 2014). Microservices. URL: https://martinfowler.com/articles/microservices.html

3. Microsoft - Understanding Service-Oriented Architecture. URL: https://msdn. microsoft.com/en-us/library/aa480021.aspx

4. Corba Website. URL: http://www.corba.org/

5. *Steve Vinoski*. CORBA: Integrating Diverse Applications Within Distributed Heterogenerous Environments / Steve Vinoski, 1997

6. *Aleks Rodriges* (16.09.2015). Web services RESTful: basics. URL: https:// www.ibm.com/developerworks/ru/library/ws-restfu/index.html

7. RabbitMQ Website. URL: https://www.rabbitmq.com/

8. Apache Kafka Website. URL: https://kafka.apache.org/intro

9. Beanstalkd Website. URL: http://kr.github.io/beanstalkd/

10. AmazonMQ Website. URL: https://aws.amazon.com/ru/amazon-mq/

11. Microsoft Message Bus (2004). URL: https://docs.microsoft.com/en-us/ previous-versions/msp-n-p/ff647328(v=pandp.10)

12. *Sam Newman*. Principles of Microservices, 2015. URL: https://vimeo.com/ 131632250

13. *Michael Hofmann, Erin Schnabel and Katherine Stanley*. Microservices Best Practices for Java. URL: http://www.redbooks.ibm.com/abstracts/sg248357.html

14. *Simon Brown*. Coding the Architecture blog. URL: http://www. codingthearchitecture.com/2013/09/03/what_is_agile_software_architecture.html

15. Agile Manifesto URL: http://agilemanifesto.org/

16. The Scrum Guide (“The Development Team” chapter) URL: https://www. scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf

17. Leanix, Why Netflix, Amazon, and Apple Care About Microservices. URL: https://blog.leanix.net/en/why-netflix-amazon-and-apple-care-about-microservices

18. NGINX, The Future of Application Development and Delivery Is Now. URL: https://www.nginx.com/resources/library/app-dev-survey/

19. Leanix, Developing Microservices with Java. URL: https://blog.leanix.net/en/ developing-microservices-with-java

СВЕДЕНИЯ ОБ АВТОРАХ

ПОРФИЛЬЕВА Анастасия Эдуардовна – студентка Высшей школы информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Anastasia Eduardovna PORFILEVA – *student of Higher School of ITIS KFU.*

email: porfileva.anastasia@gmail.com

ШАЙХУТДИНОВ Рустем Фаритович – студент Высшей школы информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Rustem Faritovich SHAIKHUTDINOV – student of Higher School of ITIS KFU.

email: rus.shaikhut@gmail.com

НУРИЕВА Гульшат Атласовна – студентка Высшей школы информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Gulshat Atlasovna NURIEVA – student of Higher School of ITIS KFU

email: nurievag97@gmail.com

СИДИКОВ Марсель Рафаэлевич – руководитель лаборатории Java Высшей школы информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Marsel Rafaelevich SIDIKOV – head of Java Lab of ITIS KFU.

email: sidikov.marsel@gmail.com

АБРАМСКИЙ Михаил Михайлович – старший преподаватель кафедры программной инженерии Высшей школы информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Mikhail Mikhailovich ABRAMSKIY – senior lecturer at Department of Software Engineering of Higher School of ITIS KFU

email: ma@it.kfu.ru

КАРПОВ Артур Иванович – студент Высшей школы информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Artur Ivanovich KARPOV – student of Higher School of ITIS KFU

email: artik100313@gmail.com

РАИМОВ Динар Ильдусович – студент Высшей школы информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

Dinar Il'dusovich RAIMOV – student of Higher School of ITIS KFU

email: dinar88raimov@gmail.com

НОВИКОВ Руслан Радикович – студент Высшей школы информационных технологий и интеллектуальных систем Казанского (Приволжского) федерального университета.

NOVIKOV Ruslan Radikovich – student of Higher School of ITIS KFU

email: ruslandia996@gmail.com

Материал поступил в редакцию 28 июля 2018 года