УДК 004.27

А. М. Шитько
Белорусский государственный технологический университет

ПРОЕКТИРОВАНИЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Статья посвящена изучению технологии построения микросервисной архитектуры, которая позволяет сохранять модульность и независимость развертывания компонент Enterprise приложения. Микросервисная архитектура – это подход к созданию приложения, подразумевающий отказ от единой, монолитной структуры. То есть вместо того чтобы исполнять все ограниченные контексты приложения на сервере с помощью внутрипроцессных взаимодействий, используется несколько небольших приложений, каждое из которых соответствует какому-то ограниченному контексту. Причем эти приложения работают на разных серверах и взаимодействуют друг с другом по сети, например посредством HTTP. В статье описывается процесс моделирования сервисов, способы интеграции микросервисов между собой, их основные свойства. Рассматривается алгоритм разбиения цельного приложения (монолита) на части, методы развертывания микросервисов. Также исследованы различные типы микросервисной архитектуры и способы их связи. Благодаря анализу полученной информации по теме статьи были выявлены преимущества и недостатки в сравнении с монолитными системами. Результатом выполненного исследования является получение научных и практических знаний для проектирования и разработки высокоэффективных Enterprise приложений, а также для получения хорошей производительности приложения при минимальных трудозатратах.

Ключевые слова: микросервисная архитектура, проектирование, Enterprise приложения.

A. M. Shytska
Belarusian State Technological University

THE DESIGN OF THE MICROSERVICE ARCHITECTURE OF SOFTWARE

The article is devoted to the study of the technology of building a microservice architecture that allows to preserve the modularity and independence of the deployment of Enterprise application components. Microservice architecture is an approach to the creation of an application, implying a refusal from a single, monolithic structure. That is, instead of executing all the restricted application contexts on the server with the help of intraprocess interactions, several small applications are used, each of which corresponds to some limited context. And these applications run on different servers and interact with each other over the network, for example, via HTTP. It describes the process of modeling services, the ways to integrate microservices between each other, their basic properties. It considers the algorithm for partitioning a single-piece application (a monolith) into parts, methods for deploying microservices. Also, various types of microservice architecture and methods of their connection were explored. Analyzing the obtained information on the subject of the article, advantages and disadvantages were revealed in comparison with monolithic systems. The result of the research is to obtain scientific and practical knowledge for the design and development of high-performance Enterprise applications, as well as to obtain good application performance with minimal labor costs.

Key words: microservice architecture, design, Enterprise applications.

Введение. С ростом объема кода и функциональности программного продукта, возникает необходимость управления сложностью приложения. Хорошо продуманная архитектура и правильное разбиение приложения на модули помогают справляться с поставленной задачей. Вариантом реализации архитектуры может быть монолитное приложение, когда вся или большая часть бизнес-задач имеет одну кодовую базу. Актуальным решением в настоящее время является построенное на микросервисах приложение, в котором общая бизнес-задача разбита на отдельные части, каждая из которых имеет отдельное приложение (микросервис) со своей кодовой базой.

Основная часть. Микросервисная архитектура – это архитектура, в которой имеются следующие сервисы.

1. Небольшие сервисы. Сервис в микросервисной архитектуре не может разрабатываться больше чем одной командой. Одна команда разрабатывает не более шести сервисов, при этом каждый сервис решает одну бизнес-задачу, и ее способен понять один человек, в противном случае сервис необходимо разделить.

2. Узко сфокусированные сервисы. Сервис в микросервисной архитектуре решает только одну бизнес-задачу. Его можно отделить от

системы, и, дописав некоторую логику, использовать как отдельный продукт.

3. Слабосвязанные сервисы. Изменение одного сервиса не требует изменений в другом.

4. Высокосогласованные сервисы. Класс или компонент сервиса содержит все нужные методы для решения поставленной задачи только в этом классе или компоненте [1].

Характеристики микросервисной архитектуры. Вне зависимости от того когда нужно применить микросервисный подход (при разработке приложения с нуля или разбиении существующего монолитного приложения) любая микросервисная архитектура должна обладать определенными свойствами. Первое из них – это разбиение через сервисы. Здесь существуют два понятия: библиотека и сервис, которые взаимодействуют по сети. В мире микросервисов библиотеки – это компоненты, которые подключаются к программе и вызываются ею в том же процессе, в то время как сервисы – компоненты, выполняемые в отдельном процессе и связывающиеся друг с другом с помощью REST или RPC. Таким образом, если имеется приложение, которое состоит из нескольких библиотек, работающих в одном процессе, любое изменение в этих библиотеках потребует переразвертывания всего приложения. Но если приложение разбито на несколько сервисов, то изменения потребуют переразвертывания только изменившегося сервиса.

Группировка команд по бизнес-задачам. Как правило, при разработке монолита разработчики группируются по интересам: группа разработчиков внешнего представления, группа разработчиков внутренней реализации и группа проектирования базы данных. Когда команды разбиты таким образом, то они выстраивают определенные границы, каждая со своей стороны. А это мешает прохождению бизнес-задачи, т. к. она проходит и слой внешнего представления, и слой внутренней реализации, и слой базы данных, и обратно. Поэтому для правильного создания микросервисной архитектуры необходимо придерживаться закона Конвея, который гласит, что структура вашего приложения повторяет структуру вашей команды, т. е. необходимо формировать команды согласно бизнес-задачам. Это позволит создавать микросервисы, которые будут полностью соответствовать всем потребностям поставленной бизнес-задачи [2].

«Умные» сервисы и простые коммуникации. Есть разные варианты взаимодействия сервисов. Чаще всего в механизмы передачи данных помещается существенная часть логики. Тогда получается очень «умный» канал передачи и «глупые» сервисы. В микровервисной архитектуре вся бизнес-логика должна находиться в сервисах, а по каналу передачи отсылаются только данные, он никак не связан с бизнес-задачей.

Децентрализованное управление данными. При проектировании микросервисной архитектуры каждый сервис имеет только свою базу данных. Пример представлен на рис. 1.

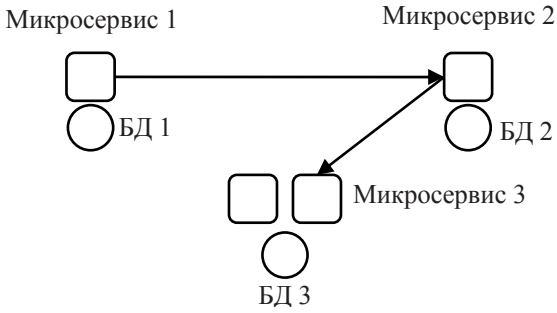

Рис. 1. Схема экземпляров баз данных микросервисов

Единственный случай, когда разные сервисы могут использовать одно хранилище – если эти службы представляют собой точные копии друг друга, но базы данных друг с другом не взаимодействуют.

Автоматизация развертывания и мониторинга. Это свойство, пожалуй, самое важное, которое должно присутствовать в микросервисной архитектуре. В данном контексте подразумевается наличие автоматического развертывания и непрерывной интеграции, т. е. автоматической доставки кода и его проверки, убеждаясь в том, что код скомпилирован и тесты с ним проходят без сбоев. Так как сервисы могу отказать в любое время, очень важно иметь непрерывный мониторинг, что даст возможность быстро обнаружить неполадки и, если возможно, автоматически восстановить работоспособность сервиса. Также мониторинг полезен для отслеживания рабочих показателей микросервисов.

Проектирование под отказы. Также важная характеристика, означающая, что начиная строить микросервисную архитектуру, необходимо исходить из предположения, что сервисы должны работать при отказе отдельных сервисов. Таким образом, необходимо реализовывать дополнительную логику для того, чтобы система была устойчива к таким сбоям.

Взаимодействие с микросервисами. Когда мы говорим о микросервисах, то всегда стоит вопрос о том, как клиенты приложения на основе микросервисов получают доступ к отдельным услугам. Существуют различные патерны взаимодействия с микросервисами. Наиболее общеизвестной является модель API Gateway, представленная на рис. 2.

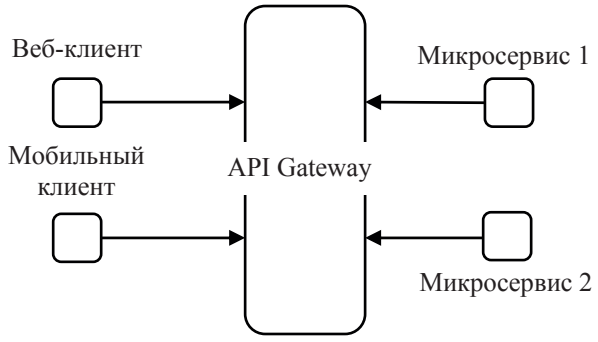

Рис. 2. Схема взаимодействия API Gateway

API Gateway – дополнительный сервис, задача которого – собирать бизнес-вызовы к целевым сервисам. То есть вместо того, чтобы обращаться к каждому сервису, клиент обращается к шлюзу, а тот в свою очередь перенаправляет запрос к нужному сервису. Также API Gateway осуществляет поддержку разных API для каждого клиента.

Теперь рассмотрим схемы связи микросервисов между собой. Первый подход называется «Обнаружение службы» (Service Discovery, SD). В данном случае сервисы знают друг о друге и общаются напрямую с использованием технологии RPC. Данный шаблон имеет два вида реализации. Первый тип – это Server-Side SD. Данный паттерн имеет следующие свойства: каждый экземпляр предоставляет удаленный API, такой как HTTP / REST с определенным адресом (хост и порт); количество экземпляров служб и их адресов изменяется автоматически; виртуальным машинам и контейнерам обычно назначаются динамические ip-адреса. На рис. 3 представлена схема взаимодействия микросервисов согласно модели Server-Side SD.

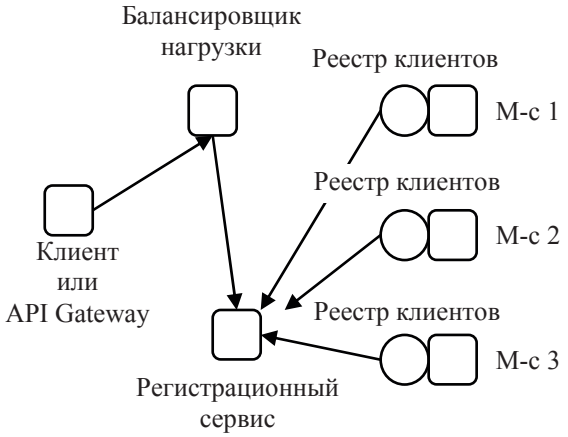

Рис. 3. Схема взаимодействия Server-Side SD

При данном типе клиент взаимодействует не напрямую с конкретным микросервисом, а с балансировщиком нагрузки. Дополнительно применяется регистрационный сервис. Его задача – хранить регистрационные данные сервисов, и делать это он может по-разному: может опрашивать сервисы, брать данные из внешнего конфигурационного файла и т. д. Балансировщик нагрузки берет все данные у регистрационного сервиса. Таким образом, задача балансировщика – получать данные о местоположении сервисов из регистрационного сервиса и передавать запросы к ним.

Другой реализацией SD модели является Client-Side SD. Данному паттерну характерны те же свойства, что и предыдущему типу связи микросервисов. На рис. 4 представлена схема взаимодействия согласно модели Client-Side SD.

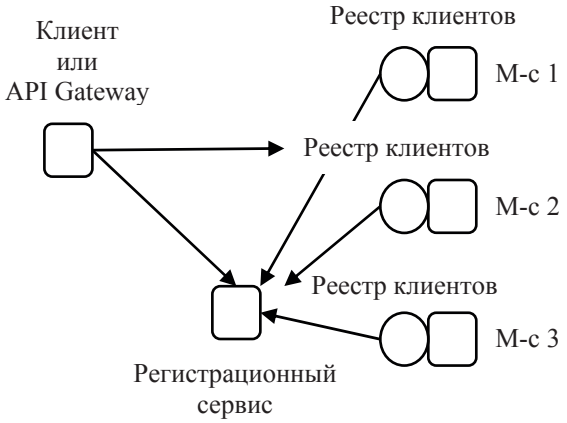

Рис. 4. Схема взаимодействия Client-Side SD

При данном виде взаимодействия отсутствует балансировщик нагрузки и клиент обращается напрямую к регистрационному сервису. Этот подход лучше тем, что выполняется на один запрос меньше, из-за чего взаимодействие происходит быстрее.

Вторым способом связи микросервисов является «Шина сообщений» (Message Bus, MB). В данном случае микросервисы подписываются на определенные события, при возникновении которых микросервисы отреагируют на это соответствующим образом. Данный способ нужен, чтобы один сервис мог сообщить другому, что у него изменилось состояние, чтобы другие сервисы могли на это среагировать.

Таким образом, при анализе микросервисной архитектуры были выявлены следующие преимущества: модульное деление команды – разбиение команды по бизнес-задачам позволяет увеличить эффективность разработки приложений за счет компетентных в данной области бизнес-задачи участников команды; высокая доступность – при нарушении работы одного из сервисов остальные продолжают работать; неограниченный выбор технологий – для определенной задачи есть возможность подобрать нужную технологию, а также есть возможность просто

протестировать ее на каком-нибудь сервисе, что не повлияет на другие сервисы; независимое развертывание – простые сервисы проще разворачивать, и меньше вероятность отказа системы.

Но, в свою очередь, есть ряд недостатков: поддержка конечной согласованности – необходимость работать с отложенными данными, что требует постоянной доступности приложения; сложность операционной поддержки – необходимость поддержки непрерывного развертывания, непрерывной интеграции и автоматического мониторинга.

Заключение. Таким образом, использование микросервисного подхода позволяет создавать отказоустойчивые приложения, которые без труда можно масштабировать без тесной привязки к целевой платформе, а также гибко конфигурировать сервисы и развертывать их в различных средах. Помимо всего прочего было установлено, что использование микросервисной архитектуры значительно сократит время непрерывного развертывания и поставки по сравнению с монолитными архитектурами за счет слабой связности между компонентами всей системы целиком. К сожалению, добавляется сложность в организации взаимодействия сервисов между собой, что приводит к падению общей производительности всей системы за счет сетевых задержек и пропускной способности каналов связи.

Литература

1. Ньюмен С. Создание микросервисов. СПб.: Питер, 2016. 304 с.
2. Microservice Architecture [Электронный ресурс]. Режим доступа: http://microservices.io/. Дата доступа: 30.01.2017.

References

1. Newman S. Building Microservices. USA, O'Reilly Media Publ., 2016. 304 p.
2. Microservice Architecture [Electronic resource]. Available at: http://microservices.io/(accessed 30.01.2017).

Информация об авторе

Шитько Андрей Михайлович – аспирант. Белорусский государственный технологический университет (220006, г. Минск, ул. Свердлова, 13а, Республика Беларусь). E-mail: andrew.shitsko@gmail.com

Information about the author

Shytska Andrei Mikhaylovich – graduate student. Belarusian State Technological University (13a, Sverdlova str., 220006, Minsk, Republic of Belarus). E-mail: andrew.shitsko@gmail.com

Поступила 20.04.2017