ВЫГОДЫ ПЕРЕХОДА ОТ МОНОЛИТНОЙ К МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ ПРИЛОЖЕНИЯ

Кабарухин А.П.

Кабарухин Алексей Павлович - старший инженер сопровождения разработки,
Акционерное общество "Нэксайн", г. Москва

Аннотация: *распространение облачных сервисов к началу 2010-х годов привело к разочарованию разработчиков в классическом варианте архитектуры приложений. В качестве альтернативы предложили архитектуру микросервисов как распределенную систему простейших и легко заменяемых модулей. Они должны работать, как рабочие у конвейера: выполнять одну элементарную функцию и передавать задачу дальше. При этом микросервисы выстраиваются не иерархично, а симметрично. Массовый переход с монолитной на микросервисную архитектуру (MSA, Micro Service Architecture) связан с развитием облачных сервисов и необходимостью обеспечить максимально оперативное обновление и модернизацию сервисов в соответствии с меняющимися бизнес-задачами. В статье, раскрываются ключевые особенности, преимущества, инструменты и сложности микросервисного подхода.*
Ключевые слова: *монолитная архитектура, микросерверная архитектура, преимущества.*

THE BENEFITS OF MOVING FROM A MONOLITHIC TO A MICROSERVICE APPLICATION ARCHITECTURE

Kabarukhin A.P.

Kabarukhin Aleksei Pavlovich - Senior DevOps Engineer,
NEXIGN, JSC, MOSCOW

Abstract: *the proliferation of cloud services by the early 2010s led to disillusionment of developers with the classical variant of application architecture. As an alternative, microservices architecture was proposed as a distributed system of simple and easily replaceable modules. They should work like workers on a conveyor belt: perform one elementary function and pass the task on. In this case, microservices are not built hierarchically, but symmetrically. Mass transition from monolithic to microservices architecture (MSA, Micro Service Architecture) is associated with the development of cloud services and the need to ensure maximum prompt updating and upgrading of services in accordance with changing business tasks. The article describes key features, advantages, tools and complexities of microservice approach.*
Keywords: *monolithic architecture, microserver architecture, advantages.*

УДК 004.03

Стремительное развитие и распространение сетевых облачных сервисов к началу 2010-х годов привело к разочарованию в классическом, так называемом монолитном варианте архитектуры приложений. Из-за сложности отдельных модулей, зачастую представляющих собой целые программные системы, а также из-за необходимости обеспечивать совместимость между ними посредством стандартных протоколов, внесение любых изменений и дополнений стало нетривиальной задачей, отнимающей слишком много времени.

Довольно долгое время в разработке информационных систем преобладал метод создания "монолитных" приложений, в которых изначально не выделялись ни пользовательский уровень, ни уровень бизнес-логики. Затем появились подходы, в которых стали выделять несколько уровней (соответствующих монолитной архитектуре клиент-сервер):

1) пользовательские интерфейсы;
2) уровень бизнес-логики системы;
3) уровень базы данных.

Первый уровень называется Front-End, второй и третий - Back-End. Соответственно, разработчики условно делятся на две группы Front-End разработчиков и Back-End разработчиков. Основным недостатком монолитных приложений является необходимость сборки всего приложения после внесения изменений, что имеет следующие последствия [1]:

- значительное увеличение времени на развертывание системы и внесение изменений;
- сильная связь, делающая разработчиков зависимыми друг от друга, поскольку отказ одного компонента приводит к отказу всей системы;
- большая часть кода не позволяет легко переходить на новые технологии, внесение изменений также затруднено, в результате приложения устаревают;
- сложно масштабировать приложения;
- при внесении изменений значительно возрастает количество ошибок, что приводит к частым сбоям системы [2].

Приложения, построенные на монолитной архитектуре (или просто монолиты) – это не обязательно приложения с одной функцией. Они содержат отдельные классы и функции, однако те крепко связаны друг

с другом. Изъятие одного модуля неизбежно вызывает изменения в работе всей системы. Отсюда – ряд значимых недостатков:

1. Чтобы внести одно изменение, необходимо пересобрать всё приложение.

2. Невозможно масштабировать отдельный модуль – придётся переделывать всё приложение.

3. Разработка ограничена изначально выбранным набором языков программирования, фреймворков и других инструментов. Хочется использовать что-то альтернативное - извините, у нас такого нет, работайте с тем, что есть.

4. Сломался один модуль – работать, скорее всего, перестанет весь сервис.

5. Сложная структура связей между модулями замедляет выход обновлений. Каждое требует серьезного тестирования на баги и регрессии кода (новые ошибки в уже протестированном функционале). Соответственно, в одном обновлении появляется целая кипа изменений.

6. Поначалу приложение имеет понятную структуру и быстро развивается. А потом понеслось: длинный и сложный код, рамки модулей постепенно стираются, между компонентами возникает все больше неочевидных взаимосвязей [3].

В том числе в классической монолитной архитектуре невозможно одновременное использование разных языков программирования. Например, если весь проект написан на Java, а модуль для бухгалтерии выгоднее и актуальнее написать на Go, то осуществить это будет проблематично.

Отказоустойчивость монолитного приложения обеспечить сложнее, чем микросервисного, поскольку при сбое единой базы данных все его модули становятся неработоспособными. Выход из строя базы данных одного из микросервисов не так сильно повлияет на остальные [4].

В качестве ответа на этот вызов была предложена архитектура микросервисов как распределенная система простейших и легко заменяемых модулей, выполняющих по возможности единственную элементарную функцию. При этом микросервисная система имеет симметричную, одноранговую, а не иерархическую организацию, что снимает необходимость в сложной организации взаимосвязей. Сервисы связываются между собой и с клиентами с использованием лёгких протоколов, например, через HTTP или текстовыми сообщениями. В результате создаётся система, простая в развёртывании и модернизации с функциями автоматической разработки и обновления.

К 2021 году микросервисная архитектура в центре внимания, причём, не только специалистов: о ней пишут в блогах, в соцсетях, обсуждают в прессе и на различных конференциях. Об успешном внедрении микросервисов заявляют представители Amazon, Google, Netflix и Twitter. В России об опыте перехода на микросервисы сообщали крупные банки, а также, например, «М.Видео-Эльдорадо» и «МегаФон» [5].

При этом в сообществе программистов всё чаще звучат голоса скептиков, не считающих микросервисы чем-то принципиально новым. По их мнению, это просто реализация сервис-ориентированной архитектуры (SOA) на более низком уровне. Как бы то ни было, микросервисная архитектура имеет очевидные преимущества, особенно в сфере Agile-разработки и развёртывания сложных приложений корпоративного уровня.

Микросервисная архитектура — это подход к созданию приложения, который предполагает отказ от единой, монолитной структуры. То есть вместо выполнения всех ограниченных контекстов приложения на сервере посредством внутрипроцессных взаимодействий используется множество небольших приложений, каждое из которых соответствует какому-то ограниченному контексту [6].

По состоянию на 2021 год микросервисная архитектура считается наиболее естественным подходом для разработки облачных приложений. Таковые состоят из множества слабо связанных и независимо разрабатываемых мелких модулей — сервисов. Для этих сервисов, как правило, характерны три свойства:

- у них есть собственный стек, включающий базу и модель данных;
- они взаимодействуют друг с другом посредством сочетания REST API, потоков событий и брокера сообщений;
- модули приложения подбираются исходя из конкретных потребностей бизнеса [7].

С точки зрения бизнеса и чисто организационных задач, преимущества и выгоды переходы к микросервисам сводятся к трём основным:

- лёгкость обновления кода;
- разные команды могут использовать разные стеки для разных модулей;
- компоненты могут масштабироваться независимо друг от друга, что снижает затраты и стоимость масштабирования всего приложения в целом в тех случаях, если узким местом выступает лишь какая-то одна функция [8].

Микросервисная архитектура приложения родилась из монолитной архитектуры, когда та стала сложной и неудобной в работе. Главное отличие микросервисов от монолита – в использовании специализированных более простых программ (модулей) при выполнении сценария приложения. Тогда как в монолитной архитектуре использовались внутрипроцессные взаимодействия. И, что самое удобное, модули могут находиться на разных серверах и их взаимодействие происходит через сеть по протоколонезависимой технологии.

Микросервисная архитектура имеет ряд преимуществ перед монолитной:

– симметричная архитектура (в монолитных приложениях – иерархическая);

– взаимозаменяемость микросервисов;

– независимость микросервисов друг от друга;
– организация модулей вокруг отдельных функций;
– написание микросервисов с помощью любых программных средств, оптимальных для каждого конкретного модуля, при этом они хорошо «понимают» друг друга благодаря интерфейсу. Создание интерфейса является наиболее сложной задачей;
– микросервисы вызываются только потребителем, но не друг другом.

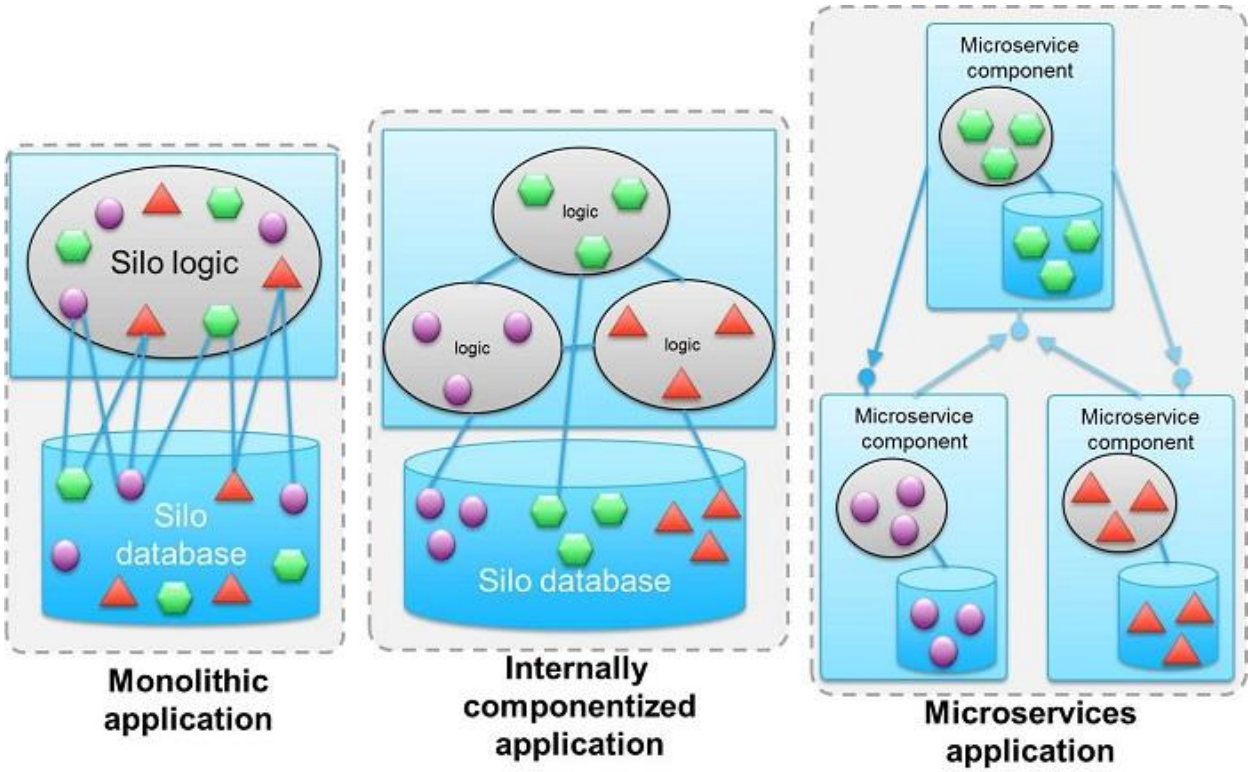

Рис. 1. Различие микросервисной и монолитной архитектуры

Чтобы уточнить отличия микросервисов от других архитектур, отметим, что микросервисное приложение состоит из множества мелких независимых и слабо связанных между собой сервисов, в то время как в монолите все его компоненты тесно взаимосвязаны и работают как единый сервис. Помимо прочего, это значит, что если какой-то один процесс в приложении с монолитной архитектурой становится более востребованным, приходится масштабировать всё приложение в целом. Сбой в каком-то одном процессе может поставить под угрозу всю систему. Наконец, такая сложность ограничивает возможности модернизации и затрудняет внедрение новых идей.

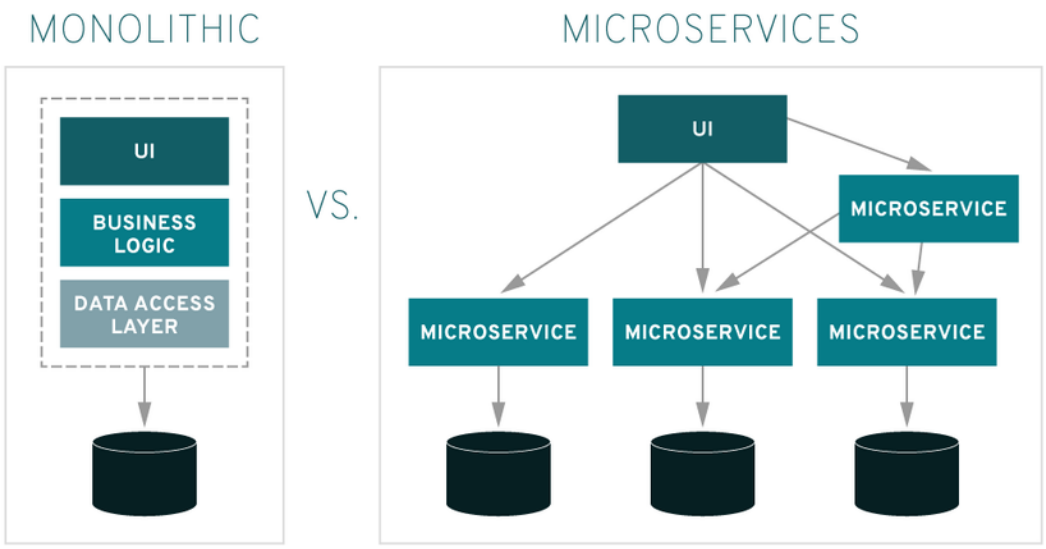

Рис. 2. Различие микросервисной и монолитной архитектуры с точки зрения сервисов

Монолитная архитектура (Monolithic) отличается тесной взаимосвязью между компонентами, включая бизнес-логику (Business Logic) и слой доступа к данным (Data Access Layer), и выступает как единый сервис. В микросервисной архитектуре (Microservices) клиент через общий пользовательский интерфейс (UI) получает доступ к отдельным слабо связанным между собой микросервисам (Microservice).

Микросервисы в современном понимании имеют следующие особенности:

- легкость и простота: каждый модуль выполняет уникальную функцию и имеет небольшой размер;
- конечность, атомарность;
- гибкость: модуль можно легко изменить, добавив в его работу необходимые опции;
- взаимозаменяемость;
- полиморфизм;
- комбинирование микросервисов для реализации разных функций.

Микросервисная архитектура устранила недостатки монолитного ПО, обеспечив:

- изоляцию и минимизацию изменений;
- ускорение разработки;
- возможность легко подстраивать ПО под структуру бизнеса [9].

В итоге отметим, что в монолитной архитектуре функции инкапсулируются в одном приложении. Когда монолитное приложение небольшое по размеру и имеет всего несколько функций, он обладает своими сильными сторонами, например, простота разработки, тестирования, развертывания и масштабирования. Если, например, необходимо больше вычислительных возможностей, достаточно продублировать весь монолит, и при небольшом размере системы будут лишь небольшие накладные расходы.

Слабые стороны этой системы проявляются, когда монолит начинает обновляться и развиваться, увеличиваясь в размере и количестве функций. Как только это произойдет, недостатки монолитной архитектуры перевесят его преимущества.

Например, если требуется больше вычислительных возможностей, масштабирование большого монолита приведет к большим накладным расходам по сравнению с небольшим монолитом. Еще один недостаток, вызванный большим размером монолита, заключается в том, что разработка может замедлиться и, как следствие, стать препятствием для непрерывного развертывания из-за более длительного времени запуска.

У микросервисов есть множество преимуществ по сравнению с монолитной архитектурой, что приводит к миграции существующих монолитных архитектур в архитектуры на основе микросервисов.

Ключевыми преимуществами микросервисов являются отказоустойчивость и ремонтопригодность. Также разделение системы на независимые и самостоятельно развертываемые службы помогает командам разработчиков тестировать свои службы и вносить изменения независимо от других разработчиков, что упрощает распределенную разработку. Небольшой размер микросервисов способствует тому, что код становится более понятным для разработчиков [10].

Еще одно ключевое преимущество микросервисов - масштабируемость. В микросервисной архитектуре каждый микросервис может быть развернут на разных машинах, каждый с разным уровнем производительности, и может быть написан на более подходящем языке программирования. Если, например, существует узкое место в конкретном микросервисе, его можно поместить в контейнер и запустить на нескольких параллельно работающих хостах, без необходимости развертывания системы на новой мощной машине.

Следующее существенное преимущество микросервисов - возможность их замены. Поскольку отдельные микросервисы имеют небольшой размер, их легко заменить на более совершенную реализацию или даже удалить. Команды разработчиков, использующие подходы с использованием микросервисов, не сталкиваются с большими трудностями с полным переписыванием микросервисов, когда это необходимо.

Важным преимуществом микросервисов является отказоустойчивость. Отказ микросервиса обычно не влияет на всю систему. Тем не менее, неисправные микросервисы также могут быть быстро перезапущены.

Еще одно ключевое преимущество микросервисов - надежность, так как данная технология разработки программного обеспечения подразумевает создание больших систем из простых и небольших компонентов с чистыми интерфейсами (микросервисами) [11].

Переход на микросервисную архитектуру приложений позволит компании обеспечить условия для роста – гибкость, масштабируемость, отказоустойчивость систем.

Список литературы / References

1. *Холодок Д.А., Пресняцкий В.Ю., Лецук Р.А.* Микросервисы как архитектурный стиль / / Образование и наука в России и за рубежом, 2019. Ц (62). С. 2Ц 218.
2. *Гольчевский Ю.В., Ермоленко А.В.* Актуальность использования микросервисов при разработке информационных систем // Вестник Сыктывкарского университета. Серия 1. Математика. Механика. Информатика, 2020. № 2 (35). [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/aktualnost-ispolzovaniya-mikroservisov-pri-razrabotke-informatsionnyh-sistem/ (дата обращения: 14.01.2022).

3. Микросервисная архитектура: теория и практика. [Электронный ресурс]. Режим доступа: https://vc.ru/dev/295980-mikroservisnaya-arhitektura-teoriya-i-praktika/ (дата обращения: 31.01.2022).
4. *Опольский В.* Переход от монолита к микросервисам: когда опыт разработчиков трансформируется в бизнес-результат. [Электронный ресурс]. Режим доступа: https://www.it-world.ru/tech/practice/173784.html/ (дата обращения: 31.01.2022).
5. Переход на микросервисы: опыт «М.Видео-Эльдорадо» и «МегаФона». [Электронный ресурс]. Режим доступа: https://mcs.mail.ru/blog/perekhod-na-mikroservisy-opyt-m-video-eldorado-i-megafona/ (дата обращения: 31.01.2022).
6. *Осипов Дмитрий Борисович*. Проектирование программного обеспечения с помощью микросервисной архитектуры // Вестник науки и образования. 2018. №5 (41). URL: [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/proektirovanie-programmnogo-obespecheniya-s-pomoschyu-mikroservisnoy-arhitektury/ (дата обращения: 14.01.2022).
7. What are Microservices? IBM Cloud Education, 2021. [Электронный ресурс]. Режим доступа: https://www.ibm.com/cloud/learn/microservices/ (дата обращения: 31.01.2022).
8. *Нечай О.* Все говорят о микросервисной архитектуре приложений. Чем она хороша и как на нее перейти? [Электронный ресурс]. Режим доступа: https://www.tadviser.ru/index.php/
9. Принципиальное отличие микросервисной архитектуры от монолитной. [Электронный ресурс]. Режим доступа: https://hawkhouse.ru/blog/kogda-opravdano-ispolzovanie-mikroservisnoj-arhitektury/ (дата обращения: 31.01.2022).
10. *Мартин Р.* Чистая архитектура. Искусство разработки программного обеспечения. СПБ: Питер, 2018.
11. *Радостев Д.К., Никитина Е.Ю.* Стратегия миграции программного кода из монолитной архитектуры в микросервисы // Вестник Пермского университета. Серия: Математика. Механика. Информатика. 2021. №2 (53). [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/strategiya-migratsii-programmnogo-koda-iz-monolitnoy-arhitektury-v-mikroservisy/ (дата обращения: 14.01.2022).