УДК 004.75+004.4

ТРАНСФОРМАЦИЯ ПРИЛОЖЕНИЙ ОТ МОНОЛИТНОЙ К МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ

Ширшов А. А., студент

Белорусский государственный университет информатики и радиоэлектроники,
Институт информационных технологий,
г. Минск, Республика Беларусь

Парамонов А.И. – канд. техн. наук, доцент, зав. каф. ИСиТ

Аннотация. Современные программные средства требуют быстрой развёртки в облачных хранилищах, легкого и удобного масштабирования, что невозможно при использовании монолитной архитектуры. Обозначены основные проблемы монолитной архитектуры и способы их возможных решений. Предлагаются пути перехода к микросервисной архитектуре.

Ключевые слова. Микросервис, монолит, архитектура, программное обеспечение, облачные вычисления, API, масштабирование.

Введение. Потребность в микросервисной архитектуре приложений возникла совсем не спонтанно. Цифровая трансформация сегодня активно входит во все отрасли экономики, что влечет изменение не только технологических процессов бизнеса, но и самих инструментов трансформации. ИТ-инфраструктура становится с каждым годом всё сложнее, обрастает новыми системами и сервисами. Объемные приложения с большим количеством взаимодействующих между собой функциональных модулей, большие команды разработчиков, создающих проект, требования к скорейшему выпуску и своевременному обновлению программного обеспечения (ПО), необходимость периодического масштабирования отдельных модулей – всё это предпосылки к тому, чтобы серьезно задуматься о разработке приложений на микросервисной архитектуре с нуля или перестраивании на микросервисы уже существующей монолитной структуры [1].

В то же время, отношение бизнеса и даже ИТ-сообщества к микросервисной архитектуре разработки приложений сегодня не столь однозначно и возможно даже противоречиво, как в свое время к RPA [2]. Энтузиасты называют монолитные приложения атавизмами уходящей эпохи и спешно переходят на микросервисы. Однако, следует отметить, что не всем, кто принял решение о переходе на микросервисы, удается окупить вложения в них.

Основная часть. Монолитная архитектура – это традиционная модель ПО, которая представляет собой единый модуль, работающий автономно и независимо от других приложений. Монолитом часто называют нечто большое и неповоротливое, и эти два слова хорошо описывают монолитную архитектуру для проектирования программного продукта. Монолитная архитектура – это отдельная большая вычислительная сеть с единой базой кода, в которой объединены все бизнес-задачи. Упрощенная схема такой архитектуры приведена на рисунке 1.

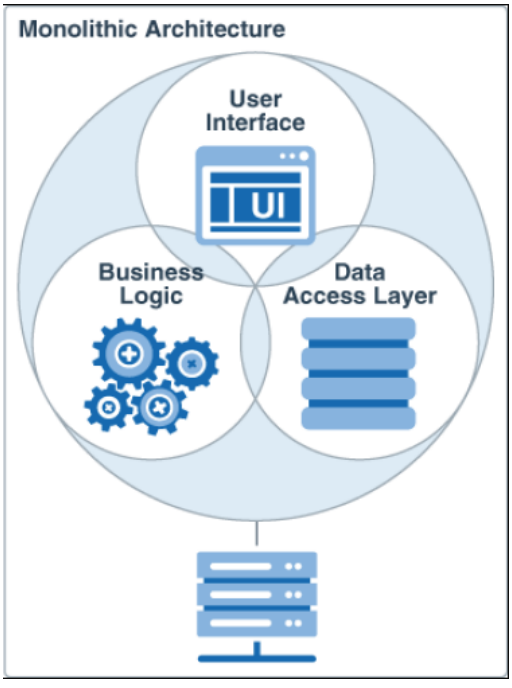

Рисунок 1 – Упрощенная схема монолитной архитектуры

«Монолиты» удобно использовать на начальных этапах проектов, чтобы облегчить развертывание и не тратить слишком много умственных усилий при управлении кодом. Это позволяет сразу выпускать все, что есть в монолитном приложении. Однако, чтобы внести изменения в такое приложение, необходимо обновить весь стек через базу кода, а также создать и развернуть обновленную версию интерфейса, находящегося на стороне службы. Это ограничивает работу с обновлениями и требует много времени. Проблема при разработке и поддержке ПО с монолитной архитектурой может возникнуть, в частности, в момент слияния веток проекта (merge conflict), когда несколько программистов одновременно модифицируют один и тот же участок кода. Этот недостаток некритичен, но увеличивает время разработки, особенно если в приложение добавляется новый модуль: например, функциональность доставки товара со склада интернет-магазина дополняется модулем закупки или аналитики.

Проблема масштабируемости приложения появляется, когда количество его пользователей начинает расти, причем выстреливший стартап зачастую демонстрирует очень активную динамику. Некоторые модули монолита могут иметь специальную архитектуру, которая будет обеспечивать высокие нагрузки, но масштабировать в любом случае придется всё приложение целиком. В монолите мы не можем отдельно поменять архитектуру только высоконагруженных модулей [3]. Эту ситуацию можно образно сравнить, к примеру, с развивающейся сетью магазинов, у которых единая служба бухгалтерии: в случае монолита при масштабировании приложения нам пришлось бы в каждый из этих магазинов добавить своего бухгалтера, при этом с ростом количества покупателей нагрузка на этот отдел осталась бы прежней.

Микросервисная архитектура — распространенный подход к разработке ПО, когда приложение разбивается на небольшие автономные компоненты («микросервисы») с четко определенными интерфейсами. Именно эта архитектура характерна для Cloud-native приложений, которые сейчас популярны благодаря преимуществам облачных сред и их возможностям для бизнеса. В микросервисной архитектуре слабо связанные сервисы взаимодействуют друг с другом для выполнения задач, относящихся к их бизнес-возможностям. Микросервисы в значительной степени получили свое название из-за того, что сервисы здесь меньше, чем в монолитной среде. Но нужно понимать, что приставка «микро» — об указании на бизнес-возможности (функционал), а не на размер. Упрощенная схема такой архитектуры представлена на рисунке 2.

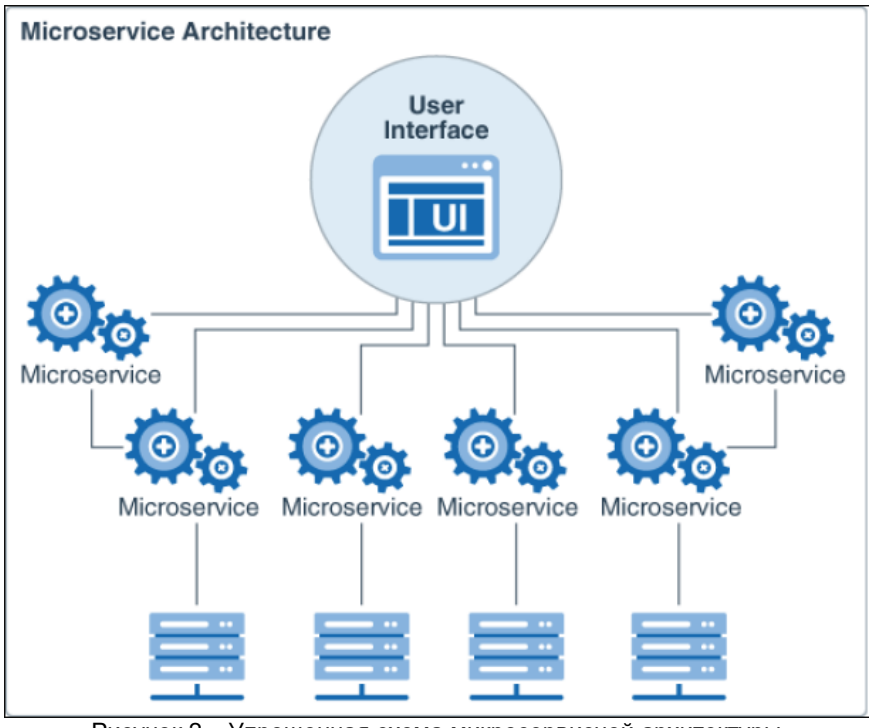

Рисунок 2 – Упрощенная схема микросервисной архитектуры

Желание заменить монолитную архитектуру на микросервисы основано на статистике, которая показывает, что между монолитными и микросервисными системами существует значительная разница в производительности, которая порой достигает 79,2% [4]. Статистика [5] показывает, что:

— 68% компаний уже используют микросервисы в производстве и разработке;

— 36% крупных компаний, 50% средних компаний, 44% малых компаний используют микросервисы в производстве и разработке;

— 26% компаний рассматривают микросервисы, но еще не начали их внедрять;

— команды, которые перешли на микросервисы, отчитались о 13-кратном увеличении частоты выпусков ПО.

Стоит отметить, что при переходе с «монолита» на «микросервисы» интеграция сталкивается с некоторыми проблемами. Обычно это означает, что разработчикам нужно восстанавливать прежние версии с нуля. По этой причине весь процесс может также привести к определенным сложностям, которые связаны с тратой ресурсов (времени и денег), отсутствием требований или фактического состояния (только код), возникновением технологических) вопросов (по технологиям, подходам, внедрению и т.п.). Ключевые моменты в сравнении архитектур представлены в таблице 1.

Таблица 1 – Сравнение монолитной и микросервисной архитектуры.

Монолит	Микросервисы
Типичное приложение состоит из базы данных, клиентского пользовательского интерфейса и серверного приложения.	В основном получили свое название из-за того, что сервисы здесь меньше, чем в монолитной среде.
Все части ПО объединены, так что управлять всеми его функциями можно в одном месте.	Сервисы связаны слабо, взаимодействуют между собой и автономно выполняют бизнес-задачи.
Компоненты ПО взаимосвязаны и взаимозависимы.	Существует много единиц развертывания. Каждый сервис разрабатывается и разворачивается независимо друг от друга.
Приложения обеспечивают быструю связь между программными компонентами благодаря общему коду и памяти.	Меньшие размеры сервисов помогают, когда дело доходит до времени компиляции, выполнения и времени запуска тестов.

Заключение. Трансформация архитектуры приложений и переход от монолита к микросервису предполагает два основные пути решения:

1. Переписать приложение с нуля на микросервисную архитектуру и использовать лучшие практики написания кода. На это решение потребовалось бы более года разработки, а также команда из пяти и более человек.

2. Внедрить один микросервис, который покрыл бы небольшую, но наиболее часто используемую часть функциональности, из-за которой возникают самые большие (или частые) проблемы с производительностью. На это потребовалось бы около двух месяцев разработки и команда всего из двух человек.

Для успешного начала необходимо внедрить логику в микросервис, создав API Getaway, благодаря которому можно было легко общаться обеим сторонам — клиентам существующих проектов и бизнес-клиентам. Далее постепенно переносить бизнес-логику в отдельные микросервисы и подключать их к Getaway, чтобы агрегировать вызовы микросервисов.

Список использованных источников:

1. Переход от монолита к микросервисам: когда опыт разработчиков трансформируется в бизнес-результат. [Электронный ресурс] Мир цифровых и информационных технологий. Режим доступа: https://www.it-world.ru/tech/practice/173784.html. (дата обращения: 03.04.2023).

2. Бадмаева А.Д., Перерва О.Л. Риски внедрения технологии RPA на наукоемкое предприятие // Научный результат. Экономические исследования. 2020. №3. [Электронный ресурс] Режим доступа: https://cyberleninka.ru/article/n/riski-vnedreniya-tehnologii-rpa-na-naukoemkoe-predpriyatie (дата обращения: 03.04.2023).

3. Переход от монолита к микросервисам: история и практика. [Электронный ресурс] Блог компании Райффайзен Банк. Режим доступа: https://habr.com/ru/company/raiffeisenbank/blog/458404/. (дата обращения: 03.04.2023).

4. От устаревшего монолита - к микросервисам: как решиться на миграцию. [Электронный ресурс] Режим доступа: https://highload.today/blogs/ot-ustarevshego-monolita-k-mikroservisam-kak-reshitsya-na-migratsiyu/. (дата обращения: 03.04.2023).

5. Отчет O'Reilly о внедрении микросервисной архитектуры в 2020 году. [Электронный ресурс] Режим доступа: https://www.businesswire.com/news/home/20200716005101/en/O%E2%80%99Reilly%E2%80%99s-Microservices-Adoption-in-2020-Report-Finds-that-92-of-Organizations-are-Experiencing-Success-with-Microservices. (дата обращения: 03.04.2023).

UDC 004.75+004.4

APPLICATION TRANSFORMATION FROM MONOLITHIC TO MICROSERVICES ARCHITECTURE

Shirshov A. A.

Institute of Information Technologies of the Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Paramonov A.I. – Candidate of Engineering Sciences, Associate Professor

Annotation. Modern software tools require fast deployment in cloud storages, easy and convenient scaling, which is impossible when using a monolithic architecture. The main problems of monolithic architecture and ways of their possible solutions are outlined. Ways of transition to microservice architecture are offered.

Keywords. Microservice, monolithic architecture, software, cloud computing, API, scaling.