ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ПОМОЩЬЮ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ

Осипов Д.Б. Email: Osipov641@scientifictext.ru

Осипов Дмитрий Борисович – бакалавр,
кафедра вычислительной техники,
Университет информационных технологий механики и оптики, г. Санкт-Петербург

Аннотация: *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. В статье исследованы различные типы микросервисной архитектуры и способы их связи. Детально рассмотрены ключевые особенности применения микросервисной архитектуры, а также сформулированы рекомендации по использованию микросервисов при разработке программного обеспечения.*

Ключевые слова: *микросервисы, микросервисная архитектура, монолитная архитектура, разработка программного обеспечения, децентрализация данных.*

DESIGNING SOFTWARE USING A MICRO-SERVICE ARCHITECTURE

Osipov D.B.

Osipov Dmitry Borisovich - Bachelor,
DEPARTMENT OF COMPUTER ENGINEERING,
UNIVERSITY OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS, SAINT-PETERSBURG

Abstract: *microservice architecture is an approach to creating applications, meaning the rejection of a single, monolithic structure. That is, instead of executing all the limited contexts, using several small applications, each of which corresponds to some limited context. In the article various types of microservice architecture and ways of their connection are investigated. Interaction with the microservice in the development of software.*

Keywords: *micro services, micro service architecture, monolithic architecture, software development, data decentralization.*

УДК 004.422.81

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.

Приложения с монолитной архитектурой.

При классическом подходе к разработке ПО, как правило, используется монолитная архитектура приложения, когда оно разрабатывается как единый блок. При разработке промышленных приложений, как правило, используется трехуровневая клиент-серверная архитектура. При таком подходе приложение состоит из трех основных частей: клиентская часть (тонкий клиент, предоставляющий весь функционал посредствам HTML страниц в браузере, написанных с использованием различных javascript библиотек или фреймворков), базы данных (БД), состоящей из множества таблиц в единой, как правило, реляционной системе управления базами данных (СУБД), и серверной части. Серверная часть приложения обрабатывает HTTP-запросы, выполняет бизнес-логику, получает и изменяет данные в БД, а также выбирает необходимые HTML страницы и данные для отображения и отправляет их в качестве HTTP-ответа клиенту (браузеру). Это серверная часть приложение представляет собой монолитную структуру, и любые изменения в ней требуют создание и развертывание новой версии данного приложения.

Монолитные сервера является естественным подходом к построению такого рода систем. На рисунке один представлена структурная схема монолитного блока.

Рис. 1. Структурная схема монолитного блока

Вся логика запроса обрабатывается в одном процессе [2, с. 164], позволяя разработчику использовать основные особенности языка, на котором разрабатывается данное приложение, а именно классы, функции, пространства имен и т.д. С некоторой осторожностью, такие приложения могут быть запущены и протестированы на рабочей машине разработчика, развернуты с использованием конвейеров развертывания (deployment pipelines), чтобы гарантировать, что изменения должным образом протестированы и развернуты на «боевых» серверах. Горизонтальное масштабирование (рис. 2) монолитных приложений осуществляется с помощью запуска нескольких его экземпляров с использованием балансировщиков нагрузки.

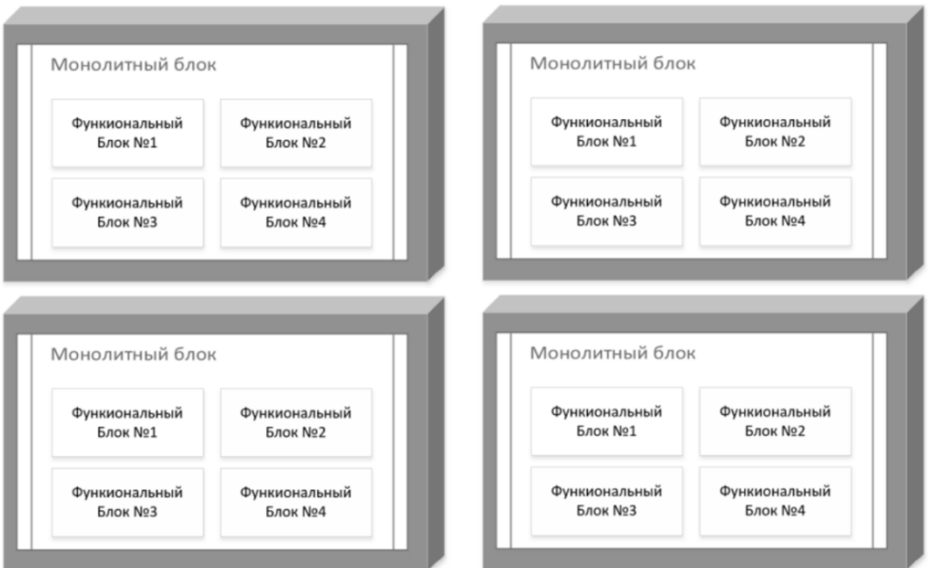

Рис. 2. Структурная схема горизонтального приложения с монолитной архитектурой

Приложения с микросервисной архитектурой

Эти три главных недостатка привели к появлению микросервисного подхода при проектировании ПО. Микросервисы - это небольшие, автономные, совместно работающие сервисы [1, с. 23]. Микро-сервисная архитектура - это подход к разработке единого приложения, как набор небольших сервисов, каждый из которых запущен в собственном процессе (рис. 3), в то время как коммуникация с другими сервисами осуществляется с помощью каких-либо «легковесных» механизмов, например, с помощью HTTP. Эти сервисы разрабатываются с учетом бизнес-требований и независимо развертываются с помощью полностью автоматизированных механизмов развертывания.

Рис. 3. Структурная схема сервисов в приложении с микросервисной архитектурой

Помимо того, что сервисы независимо развертываются и масштабируются (рис. 4), каждый сервис также обеспечивает жесткую границу модуля, делая при этом возможным разработку сервисов с использованием разных языков программирования разными командами разработчиков [1, с. 27].

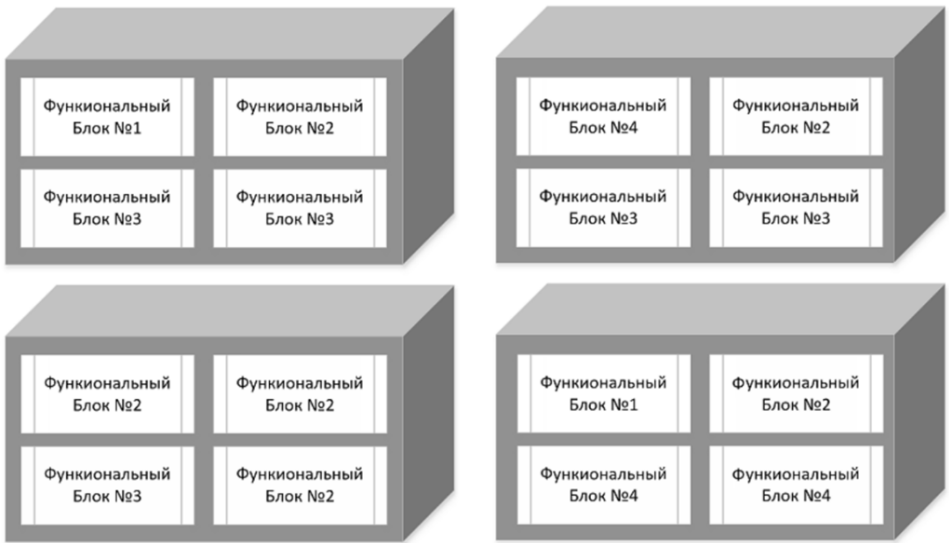

Рис. 4. Структурная схема горизонтального масштабирования приложения с микросервисной архитектурой

Нельзя сказать, что существуют какие-либо формально определенные правила для разработки приложений с микросервисной архитектурой, однако далее будут рассмотрены наиболее важные, на мой взгляд, из них.

Создание компонентов через сервисы

Под компонентом подразумевается единица ПО, которая может быть независимо обновлена или заменена. Одной из основных причин использования сервисов как компонентов является то, что сервисы можно развертывать независимо. Это значит, что если имеется приложение, состоящее из нескольких компонентов (функциональных блоков) в одном процессе, то изменение любого отдельного компонента приводит к необходимости повторного развертывания всего приложения. Но если приложение декомпозировано на сколько сервисов, каждый из которых содержит один компонент (функциональный блок), то изменение одного из них не приводит к необходимости повторного развертывания всего приложения, а требует лишь развертывания измененного сервиса. Изменения сервиса могут изменить его интерфейсы, что затронет механизмы его взаимодействия с другими сервисами, однако цель хорошей архитектуры микросервиса - минимизировать такие кардинальные изменения [3].

Другим следствием использования микросервисов как компонентов является более явный компонентный интерфейс. Так как большинство языков программирования поддерживают определение публичного интерфейса только на уровне документирования, то инкапсуляция у компонента может быть нарушена, что приводит к слишком высокой связанности компонентов. Микросервисы позволяют избежать это посредствам использования удаленных вызовов, что в свою очередь также имеет свои недостатки, главным из которых является то, что такие вызовы являются более дорогостоящими, чем вызовы внутри процесса, что требует более тщательной проработке API у микросервисов.

Кроме того, как было отмечено ранее, поскольку при разработке микросервисов могут использоваться кардинально разные технологии, это снимает ограничение, накладываемые на классические команды разработчиков, участвующих в разработке приложений с монолитной

архитектурой, позволяя балансировать человеческие ресурсы в зависимости от востребованности того или иного микросервиса.

Децентрализация

При разработке приложения с микросервисной архитектурой, все данные должны управляться децентрализованно. На абстрактном уровне это означает, что концептуальная модель хранения данных в разных развернутых вариантах системы не должна быть одинакова.

Помимо децентрализации функциональных блоков, микросервисы также децентрализуют решения в области хранения данных [5]. На рисунке 5 представлена структурная схема хранения данных в приложении с монолитной архитектурой.

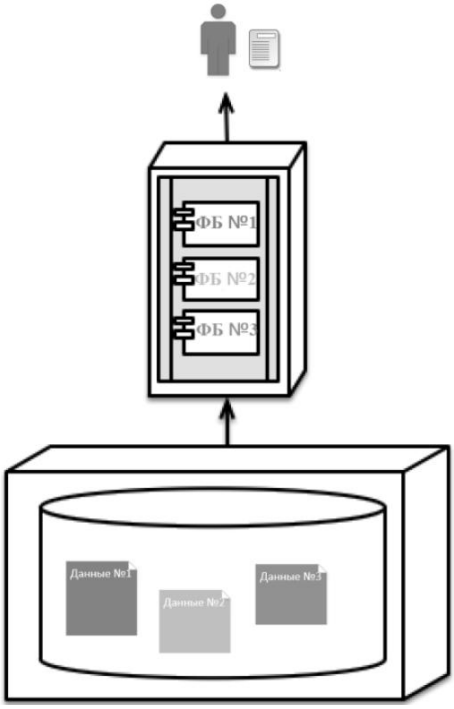

Рис. 5. Структурная схема модели хранения данных в приложении с монолитной архитектурой

В то время, как в приложениях с монолитной архитектурой используют единую СУБД для целого приложения, в приложениях с микросервисной архитектурой предпочтительно, чтобы каждый микросервис взаимодействовал со своей собственной СУБД либо разные экземпляры сервиса использовали одинаковую СУБД (рис. 6).

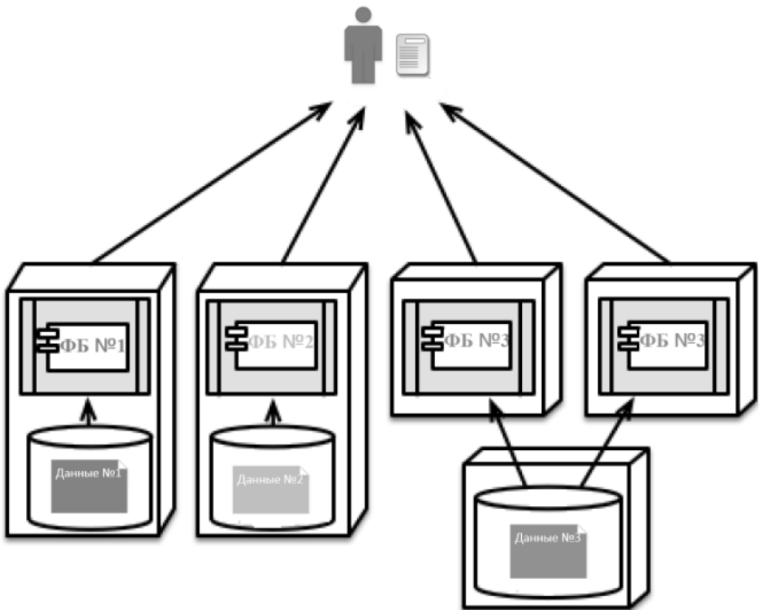

Рис. 6. Структурная схема модели хранения данных в приложении с микросервисной архитектурой

Децентрализация ответственности за данные в рамках микросервисов имеет последствия при управлении обновлениями данных. Общий подход при работе с обновлениями заключается в использовании транзакций для того, чтобы гарантировать консистентность при обновлении нескольких ресурсов, однако такой подход налагает дополнительные временные связи, которые проблематично поддерживать для нескольких микросервисов, поэтому выбор средств для управления консистентностью является отдельной задачей для команд разработчиков.

Заключение

Активно развивающееся в последнее направление в разработке программного обеспечения, а именно разработка с применением микросервисной архитектуры, является очень перспективным для внедрения. Главными преимуществами такого решения, по сравнению с монолитной архитектурой, является экономия ресурсов, как человеческих и материальных - в процессе разработки, так и вычислительных - в процессе масштабирования. Однако стоит также учитывать сложности, которые возникают при применении микросервисов, главным образом связанные с усложнением механизмов транзакций, а также необходимостью внедрения систем для автоматизированного мониторинга и логирования работы приложения.

Список литературы / References

1. *Ньюмен Сэм*. Создание микросервисов. СПб.: Питер, 2016. 304 с.: ил. (Серия «Бестселлеры O'Reilly»).
2. *Таненбаум Э., Стеен М. ван*. Распределенные системы. Принципы и парадигмы. СПб.: Питер, 2003. 877 с.: ил. (Серия «Классика computer science»).
3. Martinfowler. [Электронный ресурс]: Microservices. a definition of this new architectural term. Режим доступа: https://martinfowler.com/articles/microservices.html/ (дата обращения: 22.05.2018).
4. Хабрахабр. [Электронный ресурс]: Microsoft Azure - Azure Service Fabric и архитектура микросервисов. Режим доступа: https://habrahabr.ru/company/mailru/blog/320962/ (дата обращения: 06.05.2016).
5. MSDN Magazine Blog. [Электронный ресурс]: Архитектура микросервисов. Режим доступа: https://msdn.microsoft.com/ru-ru/magazine/mt595752.aspx/ (дата обращения: 22.05.2018).