Миграция учета остатков предприятия розничной торговли с монолитной архитектуры на микросервисы

И. В. Илларионов, email: igor.illarionov@gmail.com[1]
М. Д. Квасова, email: kvasova@gmail.com[1]

[1] Воронежский государственный университет

Аннотация. *В данной работе рассматривается процесс миграции информационной системы предприятия розничной торговли с монолитной архитектуры на микросервисную.*

Ключевые слова: *микросервисы, монолитная архитектура, SAP ERP, торговля.*

Введение

Каждый продукт, владельцы которого нацелены на рост и развитие, приходит к тому состоянию, когда добавлять новые функциональные возможности к уже существующим становится настолько дорого, что стоимость новой функциональности превосходит все возможные выгоды от ее использования.

Существует несколько подходов к решению данной проблемы. Можно разбить приложение на несколько частей, зеркалировать систему или использовать микросервисы. Каждый из подходов имеет свои неоспоримые преимущества и различные недостатки.

Самый применяемый сегодня подход подразумевает деление одного большого куска кода (монолитного приложения) на несколько проектов - микросервисов, каждый из которых отвечает за свои определенные функции. Благодаря такому делению на микросервисы вся система приносит бизнесу и клиенту большую ценность, а также может проще и быстрее адаптироваться к внешним и внутренним изменениям.

При планировании перехода должны быть выбраны способы и методы взаимодействия микросервисов друг с другом и с внешними сервисами, определены сроки и критерии успешности перехода.

1. Анализ решаемой задачи

На рис. 1 представлена общая схема монолитной и микросервисной архитектур. Далее они будут рассмотрены подробно.

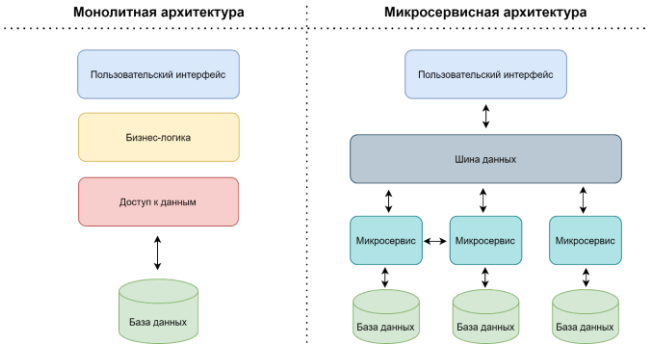

Рис. 1. Сравнение монолитной и микросервисной архитектур

Монолитная архитектура считается традиционным способом построения приложений. Это приложение, представленное в качестве единого компонента на одной платформе. Обычно такое решение включает в себя следующие уровни:

– пользовательский интерфейс – уровень представления, с которым взаимодействует пользователь;
– бизнес-логику – основная логика работы приложения;
– доступ к данным – модель данных, используемая сущностями в рамках бизнес-логики приложения;
– базу данных.

Самым большим преимуществом монолитной архитектуры считается скорость разработки, а также простота создания приложения на основе одного кода.

В монолитной архитектуре разработчики вынуждены придерживаться одной технологии, которая может иметь различные ограничения. Принятие любой новой технологии означало бы переписывание всего приложения, что чрезвычайно дорого и отнимает много времени. Следовательно, обновление - почти невыполнимая задача, кроме того, разработчики не могут извлечь выгоду из недавно выпущенных, более продвинутых фреймворков и языков. Это отсутствие гибкости становится особенно большой проблемой, когда используемая в настоящее время структура устаревает.

Обычно монолитные приложения имеют одну большую кодовую базу и не имеют модульности. Если необходимо изменить или удалить некоторую часть приложения, изменения вносятся сразу во весь стек, что является дорогостоящим и требует много времени. Вместе с тем

после внесения изменения необходимо полное повторное тестирование всего приложения.

Монолитное приложение может масштабироваться только в одном измерении. Первый из возможных вариантов – масштабирование за счет увеличения числа транзакций благодаря запуску дополнительных копий приложений. Однако такая архитектура имеем ограничения по масштабированию с увеличением объема данных. Недостаток заключается в том, что все копии приложения будет иметь доступ ко всем данным, что делает увеличивает потребление памяти и трафик ввода-вывода. Кроме того, различные компоненты приложения имеют разные требования к ресурсам. В монолитной архитектуре мы не можем масштабировать каждый компонент независимо.

По мере разработки продукты с монолитной архитектурой увеличиваются в объеме, их структура становится более размытой, а сложность системы растет. При передаче системы от команды команде, при постоянной смене разработчиков, отсутствии комментариев, тестов и документации разобраться в системе становится трудно. Следовательно, систему поддерживать с течением времени все сложнее и сложнее.

Монолитная архитектура может быть полезной, если:

- необходимо как можно быстрее запустить продукт в эксплуатацию;
- разрабатывает и поддерживает приложение одна несменная команда;
- приложение обладает ограниченной функциональностью и используется не часто.
- монолитная архитектура перестает быть подходящей, когда нам нужно:
- независимая масштабируемость различных компонентов домена;
- различные компоненты или модули, которые должны быть написаны на разных языках программирования;

В этот момент нам нужно разделить наш монолит на различные части, применив другой архитектурный стиль.

2. Микросервисная архитектура

Микросервисная архитектура (или микросервисы) – это стиль проектирования, который разбивает приложение на отдельные сервисы с разными функциями. [1]

Архитектура микросервисов структурирует приложение как набор небольших независимых служб, взаимодействующих через API.

Преимущество разработки приложения, которое следует архитектуре микросервисов, заключается в том, что команда разработчиков самостоятельно определяет инструменты реализации, имея широкий спектр возможностей для различных технологий, таких как операционные системы, платформы, языки программирования. Каждая служба имеет свою собственную функцию, кодовую базу, процессы, жизненный цикл и базу данных. Инструментальные средства могут быть выбраны в зависимости от того, что подойдет лучше для решения той или иной конкретной задачи. Это позволяет разрабатывать, развертывать, поддерживать и обновлять каждый из строительных блоков отдельно, не влияя на остальную часть приложения, в отличие от взаимозависимой монолитной архитектуры.

В результате команды разработчиков могут удовлетворять постоянно меняющиеся потребности бизнеса, быстро обновляя каждую службу или создавая новые компоненты приложения, вместо того чтобы перестраивать или повторно развертывать все это.

Каждая отдельная часть приложения может быть построена независимо из-за изолированности компонентов. Повышенная гибкость позволяет разработчикам обновлять компоненты системы, не отключая приложение. Новые функции могут быть добавлены по мере необходимости, не дожидаясь запуска всего приложения. Следовательно, релиз одной единицы не ограничен выпуском другой, которая еще не завершена.

Все сервисы микросервисной архитектуры построены как разные модули, что обеспечивает возможность масштабировать их независимо друг от друга.

При внесении изменений в один из микросервисов, его обновлении работа всей системы не будет нарушена. Вместе с тем если в каком-то из процессов возникнет ошибка, остальные процессы не будут затронуты и могут быть запущены, поскольку все сервисы работают изолировано друг от друга. Например, если в какой-то момент времени не доступна оплата с помощью кредитной карты, пользователь также может просматривать и добавлять их в корзину, оформлять заказы с оплатой при получении.

При выборе микросервисной архитектуры должны быть определены все блоки и четкие протоколы (правила) их взаимодействия. Для каждой пары взаимодействующих микросервисов должны быть согласованы формат обмена. Этот обмен происходит по сети, что может приводить к сетевым задержкам. Так, при правильной сборке монолитные приложения могут быть более производительны, чем система, состоящая из микросервисов, поскольку монолитные

приложения имеют более быструю связь между программными компонентами благодаря общей памяти и кодовой базе.

В сравнении с монолитными системами в микросервисной существует больше элементов для мониторинга, которые разработаны с использованием различных языков программирования. Поэтому один из недостатков микросервисов заключается в их сложности. Разделение приложения на независимые микросервисы влечет за собой больше единиц управления. Этот тип архитектуры требует тщательного планирования, огромных усилий, командных ресурсов и навыков.

3. Предметная область

Далее будет рассмотрен проект миграции архитектур для предприятия, представляющего собой онлайн-магазин продуктов питания, предоставляющий возможность доставки заказа до двери или в пункт выдачи заказов. Заказ может быть оформлен как через браузер так и через мобильное приложения на платформе iOS или Android. Ассортимент магазина имеет более 40000 товаров, имеется возможность доставки продукты в течение 2 часов. В компании работает более 3700 сотрудников, среди которых:

- команда разработки: разработчики, тестировщики, аналитики данных, бизнес-аналитики;
- работники склада: комплектовщики, фасовщики, кладовщики, контролеры качества, кассиры, грузчики;
- сотрудники административно-хозяйственного отдела;
- сотрудники отдела транспортной логистики: водители-курьеры, водители-экспедиторы, логисты;
- специалисты по безопасности;
- заведующие экономикой и финансами.

Компания имеет собственный автомобильный парк и более 800 машин, оборудованных специальными камерами для сохранения свежести продуктов.

4. Исходная архитектура

На рис. 2 представлена архитектура монолитного приложения предприятия, используемая ранее до возникновения следующих проблем:

- при внесении изменений в часть программного кода необходимо вносить изменения в другие его части, причем объем сопутствующих правок не может быть спрогнозирован;

– добавление новой функциональности требует большого количества времени на разработку, стабилизацию и тестирование;
– невозможно увеличить пропускную способность отдельных частей приложения.

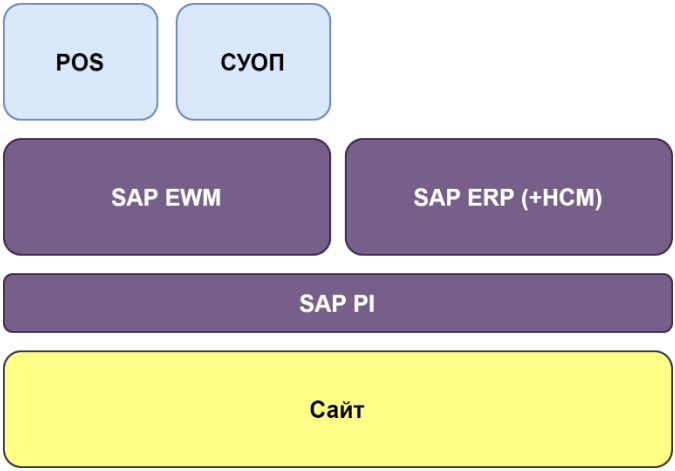

Рис. 2. Монолитная архитектура приложения

5. Целевая архитектура

Целевая архитектура представлена функционально независимыми единицами, выполняющими одну из функций монолита со своим стеком технологий и базой данных.

Архитектура, представленная на рис. 3, состоит из тех же элементов, что и монолит: Сайт, SAP PI, SAP EWM, POS, СУОП, но вместо SAP ERP выступает SAP S/4 HANA и отдельная система SAP HCM.

Сайт содержит в себя функциональность, которая описывает все процессы управления заказом, типы доставки, промоакции, все процессы, происходящие в личном кабинете клиента, включая аутентификацию, авторизацию, регистрацию.

Apache Kafka – система обмена сообщениями, хранилище для больших объемов данных, которое обеспечивает коммуникацию между сайтом и микросервисами, а также связь микросервисов между собой. В Apache Kafka консистентно хранятся данные в той последовательности, в которой они менялись. Такой подход обеспечивает надежность и

легкость отслеживания состояний. В Apache Kafka данные записываются быстрее, чем в обычную базу данных.

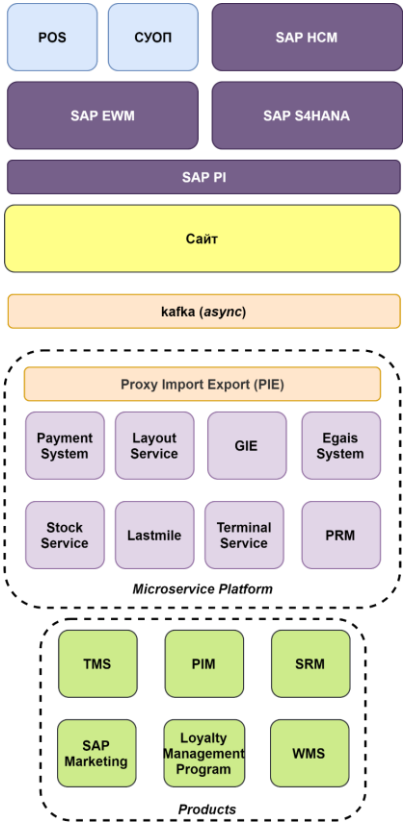

Рис. 3. Целевая система

Цель сервиса – обеспечение полноценного процесса управления автопарком и рейсами по клиентской доставке заказов, а также рейсами для хозяйственных нужд компании. Микросервис представляет собой веб-интерфейс для планирования курьеров, транспортных средств и других ресурсов доставки.

Сервис покрывает 3 крупных блока:

– планирование ресурсов доставки
– рабочее место начальника автоколонны;
– табулирование и учет рабочего времени;

– состояние ресурсов (водителей и транспортных средств).
– управление рейсами и заказами в нем

6. Учет остатков

Вся бизнес-логика по работе с остатками расположена на PHP backend. Товарные остатки обновляются в течение дня раз в 20 минут сообщением, включающим в себя изменения по остаткам. В случае, если по любым причинам обновление по остаткам не было отправлено, в течение дня накапливаются расхождения по ним.

Для нивелирования расхождений в товарных остатках существует механизм выравнивания. Выравнивание остатков подразумевает выгрузку общего файла с остатками (z_if_xi_send_ostat.xml) из ERP на FTP-ресурс и дальнейшее считывание этого файла со стороны PHP-backend. Время начала формирования общего файла с остатками на стороне ERP – 22:30. Время выравнивания остатков на стороне PHP-backend – 00:01.

В связи с увеличением количества складов и товарных единиц время создания общего файла с остатками увеличилось до 2,5-4 часов. Следовательно, файл будет выложен на FTP будет после 00:00 следующего дня, и загруженный файл будет считан PHP-backend только на следующие сутки. Например, 22.12.2020 в 22:30 начал формироваться файл на стороне ERP. Выложен файл на FTP 23.12.2020 в 00:15. Загрузка данного файла и выравнивание остатков по нему произойдёт только 24.12.2020 в 00:01.

В результате расхождения остатков на сайте (витринах) с актуальными остатками на складах и в ERP составляют временной интервал более суток.

Для оперативного устранения данной проблемы принято решение выполнить доработку механизма выравнивания остатков на стороне PHPbackend. Расхождения по остаткам провоцируют факт того, что заказ не может быть полностью скомплектован.

Если в заказе есть товары, которые не удалось скомплектовать, пользователю отправляется сообщение на СМС и e-mail. В противном случае, если все товары были скомплектованы, пользователю никаких сообщений по результатам комплектации не уходит. Если каких-то из товаров нет в наличии проверяется наличие товаров альтернатив.

В августе 2020 года процент неподтверждённых комплектаций товаров в заказах составил 0,3%. Число неподтвержденных комплектаций товаров в заказах приблизительно 2382.

Самый нагруженный интервал времени - с 15:00 до 17:00.

Максимальное наблюдаемое количество заказов в минуту - 400, с учётом ограничения базы данных. Максимально возможное количество заказов в минуту при текущих ресурсах доставки - 1 000 заказов в минуту. Для расчётов взята статистика за 16.08.2020

Всего в течение суток 251 запрос:

в среднем 10,46 RPH

в среднем 0,17 RPM

в среднем 0,003 RPS

Среднее число позиций в заказе равно 53 позициям. Для расчётов взята статистика за сентябрь 2020 года. Максимально наблюдаемое число запросов в каталогах и в корзине RPM = 2 000. Допускаем, что 50% из этих запросов – фототоваров и баннеры.

Следовательно, RPM = 1 000.

7. Показатели эффективности

В течение всего процесса роллаута микросервиса StockService производился мониторинг остатков из ERP и из StockService. Следующий этап не начинался, пока 99% процентов товаров не имели расхождение по остаткам равное 0 в течение 7 дней. Вместе с этим отслеживалось количество принятых товаров со стороны ERP через Kafka (рис. 4). Остатки передавались с периодичностью 20 минут. Если по каким-то причинам остатки не были переданы, механизм выравнивания отрабатывал в 100% случаев.

Артикул товара	ID склада	Остатки из ERP	Остатки из StockService	Разница
3428895	4000	12	12	0
3428895	5000	12	12	0
3428895	7000	12	12	0
3428896	100	0	0	0
3428896	4000	19	19	0
3428896	5000	24	24	0

Рис. 4. Разница остатков на сайте между ERP и StockService

Процент неподтверждённых комплектаций товаров в заказах составил 0,01%.

Заключение

В результате проделанной работы был спроектирован микросервис учета остатков, сохранивший все полезные функциональные возможности при этом нивелирующий недостатки учета остатков в монолитной архитектуре.

Учет остатков, реализованный в микросервисной архитектуре, уменьшил процент неподтверждённых комплектаций товаров в заказах в 30 раз благодаря отправке данных по остатках не через заданные промежутки времени, а по событиям, меняющим количество остатков таким как заказ, внутреннее перемещение товара, списание, поставка.

Полученная система может быть масштабирована для работы с большим числом наименований товаров, складов, на которых они хранятся.

Список литературы

1. Ричардсон, К. Микросервисы. Паттерны разработки и рефакторинга / К. Ричардсон; СПб.: Питер, 2019. – 546 с.
2. Ньюмен, С. От монолита к микросервисам: Эволюционные шаблоны для трансформации монолитной системы / С.Ньюмен; Пер. с англ. Андрея Логунова. - СПб.: БХВ-Петербург, 2021. – 272 с.
3. Нархид, Н. Apache Kafka. Потоковая обработка и анализ данных / Н. Нархид, Г. Шапиро, Т. Палино; Пер. с англ. И. Пальти. – СПб.: Питер, 2019. – 320 с.
4. Кочер, П. Микросервисы и контейнеры Docker / П. Кочер; Москва: ДМК Пресс, 2019. – 240 с.
5. Моуэт, Э. Использование Docker / Э. Моут; М.: ДМК Пресс, 2017. – 356 с.