ОСОБЕННОСТИ ДОБАВЛЕНИЯ МЕХАНИЗМА РАБОЧЕГО ПРОЦЕССА В ПРИЛОЖЕНИЯХ НА БАЗЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ, УПРАВЛЯЕМОЙ СОБЫТИЯМИ

Р. Дандан, *магистрант*
Московский государственный технический университет им. Н.Э. Баумана (Россия, г. Москва)

DOI:10.24412/2500-1000-2022-5-2-26-31

Аннотация. *В ходе исследования обсуждались недостатки в системах, разработанных с микросервисной архитектурой управляемой событиями, цель состоит в том, чтобы предложить решение, которое поможет сохранить видимость процессов экосистемы. Был предложен механизм рабочего процесса вместе с архитектурой добавления этого механизма в изучаемую систему электронной коммерции.*

***Ключевые слова*:** *микросервисы, архитектура, управляемая событиями, оркестровка*, хореография*, ограниченный контекст.*

Микросервисы – это подход создания приложения в качестве набора сервисов, которые работают независимо и взаимодействует друг с другом с помощью упрощенных механизмов. Эти сервисы должны быть развернуты независимо и автоматически, и они могут быть реализованы с использованием различных языков программирования и технологий хранения данных.

Модель публикации-подписки используется в архитектуре, управляемой событиями. Служба взаимодействует с другими компонентами экосистемы, публикуя события, которые управляются брокером сообщений и используются другими компонентами, подписавшимися на событие. Эта модель обеспечивает низкую связь между компонентами, в результате чего система становится более адаптируемой и масштабируемой.

Несмотря на то, что для каждого микросервиса требуются регистрация, мониторинг и предупреждение, они могут быть недостаточно при рассмотрении архитектуры, управляемой событиями. Сложность отслеживания потока данных экосистемы затрудняет понимание того, произошел ли сбой, на каком этапе система ее инициировала, и, в конечном счете, правильно ли завершены бизнес -процессы.

Поскольку это также зависит от бизнес - критериев и существующей архитектуры, в настоящее время нет универсального программного обеспечения, которое может обеспечить такое понимание и контроль над экосистемой.

В данной статье построена система электронной коммерции с использованием микросервисов и архитектуры, управляемой событиями, и работающая на контейнерах Docker. Обсуждается, как реализовать масштабируемую и надежную архитектуру микросервисов, управляемых событиями, и как получить представление о том, что происходит во всей экосистеме. Это сделано путем предоставления инструмента, облегчающего анализ текущей ситуации в экосистеме, а также позволяющего просматривать данные и, возможно, влиять на них.

Границы контекста и бизнес-процессы

Процесс, по сути, представляет собой набор задач, которые необходимо выполнить для достижения определенного результата. Процессы есть везде. У сотрудников есть процесс электронной почты, который включает в себя методы быстрой расстановки приоритетов. Как владелец бизнеса, учитываются сквозные процессы компании, такие как выполнение заказов клиентов.

Фокус в DDD гарантирует, что термины могут быть определены согласованно в пределах одного ограниченного контекста,

даже если они могут означать разные вещи в разных контекстах.

Например, «заказ» – это понятие, известное в разных контекстах, но значение которого может различаться. В контексте «Корзина» заказ относится к заполнению корзины покупателем. Этот заказ можно легко изменить. В контексте «выполнения заказа» заказ – это точная инструкция о том, что брать и отправлять, и она неизменна.

Еще одним примером является концепция клиента. Большинство контекстов связаны с этой концепцией, но они связаны с различными ее аспектами: для выполнения заказа требуется только личность клиента, для доставки требуется только адрес, а для оплаты требуются только платежные реквизиты. Даже если они используют одни и те же термины, в разных контекстах могут быть разные определения клиентов и заказов.

DDD помогает определить границы контекста. Микросервис может быть ограниченным контекстом или его частью. Другими словами, ограниченный контекст может создать более одной микросервисы. Это решение полностью зависит от потребности микросервиса в масштабируемости и независимости.

По следующим причинам контексты и границы оказывают значительное влияние на разработку бизнес-процессов, и наоборот [1]:

- многие сквозные бизнес-процессы будут затрагивать несколько контекстов в течение своего жизненного цикла. Тем не менее, следует избегать разработки всезнающей модели процесса, которая требует внутреннего знания различных контекстов для функционирования. Модели процессов представляют собой логику предметной области и поэтому должны содержаться в службе, реализующей соответствующий контекст. А поскольку модели процессов особенно хорошо видны многим заинтересованным сторонам, очень важно, чтобы они применяли общий язык своего контекста;

- моделирование и обсуждение бизнес-процессов, особенно на сквозном уровне, помогает находить кандидатов на границы контекста, понимать возникающие обязанности и, таким образом, окончательно принимать решение о границах контекста;

- наличие возможностей механизма рабочего процесса, доступных в контексте, позволяет признать долгосрочный характер многих проблем.

Оркестровка против хореографии

Оркестровка и хореография являются двумя шаблонами описания бизнес-процессов в качестве взаимодействующих служб: в виде последовательного потока выполнения сервисов и в виде правил их взаимодействия [2].

Событие – это то, что произошло. Компонент *A* генерирует событие, но у него нет никаких ожиданий относительно того, что должно произойти на основе этого события. Компонент *B* может или не может принять решение реагировать на событие.

Напротив, компонент *A* также может отправить команду компоненту *B*. Это означает, что *A* хочет, чтобы *B* что-то сделал. Есть явное намерение, и *B* не может просто проигнорировать эту команду.

Термины оркестровки и хореографии определяются следующим образом [1]:

- коммуникация, управляемая командами = оркестровка;

- коммуникация, управляемая событиями = хореография.

В этом смысле оркестровка означает координацию действий или задач. Как правило, если есть компонент, который координирует один или несколько других компонентов, то это оркестровка. Это означает, что компонент отправляет команды.

Чем более общим является компонент, тем меньше от него требуется изменений, если другим службам необходимо взаимодействовать с ним. В этом случае обычно предпочтительнее API управляемая командами.

Оркестровка не вводит временную связь; синхронная связь делает. Проблема может быть решена путем перехода на асинхронность. Оркестровка не зависит от протоколов связи [1].

В хореографии компоненты напрямую взаимодействуют друг с другом в зависимости от событий, чтобы что-то сделать.

Определенная степень связанности доменов неизбежна, если необходимо взаимодействовать различным компонентам, но может быть определена, находится ли она на стороне отправки или на стороне получателя. Это решение определяет, будете ли вы использовать события или команды.

Следует понимать обязанности, возложенные на различные компоненты. Это поможет не только установить хорошие границы контекста, но и принять решение о событиях, а не о командах. Если ответственна отправляющая сторона, то она заботится о том, чтобы что-то происходило, а значит, нужен командный стиль. Если отправляющую сторону это не волнует, а получатель отвечает за принятие мер, обычно можно использовать стиль, основанный на событиях.

Обязанности никогда не фиксированы. Это очень сильно связано с проектированием границ микросервисов. Решение о том, использовать ли события (хореографию) или команды (оркестровку), является просто результатом принятия во внимание обязанностей. В хорошей архитектуре есть оба стиля общения: оркестровка и хореография [1].

Проблемы архитектуры, управляемой событиями

Основными причинами создания систем, управляемых событиями, являются стремление к автономии команды и необходимость создания несвязанных систем. Когда связь управляется событиями, сервисы отправляют события, не зная, какие другие сервисы их подхватят. Сервисы также реагируют на события, не зная, откуда они пришли. Каждый сервис просто подписывается на темы, соответствующие его цели, и получает уведомление, когда происходит событие, связанное с этой темой.

Цепочка событий – это ряд подписок на события, которые реализуют логический поток или бизнес-процесс, так что эти подписки на события не являются независимыми.

В этом случае задачи должны выполняться в определенном порядке, например, чтобы убедиться, что платеж завершен до фактической отправки заказа. Но нет места, где можно было бы понять или даже проконтролировать эту последовательность событий.

События могут упростить добавление новых функций, но это происходит за счет того, что становится намного сложнее вносить изменения в цепочку событий, что может повлиять на несколько компонентов [1].

Цепочки событий трудно понять, в основном из-за отсутствия видимости этих цепочек. Поскольку взаимодействие микросервисов децентрализовано, оно разбросано по нескольким базам кода. Что приводит к потере зрения или непониманию системы в целом (к пониманию общей картины) [3].

В хореографии диагностировать и исправить ошибку становится сложно из-за отсутствия контекста. Сбой в одном микросервисе не может быть легко отслежен до того места, где возникла цепочка событий. Если данные искажены, может потребоваться много усилий, чтобы понять, почему они там. И может быть неясно, какие следующие шаги в настоящее время заблокированы этими инцидентами, что очень затрудняет обходные пути.

Отсутствие видимости сквозных бизнес-процессов, охватывающих несколько микросервисов, является основной проблемой. Существует очевидная необходимость в понимании того, как гарантировать мониторинг и управление связью, которая происходит между различными микросервисами, чтобы реализовать бизнес-возможности. Организации и команды должны гарантировать постоянную работу своей экосистемы не только за счет наблюдения за независимыми микросервисами, но и за бизнес-процессами, которые должны успешно выполняться системой в целом.

Роль механизм рабочего процесса

Механизм рабочего процесса – идеальный инструмент для управления состоянием длительно выполняемых процессов. Большинство механизмов рабочих процес-

сов выполняют модели процессов, которые выражены на языке моделирования процессов. Ярким примером является стандартная модель бизнес-процессов и нотация (BPMN), которая позволяет разрабатывать процессы, которые являются как графическими, так и исполняемыми. Механизм рабочего процесса сохраняет все данные для экземпляров процессов даже в течение длительного периода времени. Он также упрощает мониторинг, оповещение и отчетность на уровне процессов.

Целью механизма рабочего процесса в этом решении будет предоставление информации о состоянии сквозных бизнес-процессов, оповещение о любых сбоях, которые могут возникнуть, и даже предоставление возможности исправить или предотвратить проблемы. Для этого бизнес-процессы должны быть представлены с помощью BPMN, которая представляет собой нотацию, используемую механизмом рабочих процессов Camunda, который будет использоваться вместе с программной системой электронной коммерции, описанной ранее.

Преимущества механизма рабочего процесса для микросервисов

- обработка состояния – для отслеживания состояния каждого экземпляра бизнес-процесса; (например, каждый заказ, размещенный на веб-сайте электронной коммерции)
- компенсация за проблемы – компенсирует, если в бизнес-процессе возникает проблема, требующая отмены ранее выполненных шагов;
- обработка тайм-аута – отслеживает течение времени и, если сообщение не получено вовремя, принимает меры или переключает поток процесса на другой путь;
- обработка ошибок – позволяет указать поведение, которое должно происходить при происхождении ошибки;
- совместная работа. Предоставляет графические модели бизнес-процессов, помогающие заинтересованным сторонам, разработчикам и операционным группам взаимодействовать более эффективно [4].

Архитектура предлагаемого системного решения

Разработанная система электронной коммерции, показанная на рисунке 1, представляет собой приложение .NET Core, основанное на управляемой событиями архитектуре микросервисов, работающей в контейнерах Docker.

Любое клиентское приложение может взаимодействовать с микросервисами через шлюз API, который затем взаимодействует с микросервисами через их открытые API-интерфейсы REST. Каждый микросервис имеет свою собственную базу данных, и используется несколько различных технологий баз данных, что позволяет использовать одно из преимуществ микросервисов – неоднородность технологий.

Применяется асинхронный подход через публикацию и подписку на события. Брокер сообщений управляет входящими событиями и предоставляет их микросервисам, подписанным на эти события, гарантированно надежным способом. Когда в домене микросервиса происходит изменение, он не только сохраняет соответствующие данные в своей базе данных, но и создает событие, уведомляющее другие микросервисы об этом изменении. Эта архитектура поддерживает низкое связывание между микросервисами.

Потребуется разработать внешнее приложение для обработки событий, происходящих в экосистеме, и попытаться соотнести каждое из них с конкретным шагом данного экземпляра бизнес-процесса.

Например, в разработанной системе электронной коммерции рабочий процесс оформления заказа клиента может включать микросервис «Корзина» и микросервис «Заказ». При оформлении заказа начнется рабочий процесс. Микросервис «Корзина» будет вносить любые изменения, необходимые для своего домена, и создавать событие «BasketCheckOut», уведомляющее о том, что он может оформить заказ. Затем микросервис «Заказ» будет использовать это событие, выполнять любую соответствующую бизнес-логику и создавать событие, уведомляющее экосистему о том, что элементы для этого заказа были успешно сформулированы.

Все эти шаги составляют рабочий процесс. Каждый шаг рабочего процесса будет связан с событием. Подсистема рабочего процесса будет отвечать за сопоставление этих событий, тем самым показывая состояние рабочего процесса и его успешное завершение.

Наше предлагаемое решение состоит в том, чтобы добавить механизм рабочего процесса Camunda, который является компонентом разработанной системы электронной коммерции, как показано на рисунке 1.

Механизму рабочего процесса Camunda BPMN требуется база данных для управления пользователями, определениями процессов и экземпляров процессов.

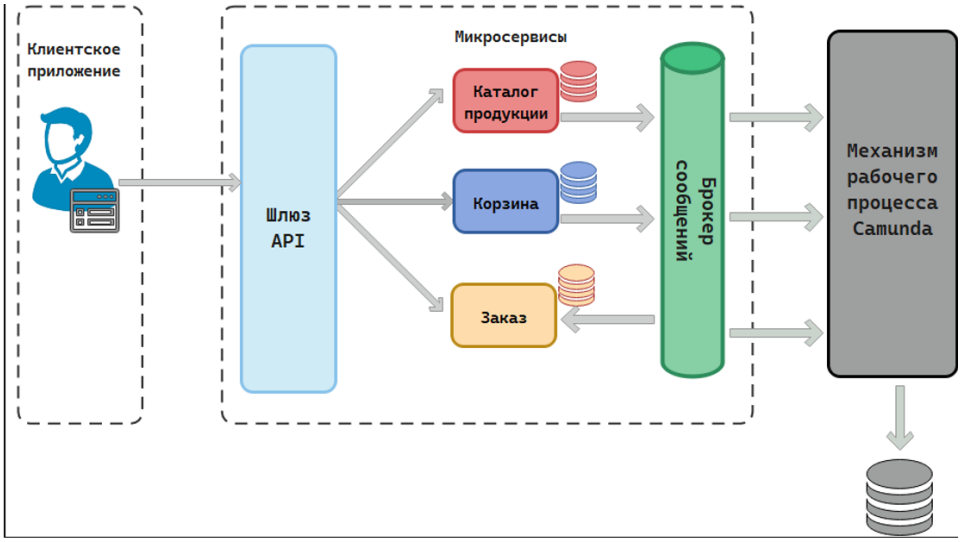

Рис. 1. Архитектура микросервисов, управляемых событиями с механизмом рабочего процесса

Заключение. В работе были обсуждены недостатки микросервисов, управляемых событиями, а также проблемы баланса между оркестровкой и хореографии в архитектуре микросервисов. Было предложено решение добавить механизм рабочего процесса в систему программного обеспечения электронной коммерции в среде ASP .Net Core, чтобы визуализировать и контролировать бизнес-процессы системы.

Библиографический список

1. Rueker B. Practical Process Automation. O'Reilly Media, Inc. – 2021.

2. Артамонов И. Оркестровка и хореография: подходы к описанию композитных бизнес-процессов. – [Электронный ресурс]. – Режим доступа: https://artamonoviv.ru/articles/business_process_orchestration_and_choreography

3. Fowler M. What is event-driven. – [Электронный ресурс]. – Режим доступа: https://martinfowler.com/articles/201701-event-driven.html

4. Camunda. What are Microservices. – [Электронный ресурс]. – Режим доступа: https://camunda.com/glossary/microservices/

PECULIARITIES OF ADDING A WORKFLOW MECHANISM TO APPLICATIONS BASED ON AN EVENT-DRIVEN MICROSERVICE ARCHITECTURE

R. Dandan, *Graduate Student*
Bauman Moscow State Technical University
(Russia, Moscow)

Abstract. *In the course of the study, the shortage in the systems developed with event driven microservices architecture were discussed, the goal is to suggest a solution that helps keeps visibility into processes of the ecosystem. A workflow engine was suggested along with the architecture of adding this engine to the studies e-commerce system.*

Keywords: *microservices, event-driven architecture, orchestration, choreography, bounded context.*