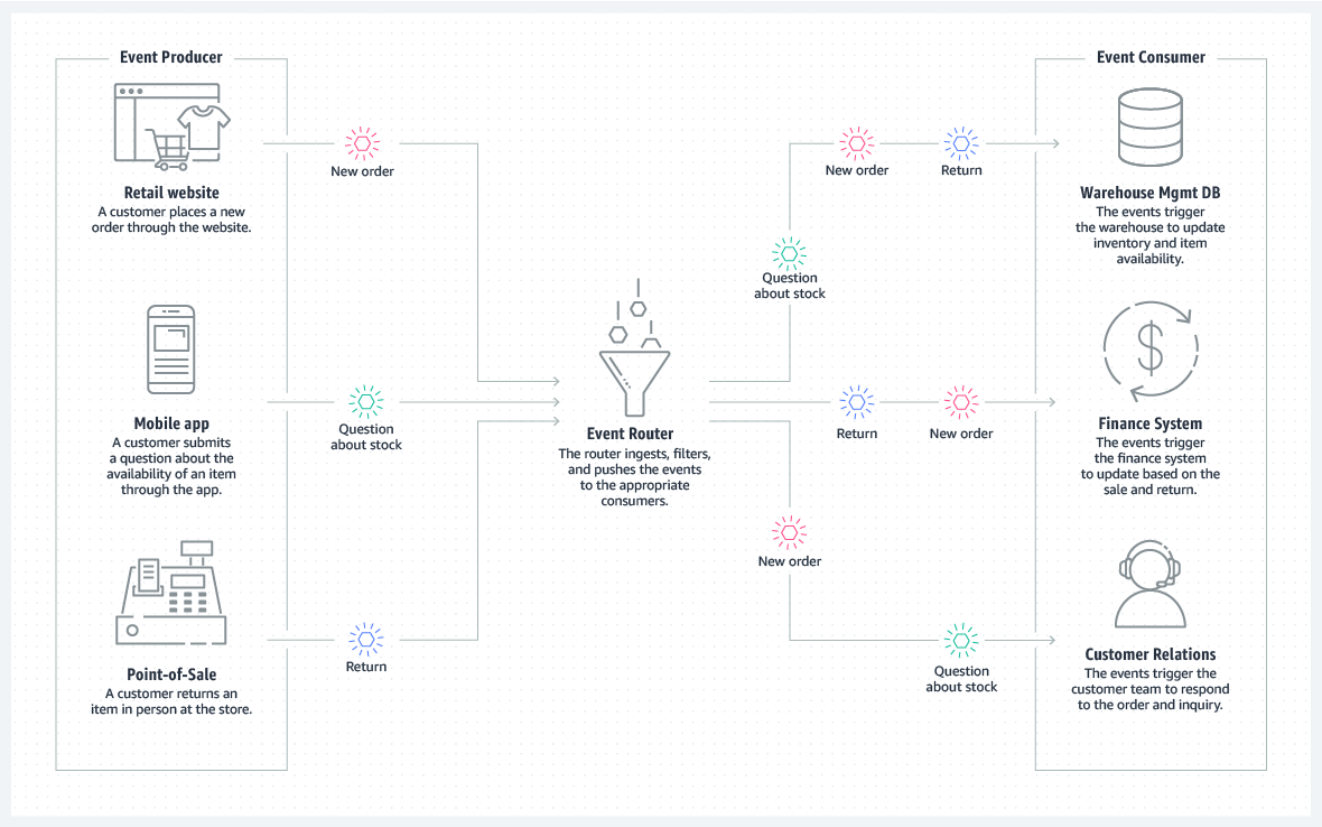
Архитектура, управляемая событиями, — оптимизация поставки программного обеспечения

19.05.2024

Мир разработки программного обеспечения не стоит на месте, и поиск лучших способов создания высококачественных приложений становится все более сложным, чем когда-либо прежде. Ожидания и требования клиентов только растут, поскольку они видят, что возможно сегодня с помощью новейших облачных платформ. Архитектура, управляемая событиями (EDA), — архитектурный паттерн, получивший широкое распространение в последние годы. Воплощая в себе принципы свободного соединения, масштабируемости и обработки данных в реальном времени, событийно-ориентированная архитектура стала переломным моментом в создании программного обеспечения. Мы погрузимся в эту кроличью нору, изучим ее многочисленные преимущества и обсудим ее применение в поставке программного обеспечения. Затем мы узнаем, как этот подход моделирует более гибкую и устойчивую систему, в которой разработчики создают приложения, способные с легкостью справляться с огромными рабочими нагрузками.

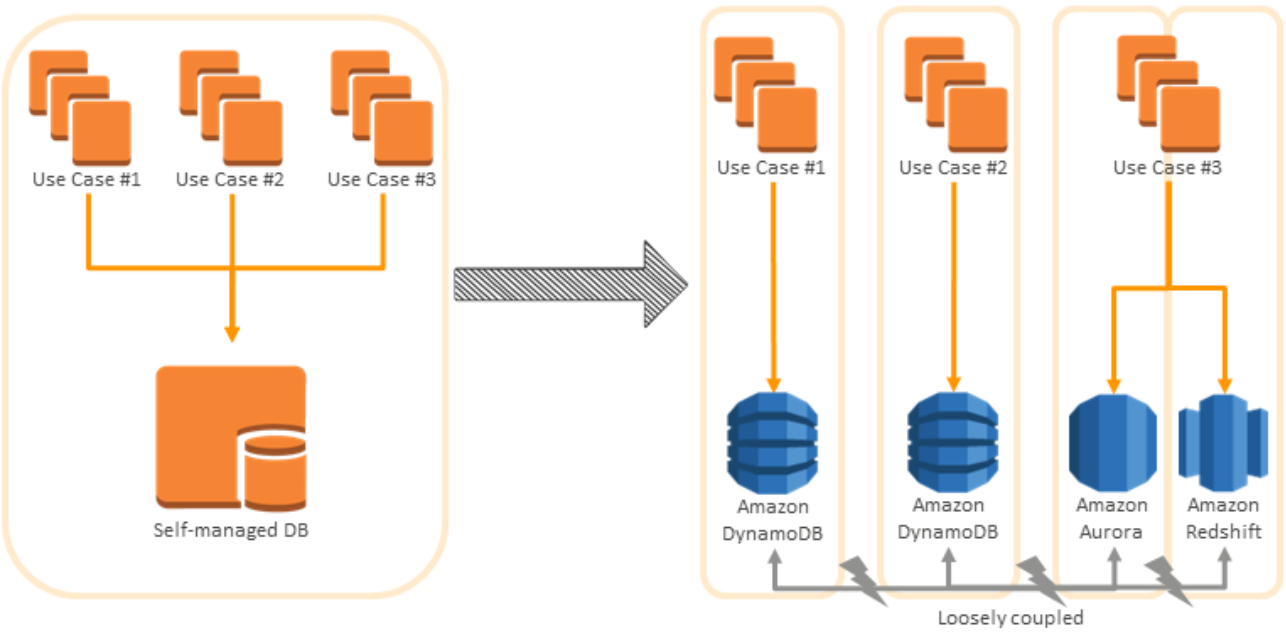
Понимание архитектуры, управляемой событиями

Событийно-ориентированная архитектура — это модель проектирования программного обеспечения, основанная на обработке событий в системе и их потоке. Компоненты системы общаются и взаимодействуют исключительно через эти события.



Вместо прямых синхронных вызовов рабочий процесс событий является асинхронным, что дает значительную свободу и независимость между компонентами. Давайте рассмотрим практические аспекты реализации событийно-управляемой архитектуры и обсудим некоторые паттерны, которые разработчики могут использовать для создания надежных систем.

Сила свободной связи в архитектуре, управляемой событиями

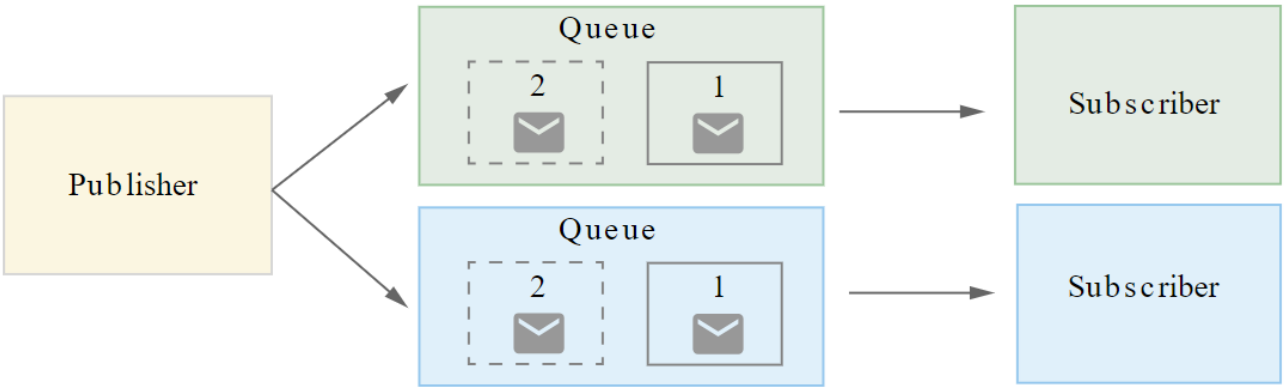


Одним из ключевых преимуществ архитектуры, управляемой событиями, является ее способность разделять компоненты в системе. Вы можете проектировать события как основное средство коммуникации, которое может быть конечным интерфейсом между компонентами. Отдельные компоненты становятся независимыми сущностями, которые могут работать автономно и, чаще всего, асинхронно. Это свойство повышает модульность всей системы, а также обеспечивает более легкое обслуживание и масштабируемость. В такой архитектуре гораздо проще добавлять, модифицировать или заменять компоненты без ущерба для остальной части системы. Поэтому такая архитектура способствует свободному взаимодействию между компонентами. Один компонент не заботится о функциональности другого. Проще говоря, отсутствует внутренняя зависимость. Гибкость при проектировании такой архитектуры гораздо выше, когда нет необходимости учитывать влияние на остальную систему при изменении лишь небольшой ее части. Свободное соединение — это настолько фундаментальный аспект, что имеет смысл немного подробнее изучить его и понять, что именно оно означает в событийно-управляемой архитектуре.

Публикация и подписка на события

Компоненты обычно публикуют события в брокер сообщений или

шину событий (можно понимать это как буфер, заполненный асинхронными сообщениями) без необходимости знать, какие именно компоненты будут потреблять эти события. Этот принцип свободной связи позволяет компонентам работать независимо, поскольку издатели и подписчики строго разделены.



Например, компонент каталога товаров может опубликовать событие “добавлен новый товар”, а другие компоненты, такие как поиск по инвентарю, могут подписаться на это событие и выполнить фактическое действие поиска.

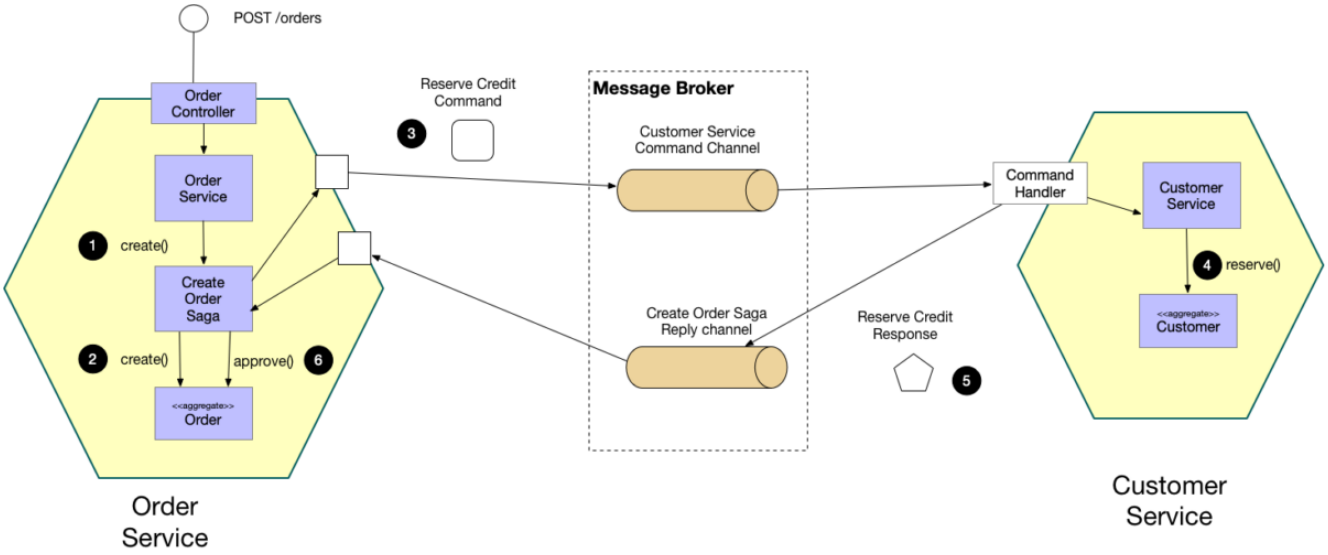
Эволюция схемы событий

По мере развития системы может потребоваться изменить структуру событий. Свободное соединение дает вам свободу для эволюции схем событий с течением времени без влияния на другие компоненты. Ничто из того, что вы уже реализовали, не является незыблемым и может быть легко улучшено с течением времени. Более того, компоненты, потребляющие события, могут быть спроектированы таким образом, чтобы быть устойчивыми к изменениям в структуре событий, что обеспечивает обратную совместимость. Это важно, поскольку вы не хотите ломать существующие потоки со временем, но при этом хотите сохранить возможность внесения изменений в как можно большее количество компонентов. Это позволяет системе развиваться и адаптироваться к изменяющимся требованиям, не вызывая сбоев.

Добавление динамических компонентов

Событийно-ориентированная архитектура дает преимущество простого внедрения новых компонентов, если вам нужно выполнить дополнительную обработку по мере развития платформы. Например, в любой момент в систему может быть добавлен компонент обнаружения мошенничества для анализа событий покупки и выявления потенциально подозрительных действий. Таким образом, свободное соединение позволяет системе легко расширять свою функциональность без жесткого соединения новых компонентов с существующими.

Микросервисы, управляемые событиями

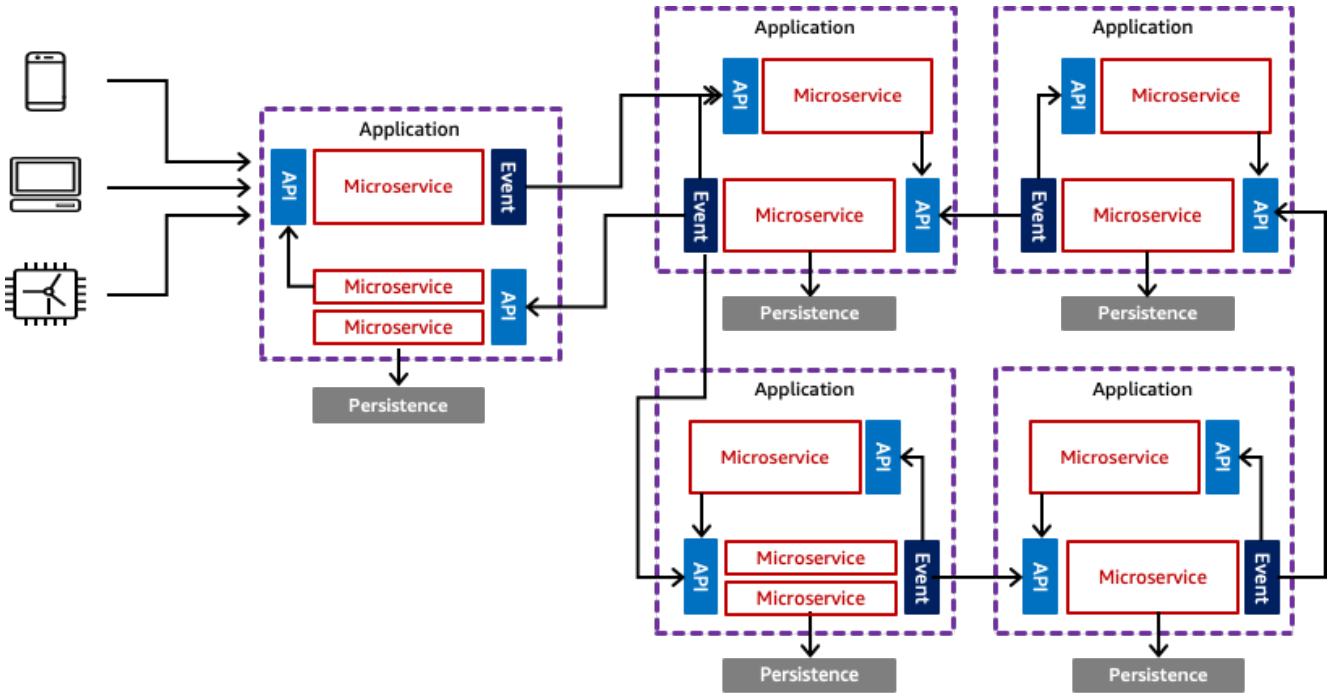


Каждый микросервис, который вы используете в платформе, может быть спроектирован как независимый компонент, взаимодействующий через события. Именно здесь свободное соединение позволяет разрабатывать, развертывать и масштабировать даже микросервисы независимо друг от друга. Например, микросервис управления пользователями может публиковать события, связанные с регистрацией или аутентификацией пользователя, а другие микросервисы могут подписываться на эти события, чтобы выполнять свои функциональные задачи внутри приложения после успешной аутентификации пользователя.

Хореография мероприятий

Компоненты могут взаимодействовать через хореографию событий, что, по сути, означает, что события организуют поведение системы. Они определяют, что делают сквозные процессы внутри платформы. Такой принцип свободной связи позволяет компонентам взаимодействовать опосредованно через события, не зная особенностей реализации друг друга. Это огромное преимущество, поскольку вам даже не нужно привязывать свое приложение к определенному набору языков программирования. Например, в системе обработки заказов такие события, как “заказ оформлен”, “оплата получена” и “груз отправлен”, могут запускать разные компоненты для выполнения своих задач, но они связаны между собой через определенные события, образуя единый рабочий процесс.

Достижение масштабируемости в архитектуре, управляемой событиями



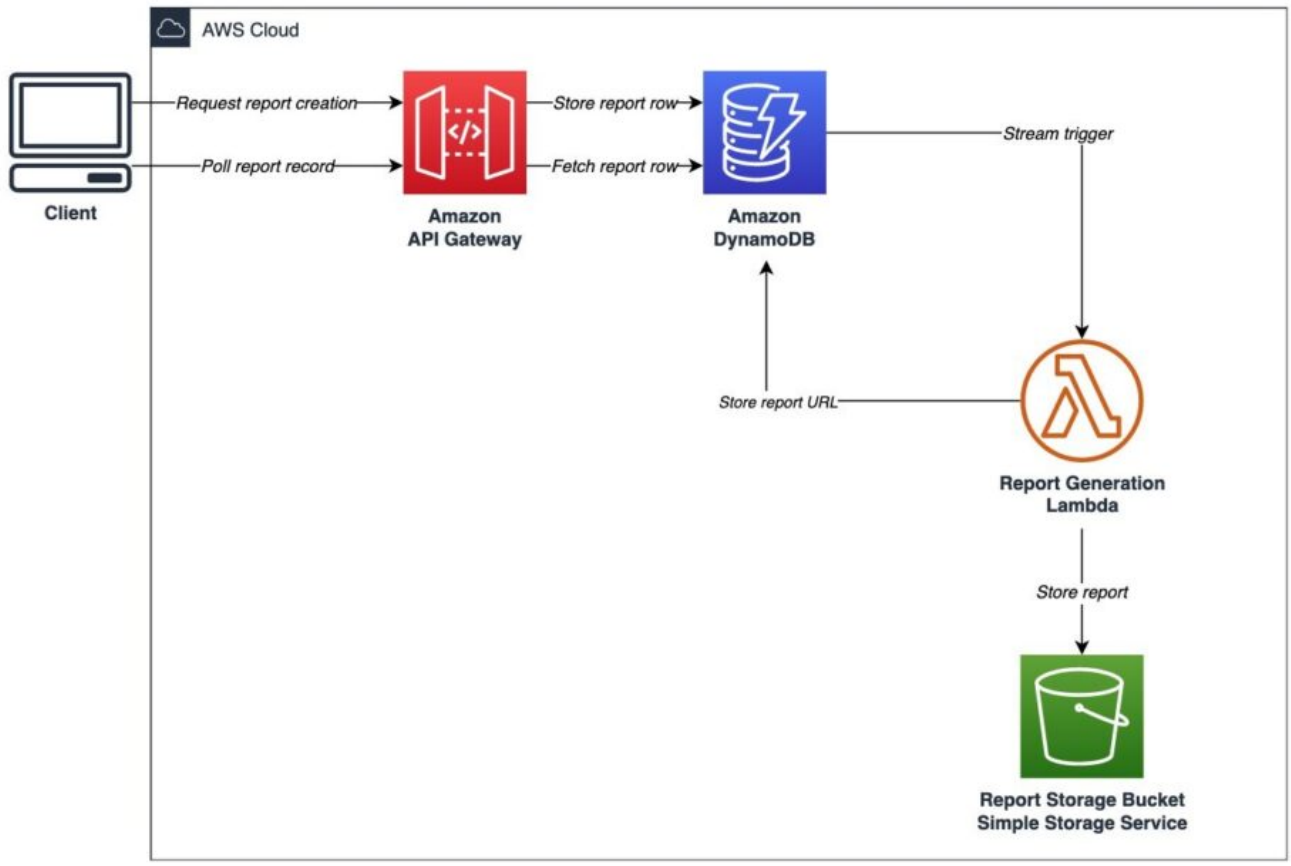
Масштабируемость — это еще одна область, в которой архитектура, управляемая событиями, имеет свои преимущества. Благодаря присущей ей способности обрабатывать большое количество событий одновременно, вы можете увеличивать или

уменьшать масштаб приложений в зависимости от фактической потребности в них. Если рабочая нагрузка распределена между несколькими компонентами, системы, управляемые событиями, могут справиться со скачками трафика и обеспечить оптимальную производительность даже при высоких нагрузках. Это очень важно для приложений, которые испытывают непредсказуемые скачки трафика во время своей работы. Или если им необходимо непредсказуемо обрабатывать большие объемы данных. Масштабируемые системы могут справляться с такими переменными нагрузками без ущерба для производительности. Что же означает эффективное масштабирование компонентов? Чтобы ответить на этот вопрос, необходимо знать несколько ключевых принципов реализации масштабируемости в вашей архитектуре.

Горизонтальное масштабирование

Горизонтальное масштабирование означает добавление большего количества экземпляров одних и тех же компонентов для распределения той же рабочей нагрузки. Таким образом, вы увеличиваете параллельность существующих процессов, но при этом они не потребляют одинаковые необработанные системные ресурсы. Вместо этого каждый из экземпляров работает со своими системными ресурсами. В архитектуре, управляемой событиями, этого можно достичь путем развертывания нескольких экземпляров компонентов, потребляющих события, таких как процессоры событий или обработчики событий. Распределяя рабочую нагрузку между этими экземплярами, система может обрабатывать больший объем одновременно. Горизонтальное масштабирование особенно эффективно в сочетании с методами балансировки нагрузки для равномерного распределения событий между экземплярами. Это не позволит одному из экземпляров стать узким местом для остальных.

Асинхронная обработка



Событийно-ориентированная архитектура по умолчанию поддерживает асинхронную обработку, что очень важно для масштабируемости. Асинхронная обработка позволяет компонентам обрабатывать события независимо друг от друга, не блокируя и не ожидая завершения каждого события. Если вы используете только асинхронные процессы, вы не можете попасть в тупик или заблокировать ресурсы в целом. Вместо этого вы постоянно используете все ресурсы и обрабатываете большее количество событий одновременно. Однако ваша архитектура должна опираться на асинхронную связь, что не всегда легко спроектировать.

Разделение событий

С помощью разделения событий вы можете разделить события на логические разделы на основе определенных критериев, таких как тип события, источник или пункт назначения. Каждый раздел может обрабатываться независимо отдельными компонентами или экземплярами, поскольку они отличаются друг от друга с точки зрения данных. Таким образом, поддерживается параллельная обработка и улучшается масштабируемость. Разделение событий

может быть достигнуто с помощью таких методов, как разделение событий или разделение тем событий, что в основном зависит от конкретных требований вашей системы.

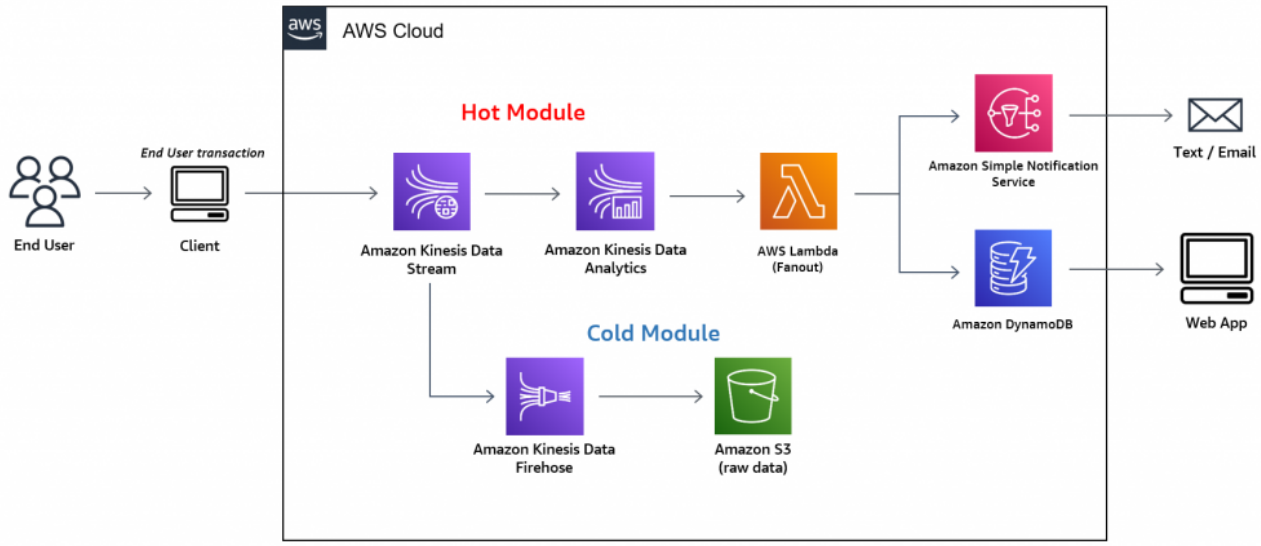
Распределенные брокеры сообщений

Брокеры сообщений играют центральную роль в архитектуре, управляемой событиями, поскольку они облегчают обмен сообщениями между компонентами. Для поддержания масштабируемости очень важно использовать распределенные брокеры сообщений, которые могут работать с высокой пропускной способностью. Распределенные брокеры сообщений распределяют события по нескольким узлам, что обеспечивает горизонтальное масштабирование и гарантирует, что система сможет обрабатывать поступающие события.

Мониторинг и автомасштабирование

Наконец, внедрение механизмов мониторинга и автоматического масштабирования необходимо для того, чтобы ваша система могла динамически регулировать свои ресурсы в зависимости от текущей рабочей нагрузки. Без этого вы не сможете увидеть, когда такое масштабирование необходимо. Отслеживая такие ключевые показатели, как длина очереди событий, время обработки и загрузка ресурсов, вы можете позволить системе автоматически увеличивать или уменьшать масштаб, добавляя или удаляя экземпляры по мере необходимости. Такое динамическое масштабирование обеспечивает оптимальную производительность и быстроту реакции даже при пиковых нагрузках.

Обработка данных в реальном времени в архитектуре, управляемой событиями



Обработка данных в реальном времени — критическое требование для многих современных приложений, и событийно-ориентированная архитектура отлично подходит для этой области. Вы можете перехватывать и обрабатывать события по мере их возникновения, реагируя на них в режиме реального времени. В результате вы получаете бесперебойную актуальную информацию. Такая возможность важна для приложений, требующих немедленного реагирования или предоставления актуальной информации пользователям. Будь то мониторинг различных устройств Интернета вещей (IoT) или обработка финансовых транзакций, событийно-ориентированная архитектура должна обеспечивать реакцию на события с минимальной задержкой. Существует существенная разница между обработкой данных в реальном и близком к реальному времени. Последнее, как правило, достигается гораздо легче, поскольку вы все же можете позволить себе некоторую задержку (обычно в минутах). Однако обеспечение архитектуры реального времени — очень сложная задача. Чтобы лучше понять, что именно это значит, давайте рассмотрим конкретный пример, иллюстрирующий, чего стоит ожидать от настоящей событийно-управляемой архитектуры обработки данных в реальном времени. Рассмотрим платформу совместного использования поездок, работающую в городе. Платформа должна обрабатывать большое количество событий в режиме реального времени, чтобы быть полезной как для водителей, так и для пассажиров. Если она будет работать

только в режиме близком к реальному времени, то желаемого эффекта уже не будет. Вот что необходимо учесть в такой архитектуре.

Генерация событий

События генерируются, как только пассажиры запрашивают поездку или когда водители принимают запрос на поездку. Затем эти события публикуются в брокере сообщений. Например, создается событие с такими деталями, как место сбора и пункт назначения.

Сопоставление в режиме реального времени

Когда публикуется событие запроса на поездку, компонент подбора подписывается на это событие и обрабатывает его в режиме реального времени (именно там доступный водитель будет сопряжен с этим запросом). Компонент реализации использует алгоритмы для сопоставления запроса с наиболее подходящим доступным водителем на основе таких факторов, как близость или тип автомобиля. Опять же, требование к обработке в реальном времени очень сильное, так как водитель перемещается в автомобиле довольно быстро. Через пять минут расстояние может значительно отличаться от того, которое было на момент запроса пассажира.

Динамическое ценообразование

По мере обработки запросов на поездку и событий о наличии водителя компонент ценообразования может подписаться на эти события как на следующие. Этот компонент будет рассчитывать фактическую стоимость поездки. Например, в часы пик или при высоком спросе компонент ценообразования может увеличить стоимость проезда или уменьшить ее при низком спросе. Высокий и низкий спрос может меняться очень быстро, поэтому даже здесь задержка нежелательна.

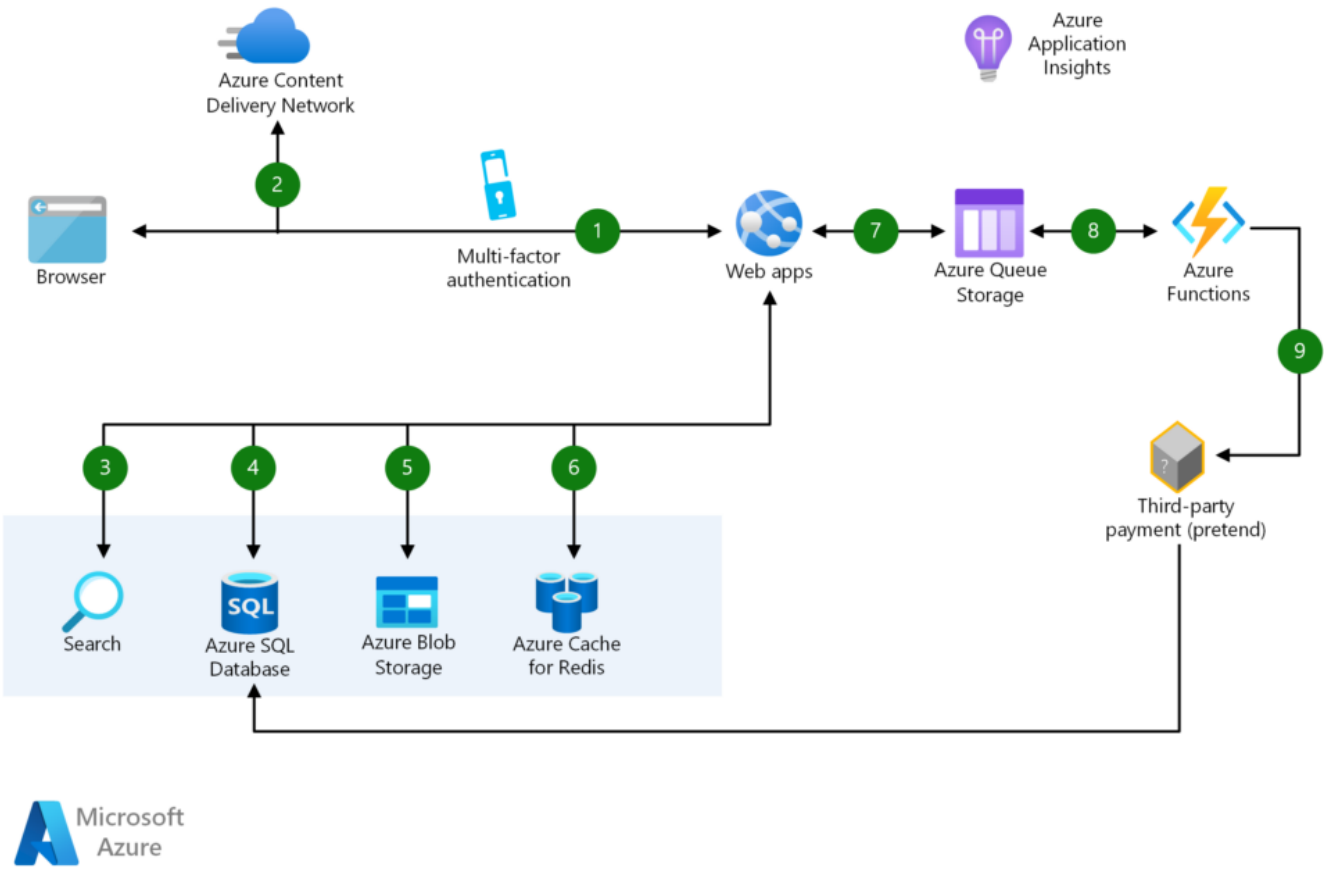
Отслеживание в режиме реального времени

Как только поездка принята, местоположение водителя в реальном времени становится очень важным. Обновления местоположения водителя фиксируются как события и обрабатываются в режиме реального времени. Пассажиры могут видеть и отслеживать эту информацию с помощью компонента отслеживания.

Аналитика в режиме реального времени

Платформа выполняет аналитику различных аспектов в режиме реального времени. Например, события, связанные с оценками водителей или отзывами клиентов, обрабатываются в режиме реального времени для получения информации и запуска таких действий, как уведомления об эффективности работы водителей или программы вознаграждения.

Практическая реализация архитектуры, управляемой событиями



Давайте рассмотрим еще один практический пример реализации событийно-управляемой архитектуры в контексте платформы электронной коммерции. В этом сценарии подумайте об онлайн-площадке, которая соединяет покупателей и продавцов. Платформа должна обрабатывать большое количество событий. Например, объявления о новых товарах, покупки, обновление запасов или взаимодействие с пользователями. Во-первых, платформа может разделять свои компоненты. Каталог товаров, управление запасами, обработка заказов или уведомления пользователей являются отдельными компонентами. Каждый компонент может взаимодействовать через события. Например, когда продавец добавляет новое объявление о товаре, генерируется событие и публикуется в брокере сообщений. Когда платформа испытывает скачки трафика, архитектура масштабируется горизонтально за счет добавления новых экземпляров компонентов, обрабатывающих события. Например, когда большое количество пользователей одновременно совершают покупки, компонент обработки заказов масштабируется, чтобы справиться с нагрузкой, потребляя события от брокера сообщений. Кроме того, платформа предоставляет пользователям актуальную информацию. Например, когда покупатель размещает заказ, компонент обработки заказов немедленно обновляет инвентарь и запускает уведомления для продавца и покупателя. Это должно происходить немедленно. В противном случае может возникнуть множество несоответствий. Далее, если вы решите внедрить механизм рекомендаций, вы можете добавить новый компонент, который будет потреблять события взаимодействия с пользователем и генерировать персонализированные рекомендации, не затрагивая другие (существующие) компоненты. В случае сбоев компонентов событийно-ориентированная архитектура должна иметь механизм восстановления. Предположим, компонент инвентаризации вышел из строя. В этом случае события, связанные с обновлением запасов, можно буферизировать или перенаправить на альтернативный компонент. Для конечного пользователя это все равно будет выглядеть как бесперебойная обработка заказов.

Заключение

Событийно-ориентированная архитектура предлагает практичный и эффективный подход к созданию программного обеспечения. Ее нелегко внедрить, и команды не слишком хорошо знакомы с паттернами, которых требует такая архитектура. Однако его преимущества, включая свободное соединение, масштабируемость и обработку данных в реальном времени, делают его привлекательным выбором для создания надежных приложений для будущих программных платформ.