Очереди сообщений

Сервисы обмена сообщениями между серверами делятся на два типа:

- Очереди (ФИФО-метод) - «первый пришел и первый ушел» (First in First out). Принцип FIFO означает: сообщение, которое первым попало в очередь, первым же отправляется на обработку.

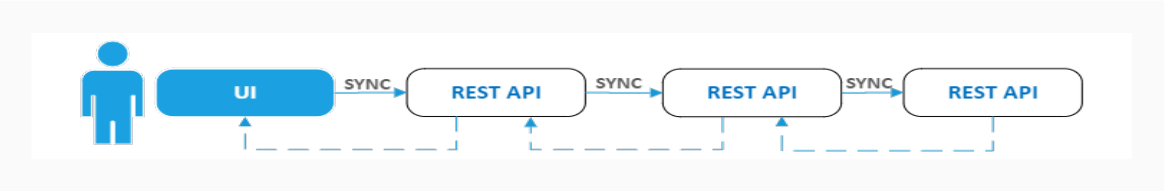
- Стеки (LIFO) - «пришел последним, а ушел первым» (Last in First out). В отличие от системы FIFO, стек можно представить в виде стопки тарелок: вы кладете тарелки друг на друга и сначала берете верхние тарелки.

Очереди сообщений (Message Queue) — это форма асинхронной коммуникации между сервисами. Поэтому, прежде чем говорить о них, покажем на упрощенном, немного искусственном примере разницу между синхронным и асинхронным взаимодействием.

Предположим, вы разрабатываете сайт книжного магазина и у вас есть сервис, к которому обращается пользователь, например отправка рецензии на прочитанную книгу. При нажатии кнопки «Отправить» вызывается некоторый API, который, в свою очередь, может обратиться к другим API.

При синхронном взаимодействии все запросы в этой цепочке вызовов выполняются строго друг за другом, а при выполнении последнего запроса ответы последовательно передаются в обратном направлении. В итоге пользователь вынужден пару секунд ждать сообщения о публикации своего отзыва, хотя его не интересуют особенности серверной обработки и он вполне обоснованно хочет увидеть сообщение сразу после нажатия кнопки. Конечно, время ожидания будет во многом определяться мощностью оборудования, но при пиковых нагрузках оно может стать серьезной проблемой.

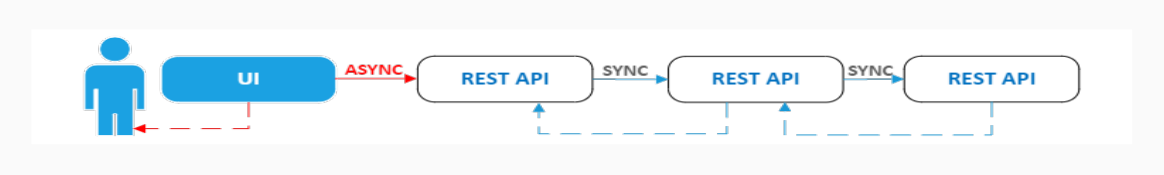
Еще один недостаток такой схемы — обработка сбоев. Если на одном из шагов возникнет исключение, оно каскадно возвратится назад, и пользователь получит уведомление об ошибке с просьбой повторно отправить рецензию. Вряд ли кого-то обрадует получение подобного сообщения после длительного ожидания.



Синхронное взаимодействие на основе REST API

Описанную схему можно изменить, добавив асинхронные вызовы. Достаточно вызвать в асинхронном режиме первый REST API и параллельно вернуть пользователю сообщение о том, что его рецензия принята и будет размещена, например, в течение суток. В итоге сайт не блокируется, а вызовы всех последующих API происходят независимо от пользователя.

Но у такой схемы также есть существенный недостаток: в случае сбоя в одном из API информация, введенная пользователем, будет потеряна. Если в первом примере в случае ошибок достаточно повторно отправить рецензию, то здесь ее необходимо заполнить заново.



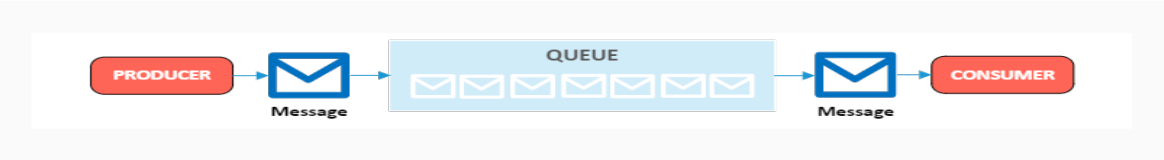
Вариант асинхронного взаимодействия на основе REST API

Для устранения недостатков обеих схем как раз и предназначены очереди сообщений.

Принципы работы очередей сообщений

Очереди предоставляют буфер для временного хранения сообщений и конечные точки, которые позволяют подключаться к очереди для отправки и получения сообщений в асинхронном режиме.

В сообщениях могут содержаться запросы, ответы, ошибки и иные данные, передаваемые между программными компонентами. Компонент, называемый производителем (Producer), добавляет сообщение в очередь, где оно будет храниться, пока другой компонент, называемый потребителем (Consumer), не извлечет сообщение и не выполнит с ним необходимую операцию.



Очередь сообщений

Очереди поддерживают получение сообщений как методом Push, так и методом Pull:

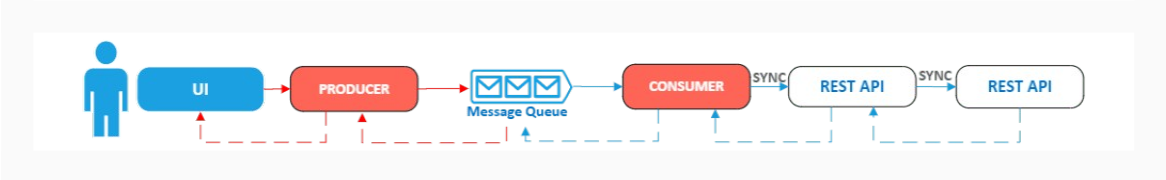
- метод Pull -подразумевает периодический опрос очереди получателем по поводу наличия новых сообщений;
- метод Push - отправку уведомления получателю в момент прихода сообщения. Второй метод реализует модель «Издатель/Подписчик» (Publisher/Subscriber).

Так как очереди могут использоваться несколькими производителями и потребителями одновременно, обычно их реализуют с помощью дополнительной системы, называемой брокером. **Брокер сообщений** (Message Broker) занимается сбором и маршрутизацией сообщений на основе предопределенной логики. Сообщения могут передаваться с некоторым ключом — по этому ключу брокер понимает, в какую из очередей (одну или несколько) должно попасть сообщение.

Вернемся к примеру с отправкой рецензии. Пусть та часть сервиса, к которому обращается пользователь, выступит в качестве производителя и будет направлять запросы на создание рецензий в очередь. Сразу после добавления сообщения в очередь пользователю можно направлять уведомление об успехе операции. Вся последующая логика обработки будет

выполняться независимо от него на стороне потребителя, подписанного на очередь.

Завершив обработку, потребитель отправит подтверждение в очередь, после чего исходное сообщение будет удалено. Но если во время обработки произойдет сбой и подтверждение не будет получено вовремя, сообщение может быть повторно извлечено потребителем из очереди.



Вариант асинхронного взаимодействия на основе очереди сообщений

Таким образом, использование очередей сообщений решает сразу две задачи: сокращает время ожидания пользователя за счет асинхронной обработки и предотвращает потерю информации при сбоях. Но не следует рассматривать очереди как универсальное средство для любого вида приложений: как и у любого инструмента, у них есть свои преимущества и недостатки, о которых мы поговорим ниже.

Польза и преимущества очередей сообщений в микросервисной архитектуре

Используя очереди сообщений в качестве основного средства взаимодействия микросервисов (Microservices Communication), можно добиться следующих преимуществ:

1. Отделение логически независимых компонентов друг от друга (Decoupling)

   Отличительная черта микросервисов — их автономность. И очереди во многом помогают уменьшить зависимости между ними. Каждое сообщение, передаваемое в очереди, — это всего лишь массив байтов с некоторыми метаданными. Метаданные нужны для направления в

конкретную очередь, а информация, содержащаяся в основной части (теле) сообщения, может быть практически любой. Брокер не анализирует данные, он выступает лишь в качестве маршрутизатора. Это позволяет настроить взаимодействие между компонентами, работающими даже на разных языках и платформах.

2. Улучшение масштабируемости

   Очереди сообщений упрощают независимое масштабирование микросервисов. Наблюдая за состоянием очередей, можно масштабировать те сервисы, на которые приходится большая часть нагрузки. Кроме этого, очереди легко позволяют не только увеличивать число экземпляров существующих сервисов, но и добавлять новые с минимальным временем простоя. Все, что для этого требуется, — добавить нового потребителя, прослушивающего события в очереди.

   Однако сами очереди также необходимо масштабировать, и это может создать дополнительные сложности.

3. Балансировка нагрузки

   Если один из сервисов не справляется с нагрузкой, требуется возможность запускать больше его экземпляров быстро и без дополнительных настроек. Обычно для этих целей используют балансировщик нагрузки, интегрированный с сервером обнаружения служб и предназначенный для распределения трафика. При использовании очередей сообщений сам брокер по умолчанию является балансировщиком нагрузки. Если несколько потребителей слушают очередь одновременно, сообщения будут распределяться между ними в соответствии с настроенной стратегией.

4. Повышение надежности

Выход из строя одного из компонентов не сказывается на работе всей системы: при восстановлении он обработает сообщение, находящееся в очереди. Ваш веб-сайт по-прежнему может работать, даже если задерживается часть обработки заказа, например, из-за проблем с сервером БД или системой электронной почты.

Правда, при этом очередь сама приобретает статус SPoF (Single Point Of Failure), поэтому необходимо заранее предусмотреть действия на случай ее аварийного отключения.

5. Безопасность

   Большинство брокеров выполняют аутентификацию приложений, которые пытаются получить доступ к очереди, и позволяют использовать шифрование сообщений как при их передаче по сети, так и при хранении в самой очереди. Таким образом, очередь снимает с ваших сервисов бремя организации авторизации запросов.

Варианты использования очередей сообщений

Очереди сообщений полезны в тех случаях, где возможна асинхронная обработка. Рассмотрим наиболее частые сценарии использования очередей сообщений (Message Queue use Cases):

1. Фоновая обработка долгосрочных задач на веб-сайтах

   Сюда можно отнести задачи, которые не связаны напрямую с основным действием пользователя сайта и могут быть выполнены в фоновом режиме без необходимости ожидания с его стороны. Это обработка изображений, преобразование видео в различные форматы, создание отзывов, индексирование в поисковых системах после изменения данных, отправка электронной почты, формирование файлов и так далее.

2. Буферизация при пакетной обработке данных

Очереди можно использовать в качестве буфера для некоторой массовой обработки, например пакетной вставки данных в БД или HDFS. Очевидно, что гораздо эффективнее добавлять сто записей за раз, чем по одной сто раз, так как сокращаются накладные расходы на инициализацию и завершение каждой операции. Но для стандартной архитектуры может стать проблемой генерация данных клиентской службой быстрее, чем их может обработать получатель. Очередь же предоставляет временное хранилище для пакетов с данными, где они будут храниться до завершения обработки принимающей стороной.

3. Отложенные задачи

   Многие системы очередей позволяют производителю указать, что доставка сообщений должна быть отложена. Это может быть полезно при реализации льготных периодов. Например, вы разрешаете покупателю отказаться от размещения заказа в течение определенного времени и ставите отложенное задание в очередь. Если покупатель отменит операцию в указанный срок, сообщение можно удалить из очереди.

4. Сглаживание пиковых нагрузок

   Помещая данные в очередь, вы можете быть уверены, что данные будут сохранены и в конечном итоге обработаны, даже если это займет немного больше времени, чем обычно, из-за большого скачка трафика. Увеличить скорость обработки в таких случаях также возможно — за счет масштабирования нужных обработчиков.

5. Гарантированная доставка при нестабильной инфраструктуре

   Нестабильная сеть в сочетании с очередью сообщений создает надежный системный ландшафт: каждое сообщение будет отправлено, как только это будет технически возможно.

6. Упорядочение транзакций

Многие брокеры поддерживают очереди FIFO, полезные в системах, где важно сохранить порядок транзакций. Если 1000 человек размещают заказ на вашем веб-сайте одновременно, это может создать некоторые проблемы с параллелизмом и не будет гарантировать, что первый заказ будет выполнен первым. С помощью очереди можно определить порядок их обработки.

7. Сбор аналитической информации

   Очереди часто применяют для сбора некоторой статистики, например использования определенной системы и ее функций. Как правило, моментальная обработка такой информации не требуется. Когда сообщения поступают в веб-службу, они помещаются в очередь, а затем при помощи дополнительных серверов приложений обрабатываются и отправляются в базу данных.

8. Разбиение трудоемких задач на множество маленьких частей

   Если у вас есть некоторая задача для группы серверов, то вам необходимо выполнить ее на каждом сервере. Например, при редактировании шаблона мониторинга потребуется обновить мониторы на каждом сервере, использующем этот шаблон. Вы можете поставить сообщение в очередь для каждого сервера и выполнять их одновременно в виде небольших операций.

9. Прочие сценарии, требующие гарантированной доставки информации и высокого уровня отказоустойчивости

   Это обработка финансовых транзакций, бронирование авиабилетов, обновление записей о пациентах в сфере здравоохранения и так далее.

Сложности использования и недостатки очередей сообщений: как с ними справляться

Несмотря на многочисленные преимущества очередей сообщений, самостоятельное их внедрение может оказаться довольно сложной задачей по нескольким причинам:

- По сути, это еще одна система, которую необходимо купить/установить, правильно сконфигурировать и поддерживать. Также потребуются дополнительные мощности.
- Если брокер когда-либо выйдет из строя, это может остановить работу многих систем, взаимодействующих с ним. Как минимум необходимо позаботиться о резервном копировании данных.
- С ростом числа очередей усложняется и отладка. При синхронной обработке сразу очевидно, какой запрос вызвал сбой, например, благодаря иерархии вызовов в IDE. В очередях потребуется позаботиться о системе трассировки, чтобы быстро связать несколько этапов обработки одного запроса для обнаружения причины ошибки.
- При использовании очередей вы неизбежно столкнетесь с выбором стратегии доставки сообщений. В идеале сообщения должны обрабатываться каждым потребителем однократно. Но на практике это сложно реализовать из-за несовершенства сетей и прочей инфраструктуры. Большинство брокеров поддерживают две стратегии: доставка хотя бы раз (At-least-once) или максимум раз (At-most-once). Первая может привести к дубликатам, вторая — к потере сообщений. Обе требуют тщательного мониторинга. Некоторые брокеры также гарантируют строго однократную доставку (Exactly-once) с использованием порядковых номеров пакетов данных, но даже в этом случае требуется дополнительная проверка на стороне получателя.

В каких случаях очереди неэффективны

Конечно, очереди не являются универсальным средством для любых приложений. Рассмотрим варианты, когда очереди не будут самым эффективным решением:

- У вашего приложения простая архитектура и функции, и вы не ожидаете его роста. Важно понимать, что очереди сообщений — это дополнительная сложность. Эту систему также необходимо настраивать, поддерживать, осуществлять мониторинг ее работы и так далее. Да, можно использовать Managed-решение, но вряд ли это будет оправдано для небольших приложений. Добавление очередей должно упрощать архитектуру, а не усложнять ее.
- Вы используете монолитное программное обеспечение, в котором развязка (Decoupling) невозможна или не приоритетна. Если вы не планируете разбивать монолит на микросервисы, но вам требуется асинхронность — для ее реализации обычно достаточно стандартной многопоточной модели. Очереди могут оказаться избыточным решением до тех пор, пока не возникнет явная необходимость в разделении приложения на автономные компоненты, способные независимо выполнять задачи.

Выбирая инструмент для будущего приложения, обязательно взвесьте все за и против. Не стоит использовать очереди сообщений для задач, которые могут быть решены другим, более простым в настройке и обслуживании способом. Но в тех случаях, когда запланирован переход на микросервисы и бизнес-логика допускает возможность асинхронной обработки, очереди сообщений могут стать лучшим выбором для повышения производительности и надежности вашего продукта.

Брокер сообщений

Брокер сообщений представляет собой тип построения архитектуры, при котором элементы системы «общаются» друг с другом с помощью посредника. Благодаря его работе происходит снятие нагрузки с веб-сервисов, так как им не приходится заниматься пересылкой сообщений: всю сопутствующую этому процессу работу он берёт на себя.

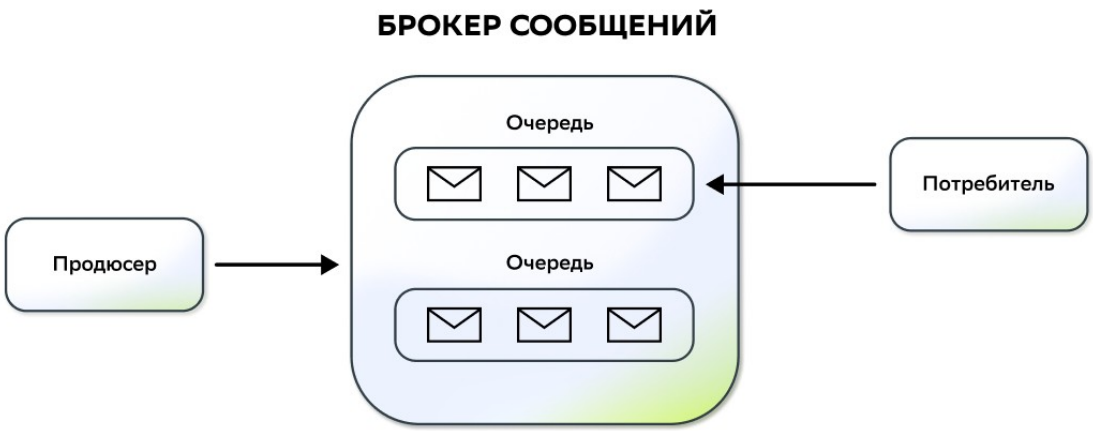
Можно сказать, что в работе любого брокера сообщений используются две основные сущности: producer (издатель сообщений) и consumer (потребитель/подписчик).

Одна сущность занимается созданием сообщений и отправкой их другой сущности-потребителю. В процессе отправки есть ещё серединная точка, которая представляет собой папку файловой системы, где хранятся сообщения, полученные от продюсера.

Здесь возможны несколько вариантов:

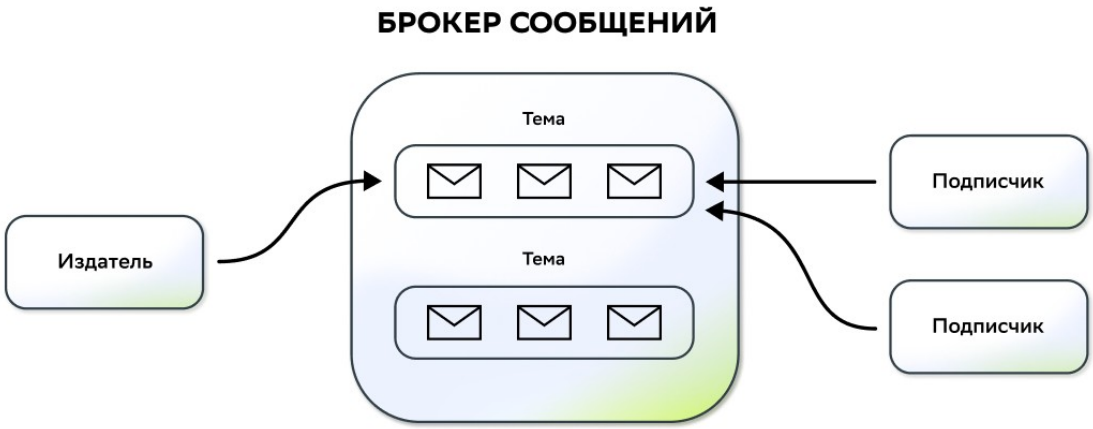
- Сообщение отправляется напрямую от отправителя к получателю.

В этом случае каждое сообщение используется только однократно;



●Схема публикации/подписки.

В рамках этой схемы обмена сообщениями отправитель не знает своих получателей и просто публикует сообщения в определённую тему. Потребители, которые подписаны на эту тему, получают сообщение. Далее на базе этой системы может быть построена работа с распределением задач между подписчиками. То есть выстроена логика работы, когда в одну и ту же тему публикуются сообщения для разных потребителей. Каждый «видит» уникальный маркер своего сообщения и забирает его для исполнения.



Для чего нужны брокеры сообщений?

●Для организации связи между отдельными службами, даже если какая-то из них не работает в данный момент. То есть продюсер может отправлять сообщения, несмотря на то, проявляет ли активность потребитель в настоящее время.

●За счёт асинхронной обработки задач можно увеличить производительность системы в целом.

●Для обеспечения надёжности доставки сообщений: как правило, брокеры обеспечивают механизмы многократной отправки сообщений в тот же момент или через определённое время. Кроме того, обеспечивается

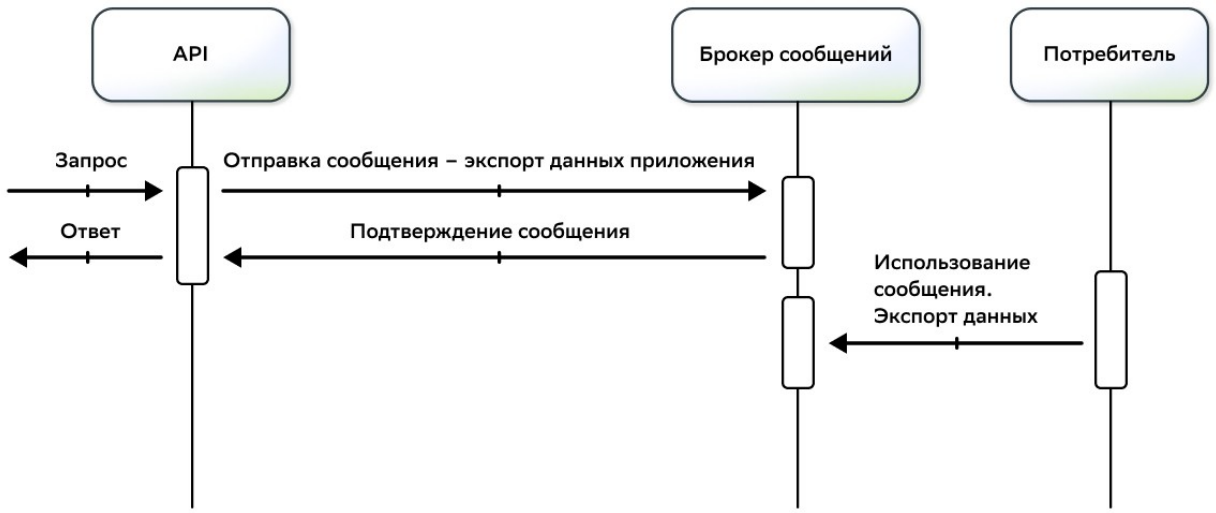
соответствующая маршрутизация сообщений, которые не были доставлены.

Недостатки брокеров сообщений:

- Усложнение системы в целом как таковой, так как в ней появляется ещё один элемент. Кроме того, возникает зависимость от надёжности распределённой сети, а также потенциальная возможность возникновения проблем из-за потребности в непротиворечивости данных, так как некоторые элементы системы могут обладать неактуальными данными.
- Из-за асинхронной работы всей системы, а также её распределённого характера могут возникать ошибки, выяснение сути которых может стать непростой задачей.
- Освоение подобных систем является не самым простым вопросом и может занять существенное время.

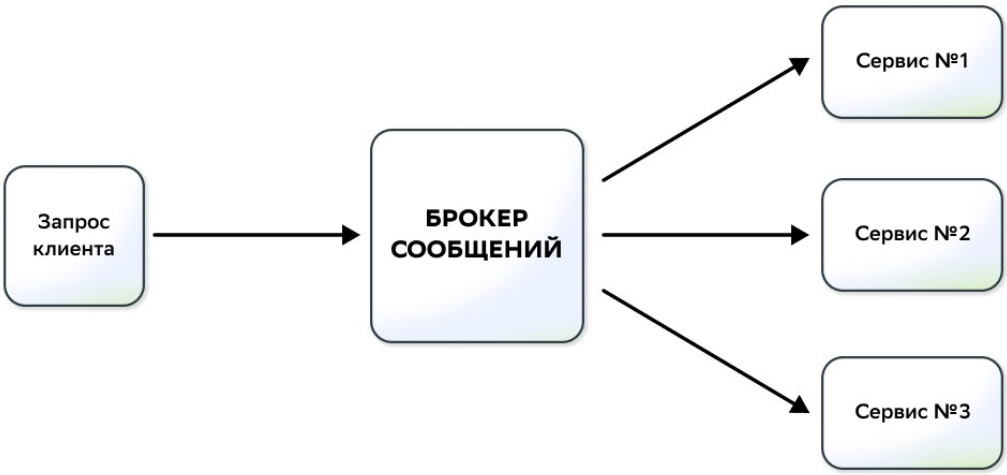
Когда брокеры сообщений могут быть полезны:

- Если в рамках вашей системы есть действия, которые требуют для своего выполнения много времени и потребляют много ресурсов, при этом они не требуют немедленного результата.



- Микросервисы: если ваша система достаточно сложна и состоит из отдельных сервисов, то для их координации можно использовать брокер

сообщений, который в этом случае будет выступать в роли как бы центрального роутера. Каждый сервис подписывается только на свой тип сообщений, выстраивается определённая логика их обработки.



●Мобильные приложения: здесь возможен вариант с задействованием push-уведомлений, когда множество смартфонов с установленным приложением подписаны на определённую тему. Если в ней публикуется какая-либо новость, то подписанный смартфон выводит уведомление.

●Транзакционные системы: если какое-то действие системы состоит из множества отдельных этапов, каждый из которых выполняется отдельным элементом системы, то в этом случае брокер сообщений может выступить в роли своеобразной «доски уведомлений». Каждый сервис отписывается после того, как его этап общей задачи был выполнен. После этого в работу вступает следующий сервис, обрабатывающий, соответственно, следующий этап общей задачи.

Какие есть брокеры сообщений?

Следует сразу оговориться, что брокеров сообщений существует великое множество, и приведённый по ссылке список прямо говорит об этом.

Каждый из них обладает определёнными «фишками» и хорошо решает возложенные на него задачи. Например, если одни предназначены для создания инфраструктуры связи между распределёнными частями приложения, другие предназначены для достаточно специфических задач, например «интернета вещей», и функционируют на основе легковесного протокола MQTT. Подобные брокеры служат для сбора статистики, температуры и других показателей с распределённых датчиков, установленных на определённых машинах, элементах конструкций, географически разных точках территории.

Малое время задержки для прохода сообщения по сети, а также возможность двунаправленной связи в реальном времени (так как MQTT является надстройкой над дуплексным протоколом websockets) позволяют реализовывать достаточно интересные применения. Среди них может быть даже управление робототехническими устройствами в реальном времени. При таком управлении задержка не превышает десятков миллисекунд, что вполне допустимо для низкоскоростных устройств. Например, для управления роботами,

движущимися в промышленных цехах и использующимися для перевозки деталей от станков или сырья к производственным станкам.

Также информационные системы на базе протокола MQTT используются в автомобильной сфере и в ряде других промышленных областей (HiveMQ).