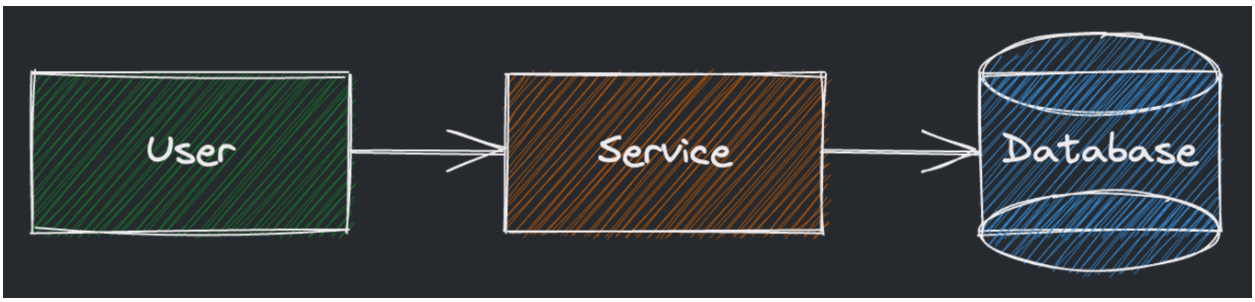
Урок 6 > Модульный подход к дизайну

> Оглавление

> Модульный подход к дизайну

Ранее мы рассматривали высокоуровневые компоненты, из которых будет состоять большинство систем.

При этом всю логику внутри приложения мы часто прятали за одним логическим «квадратом». Давайте попробуем углубиться в детали того, что реально может происходить под капотом.



Общее представление компонент

Интернет-магазин

Допустим, мы проектируем интернет-магазин и у нас есть часть логики, отвечающая за совершение покупки.

Добавление в корзину и доставка

Подсервис покупки логически должен далее обратиться в подсервис, связанный отправкой на доставку.

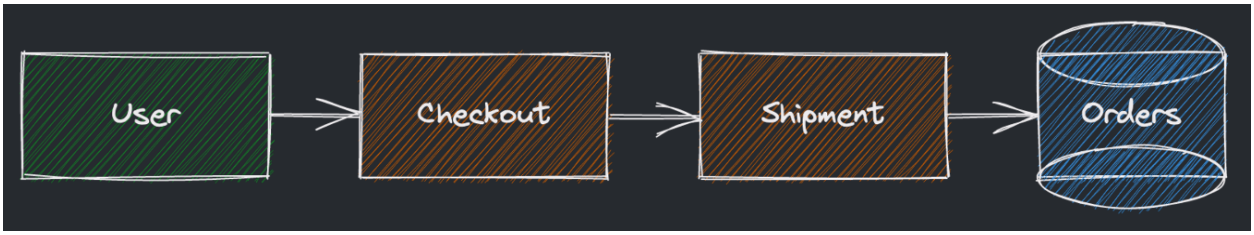
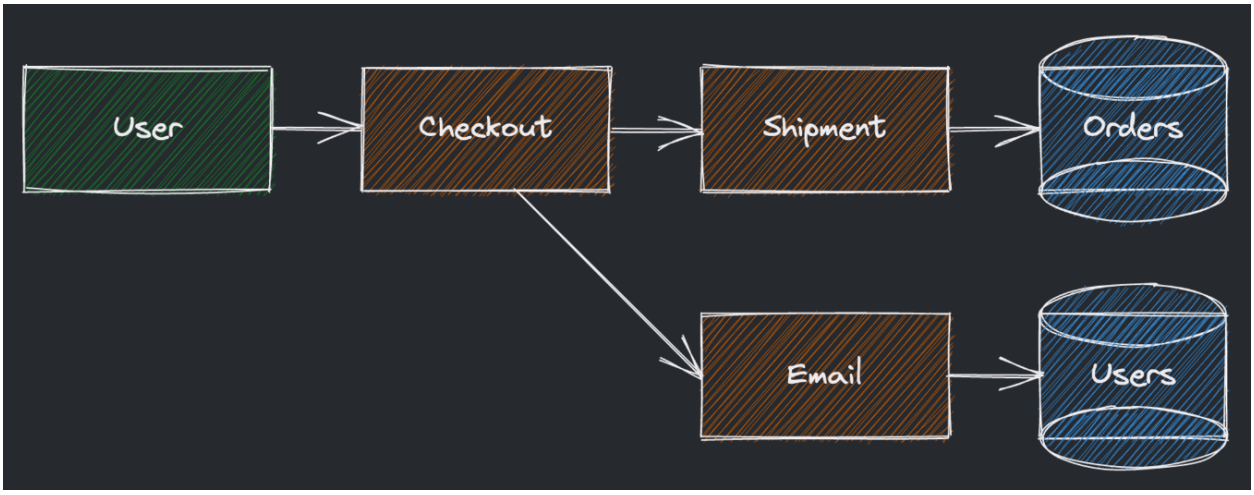


Схема подсервисов для интернет-магазина

Уведомления

Помимо функциональности по добавления товара в корзину покупателем и оформления доставки, могут быть как другие функции, так и другие категории пользователей.

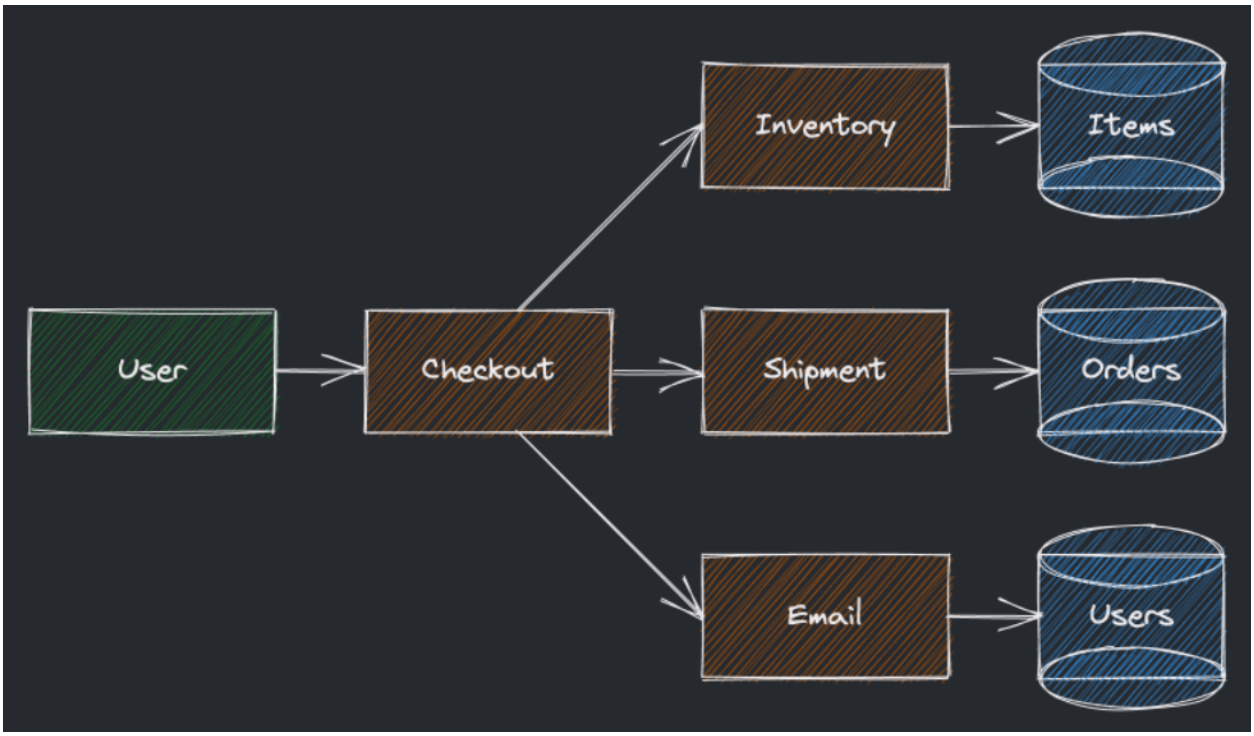
После совершение покупки логично отправить какое-то уведомление пользователю.



Сервис отправки уведомлений взаимодействует с БД пользователей, чтобы узнать почту и предпочтения по уведомлениям

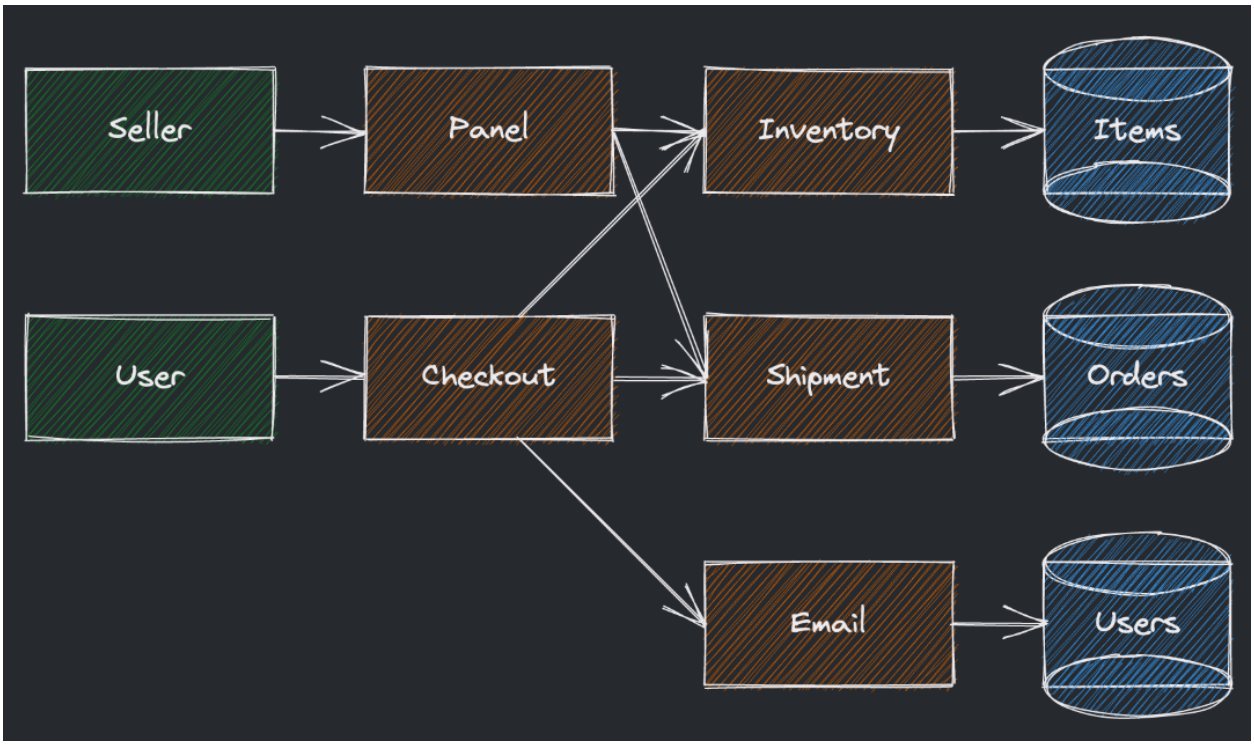
Изменения остатков

После совершения покупки нам также необходимо обновить информацию о наличии тех или иных товаров.



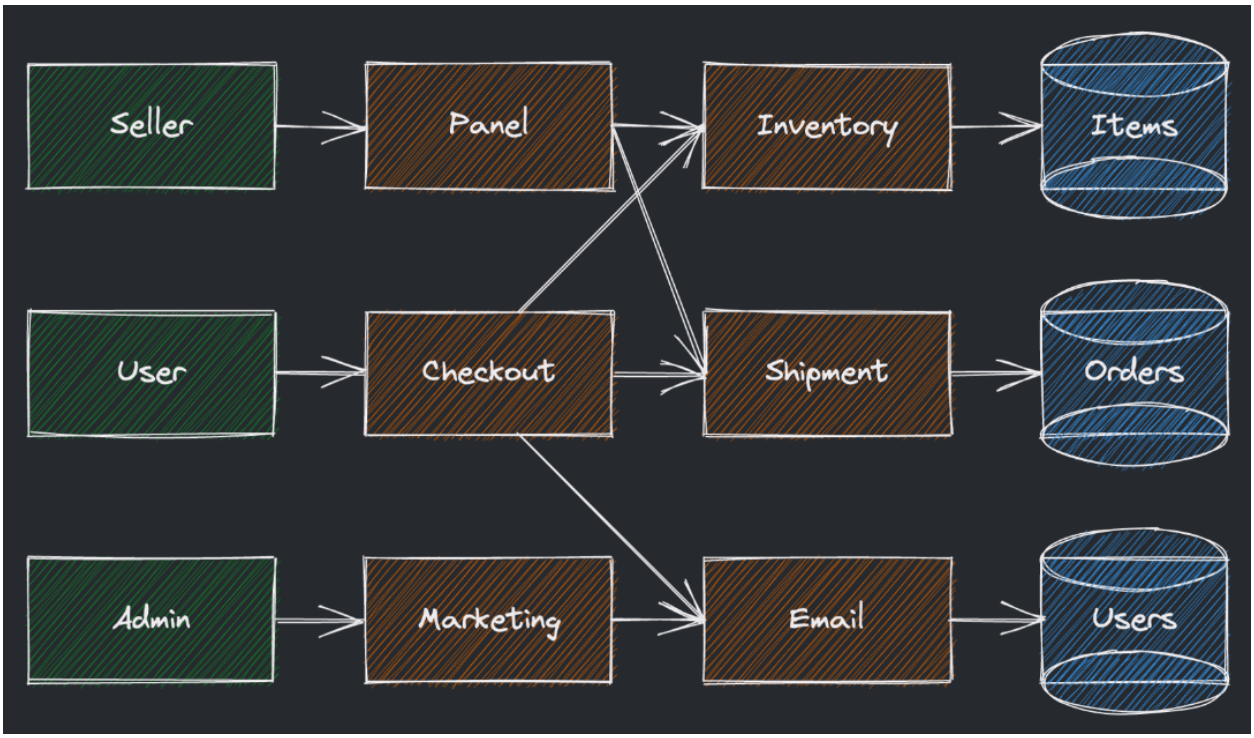
Продавцы на площадке

Помимо простых покупателей обновлять инвентарь и работать с заказами могут и продавцы. Им предоставляется панель администрирования, где можно создавать новые товары, обновлять количества, делать выгрузки в csv файлы по различным статистикам.



Администрация

На площадке могут присутствовать другие акторы: администраторы, модераторы и так далее — люди, следящие за совершением покупок. Например, рассылкой писем могут заниматься и люди из отдела маркетинга через соответствующие инструменты. Здесь будет использоваться тот же подсервис отправки уведомлений почтой, что и для рассылок по подтверждению заказов.



Таким образом, от абстрактных квадратов мы перешли к:

- реальным акторам, которые взаимодействуют между собой
- категориям акторов
- сценариям взаимодействия
- функционалу сервисов

Преимущества и сложности подхода

Минусы

- Со временем зависимости могут расти
- Неравномерная нагрузка (пользователи чаще совершают покупки, чем продавец влияет на заказ)

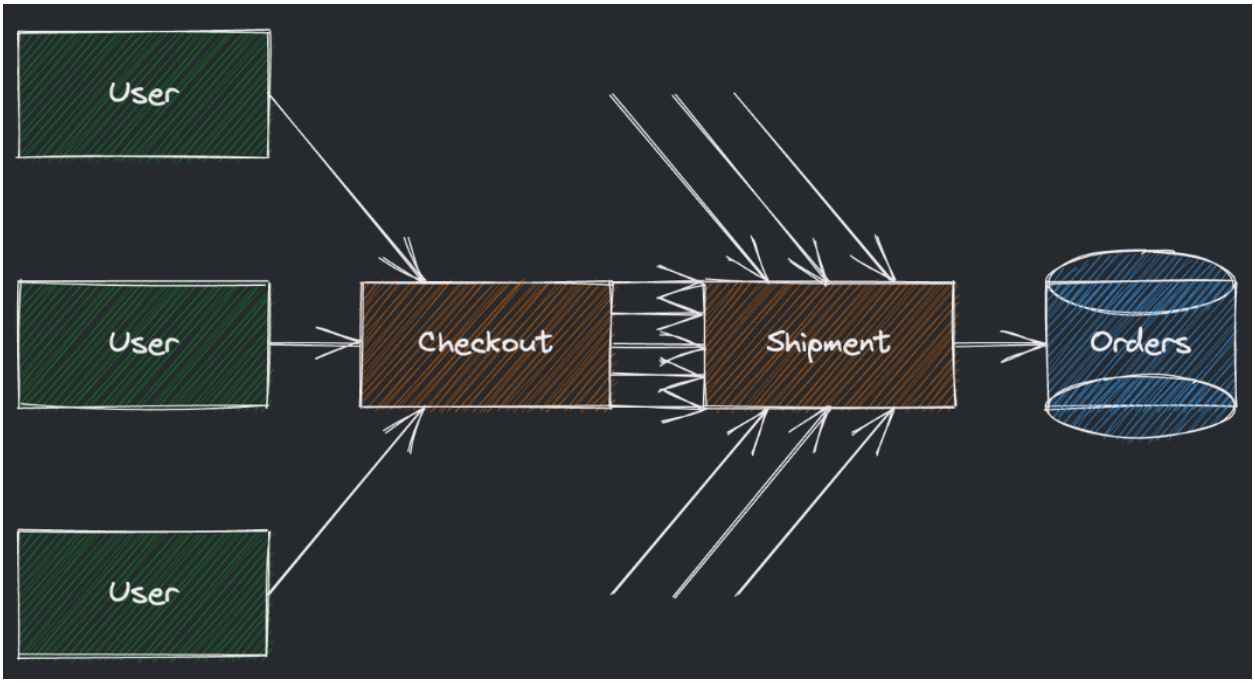
Плюсы

- Мы сразу отображаем картину того, какие смысловые части приложения должны быть реализованы

- Легко переносить функциональные требования в дизайн, на каждое требование можно выделить модуль
- У разных модулей разная нагрузка, масштабировать их можно независимо (но будут нужны балансировщики)

Сложности

- Уже на рассмотренном примере становится очевидна сложность образующихся связей между частями системы
- Продолжение работы зависимого сервиса в принципе зависит от работоспособности используемого
- Зависимый сервис начнет тормозить в случае наплыва обращений в используемом сервисе (вероятно, нужно создавать несколько инстансов, использовать локдаун и тому подобное — поговорим чуть позднее)



Далее поговорим о том, как можно разделить разные сервисы. Особенно хорошо этот подход работает, когда не требуется срочный ответ от вызываемого сервиса.

Например, при совершении заказа, можно асинхронно передать сообщение о том, что покупка совершена (есть факт подтверждения), а сервис, отвечающий за

доставку, в порядке поступления сообщений, передаст данные в транспортную компанию. Это можно реализовать через **очередь сообщений**.

> Очереди сообщений

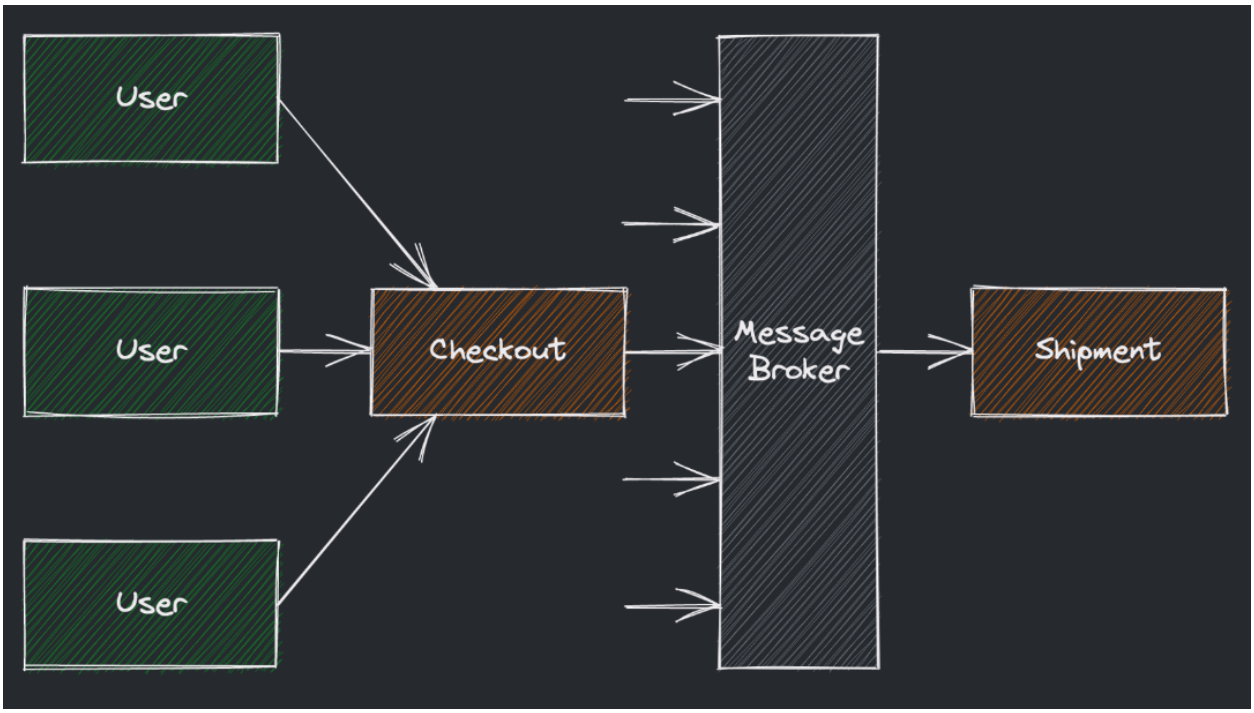
Очередь сообщений — сущность, в которую будут приходить сообщения от разных сервисов, которые называются продьюсерами (создают новые сообщения). Далее сообщения будут использоваться в других сервисах, которые называются консьюмерами (получают сообщения и осуществляют какую-то работу над ними).

Интернет-магазин

Вернемся к примеру: после совершения заказа и оплаты придет сообщение о том, что товар куплен. Оно попадает в очередь сообщений. Далее сервис, отвечающий за доставку:

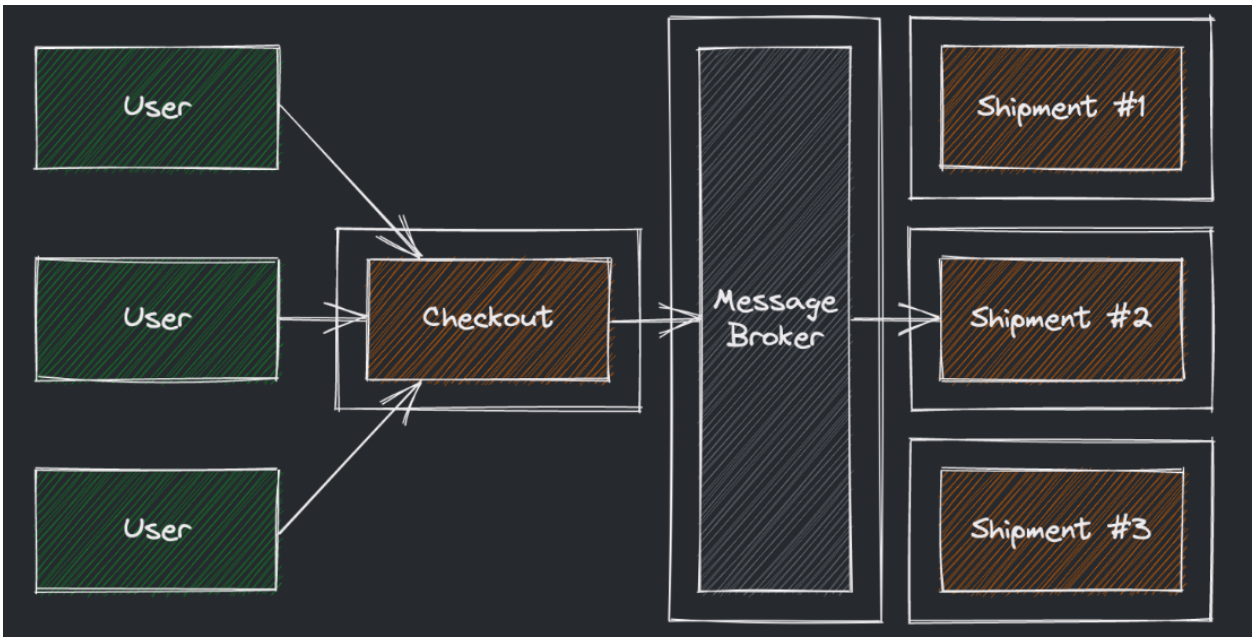
- либо сразу получает такие сообщения
- либо приходит и забирает накопившиеся сообщения

Соответственно, очередь сообщений используется внутри какого-то брокера сообщений, отвечающего за взаимодействие с очередью.



Преимущества очередей сообщений

- Сервисы разъединяются на логические составляющие и не используют друг друга напрямую
- Разъединенные сервисы можно независимо и просто масштабировать за счет количества воркеров
- Очереди лежат на отдельном инстансе и основные свободны от лишней работы, косвенно ускорены

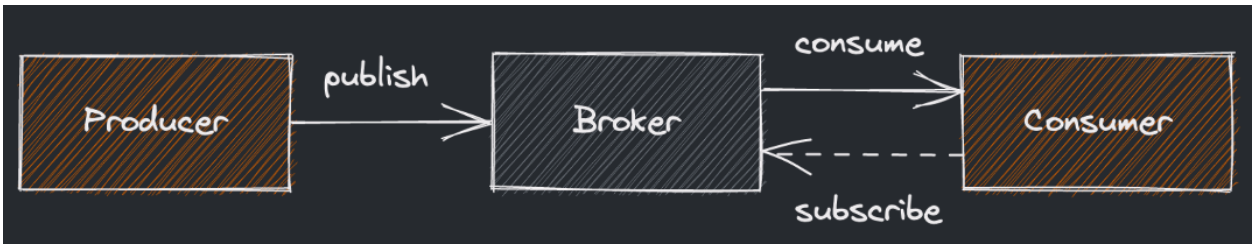


Произвольное количество консьюмеров сервисов доставки (воркеров), обрабатывающих сообщения, взаимодействуют с компаниями по доставке

Например, сервис заказов генерирует 100 RPS, а один консьюмер доставки только 10 RPS, тогда можно сделать 10 воркеров. Если появляется нужда обрабатывать больше сообщений (например, отмены заказов продавцами), можно добавить еще необходимое количество воркеров.

Устройство очереди сообщений

- Одни клиентские приложения (продьюсеры) создают и отправляют сообщения брокеру (т.е. кладут в очередь)
- Другие же приложения (консьюмеры) подключаются к очереди и подписываются на сообщения для обработки
- Приложения могут быть как продьюсерами, так и консьюмерами, так и тем и другим одновременно
  Сообщения лежат в очереди до тех пор, пока какой-либо консьюмер не получит их



Есть брокеры сообщений, которые хранят сообщения до того, как их получат, а есть те, которые хранят и после.

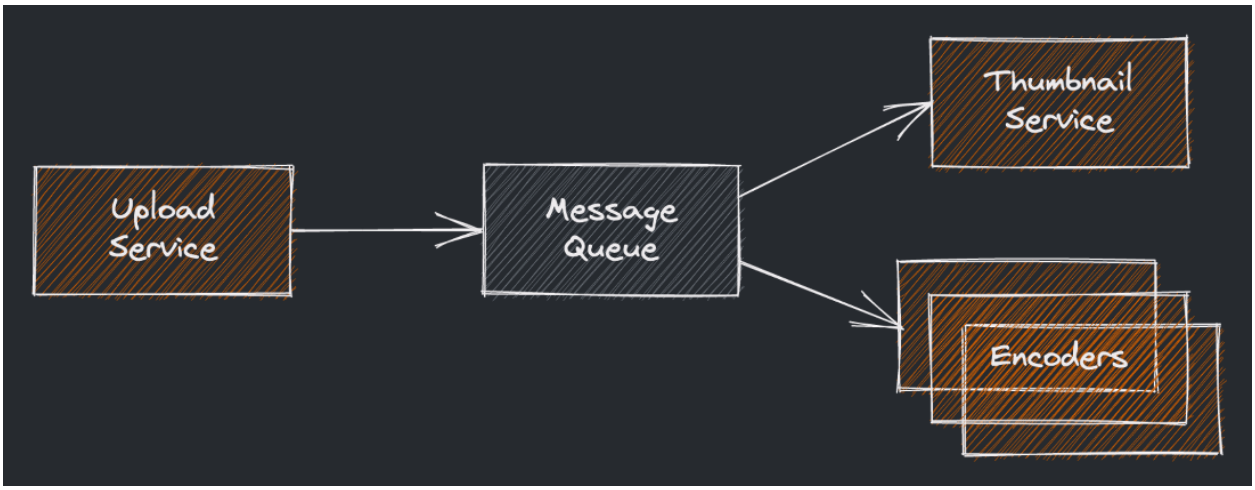
Сценарии взаимодействия:

- push based подход — брокер получает сообщение и тут же посылает его в какой-то консьюмер, подписанный на данный вид сообщений
- pull based подход — сообщения, посылаемые брокеру, складываются, хранятся некоторое время, а потом приходит консьюмер и забирает нужное количество сообщений. Например, вы не успели отмасштабировать сервис, пришла запредельная нагрузка, сообщения будут скапливаться в очереди и обрабатываться чуть позднее. В такие часы пик можно постфактум добавить больше консьюмеров, а потом их убрать.

Сценарии применения

- Один сервис использует функционал другого сервиса, возможно нескольких других
- Нет необходимости получать финальный ответ, но при этом надо продолжать работу

Пример: пользователь загружает видео, а затем в фоне оно перекодируется и создаются превью.

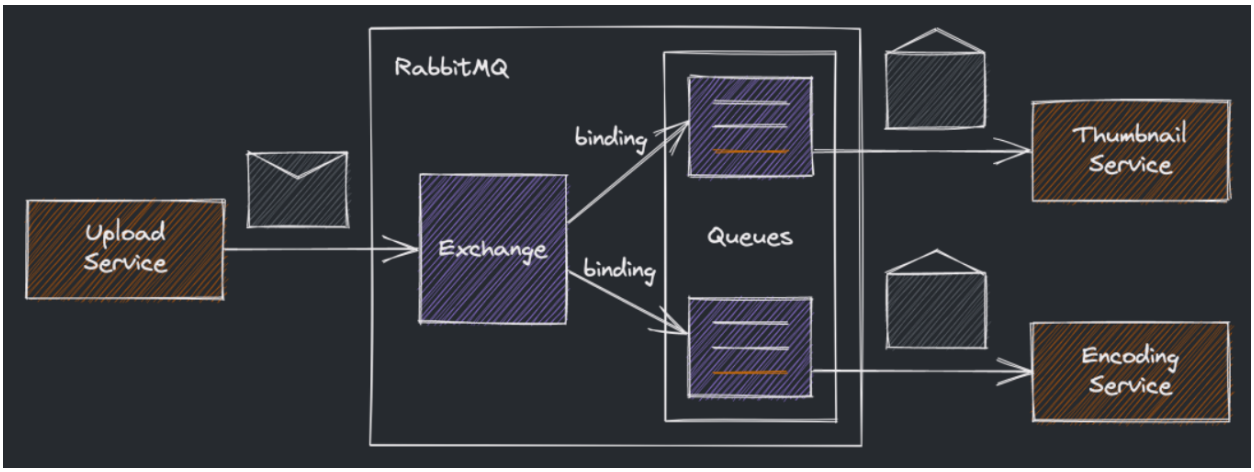


> RabbitMQ

В реальности сервисы скорее всего общаются с каким-то брокером сообщений, который внутри отвечает за взаимодействие с очередью.

Устройство

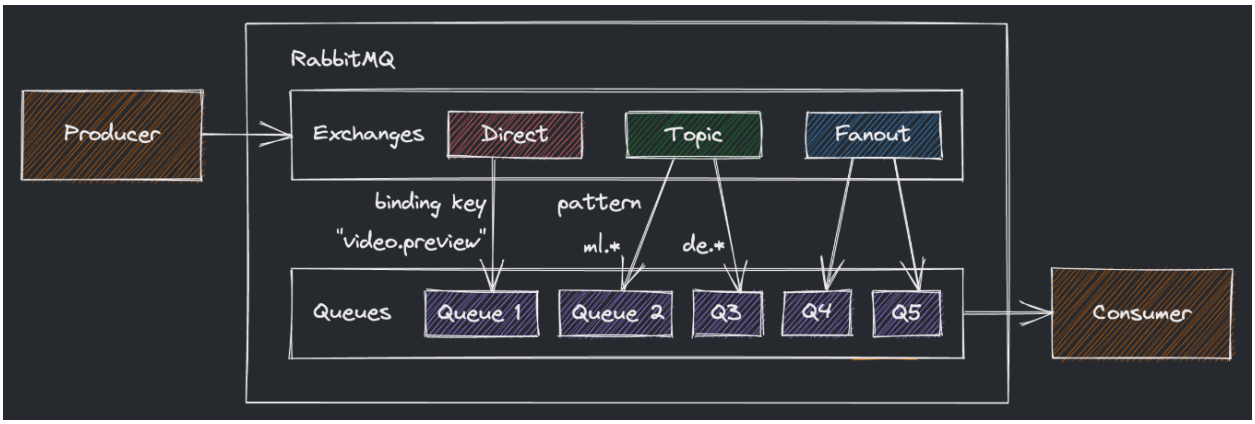
1. Продьюсеры отправляют сообщения не в очередь напрямую, а в обменник (exchange)
2. Обменник в свою очередь отвечает за маршрутизацию сообщений по разным очередям
3. Сообщения маршрутизируются на основе привязок (binding, подобие доменного адреса) и ключей маршрутизации (routing keys)

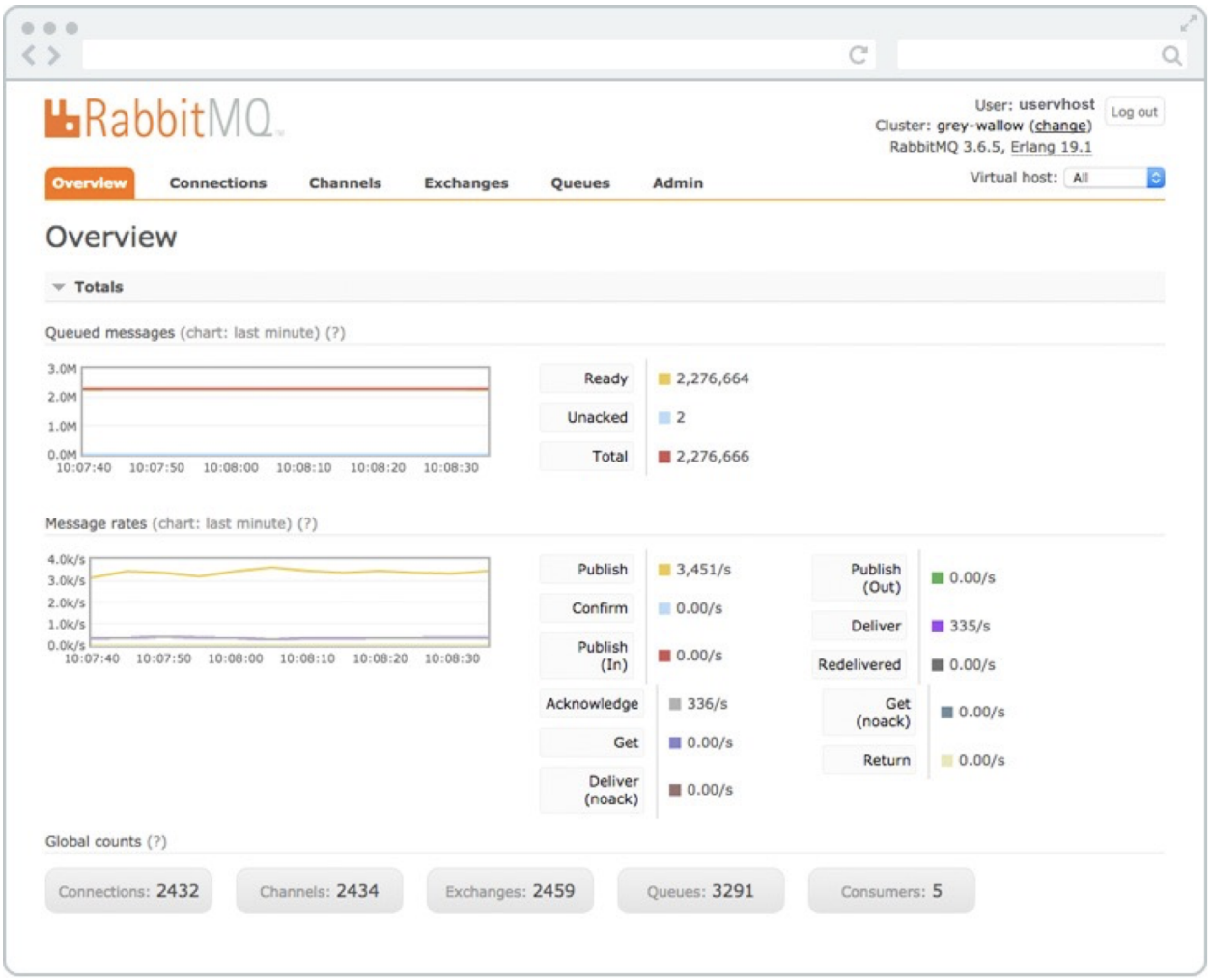


Сервис загрузки видео

Типы используемых обменников

- Direct: сообщение отправляется в очередь, ключ которой совпадает с ключом привязки
- Topic: сообщения отправляются в очереди, ключ которых удовлетворяет шаблону
- Fanout: все сообщения отправляются во все привязанные очереди
- Header: маршрутизация осуществляется на основе заголовков в сообщениях.



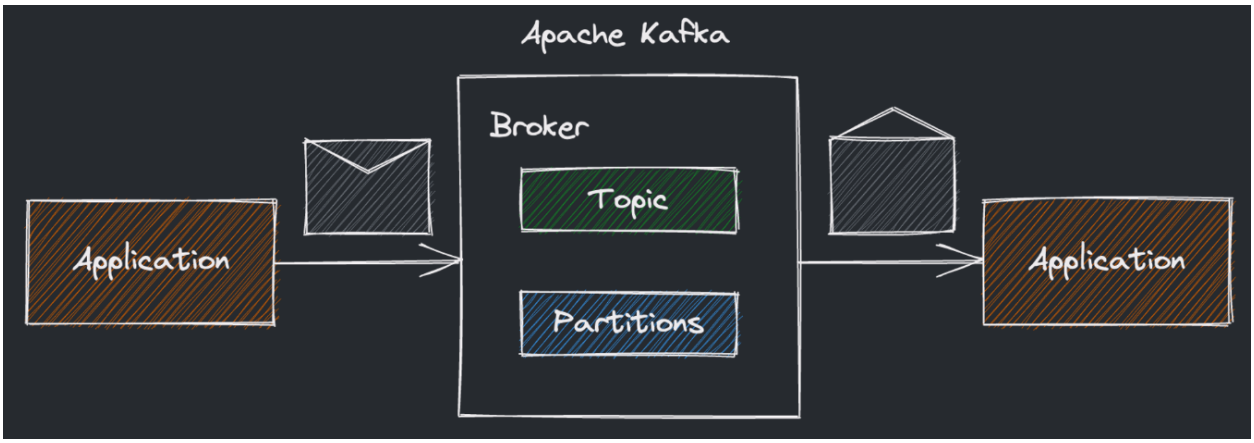


> Apache Kafka

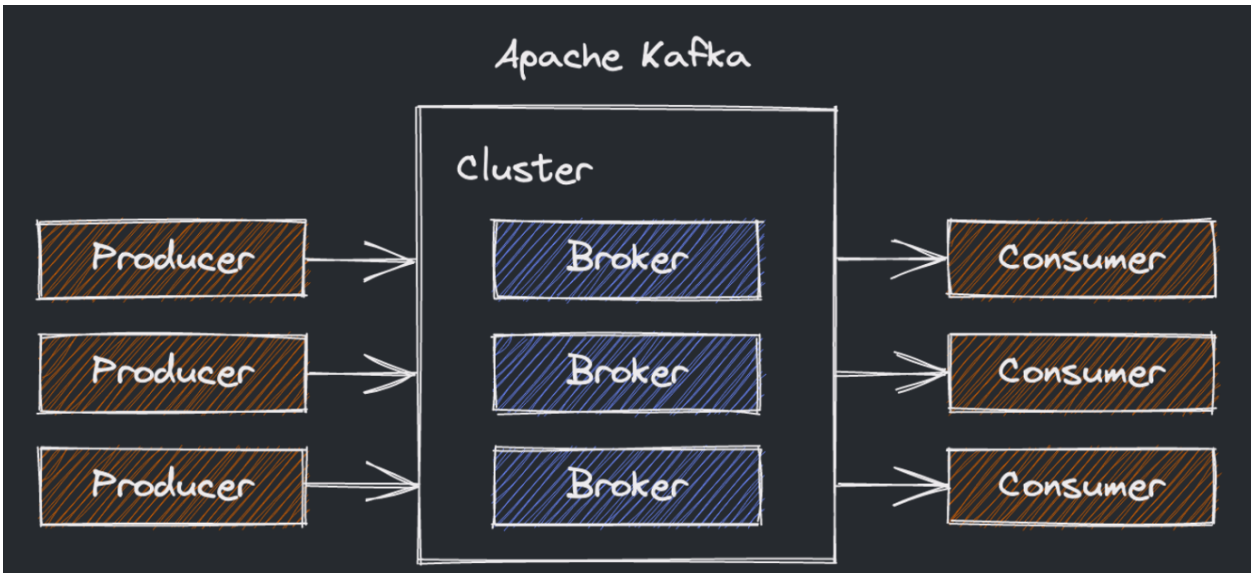
Устройство

- Приложения может подключаться к системе и добавлять записи (records) в создаваемые топики (topics)
- Другие приложения могут подключаться, чтобы обрабатывать накопленные записи в топиках
- Присланные данные сохраняются в течение заданного в системе срока (может даже и месяца)

- Записи обязательно имеют ключ и значение (для ключа хранится последнее значение), а наличие временных меток и заголовков опционально



- Кластер в Кафке состоит из одного или нескольких серверов (брокеров Кафки)
- Продьюсеры присылают (push) записи в топики в рамках брокера
- Консьмеры забирают (pull) записи из выбранного топика
- Можно работать и с одним брокером, но кластер дает больше возможностей (например, репликацию)
- Брокеры управляются отдельными приложениями в рамках системы – Zookeeper



Топики

- Топик это название категории/фида, в котором публикуются и хранятся записи. Все запись разделены по топикам
- Продьюсеры пишут в топики, консьюмеры забирают записи из них. Записи хранятся в течение настроенного срока
- Кафка фактически хранит весь лог из записей и это задача консьюмеров следить за отступами (offset) в топиках
- Обычно консьюмеры просто сдвигают отступ по мере прочтения записей, но могут и откатить при необходимости
- Кафка-топики разбиваются на определенное количество партиций, которые хранят записей последовательно
- Топики тем самым могут быть распараллелены по нескольким брокерам
- Репликация также работает на уровне партиций, у каждой партиции есть реплика-лидер и её последователи
- Продьюсеры присылают записи лидеру, он коммитит записи в лог и только потом они становятся видны консьюмерам
- Продьюсеры сами определяют, в какую партицию им нужно писать, для этого можно добавить ключ к записи (или хэш)

Низкоуровневые консьюмеры

- Сами должны выбрать топик и партицию, а также отступ, с которого начинается чтение
- В качестве отступа можно выбрать как конкретную позицию, так и начало или конец топика

Высокоуровневые консьюмеры

- Группа консьюмеров, состоящая из одного или нескольких консьюмеров с одним свойством "group.id"

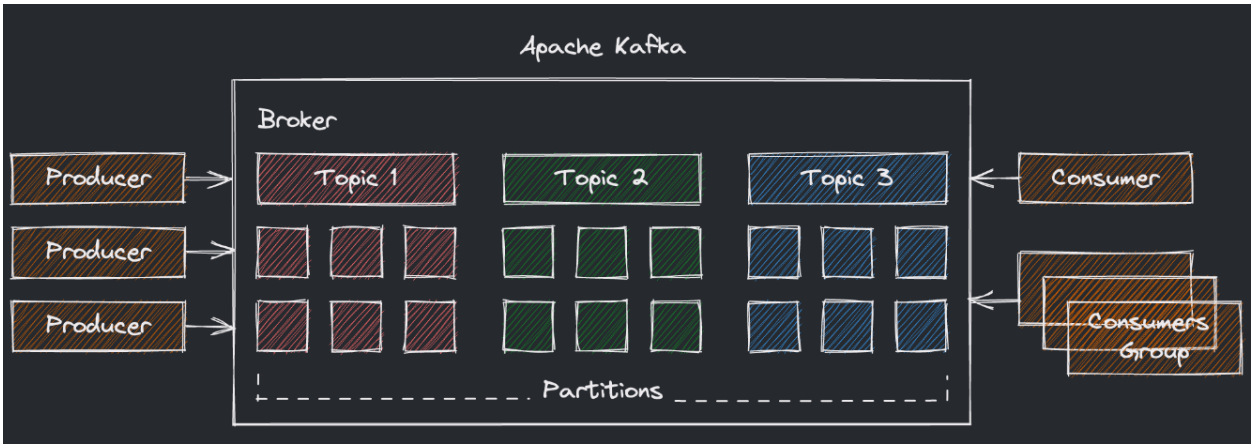
- Брокер распределяет и перераспределяет партиции между консьюмерами из одной группы
- Брокер также следит за оффсетами группы во всех партициях
- Количество консьюмеров в группе может быть большим, но не больше числа партиций

Консьюмеры всегда сами запрашивают записи, когда они готовы к их обработке!

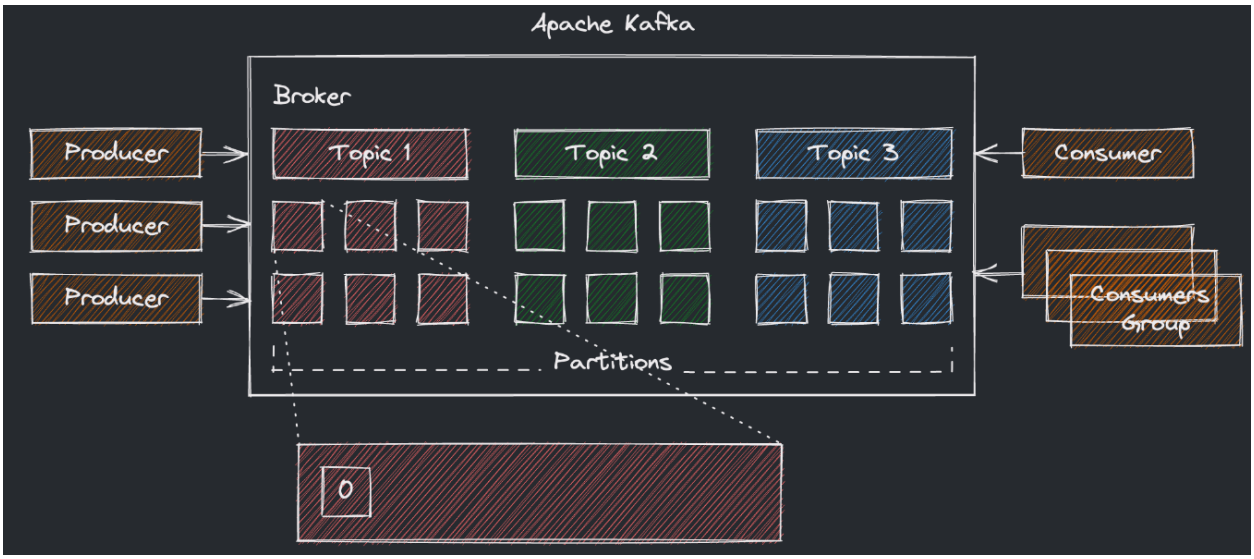
Все записи хранятся в Кафке, поэтому у консьюмеров не случается перегрузов, они всегда могут «нагнать» топик.

Записи

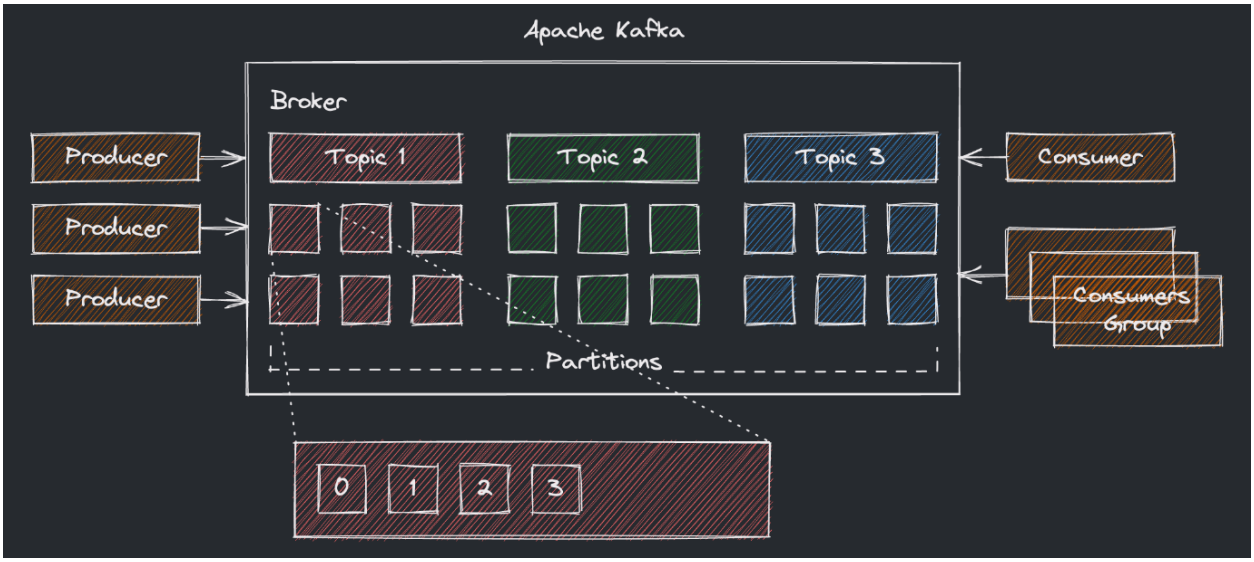
Топики разбиты на какое-то количество партиций:



6 партиций в примере на рисунке



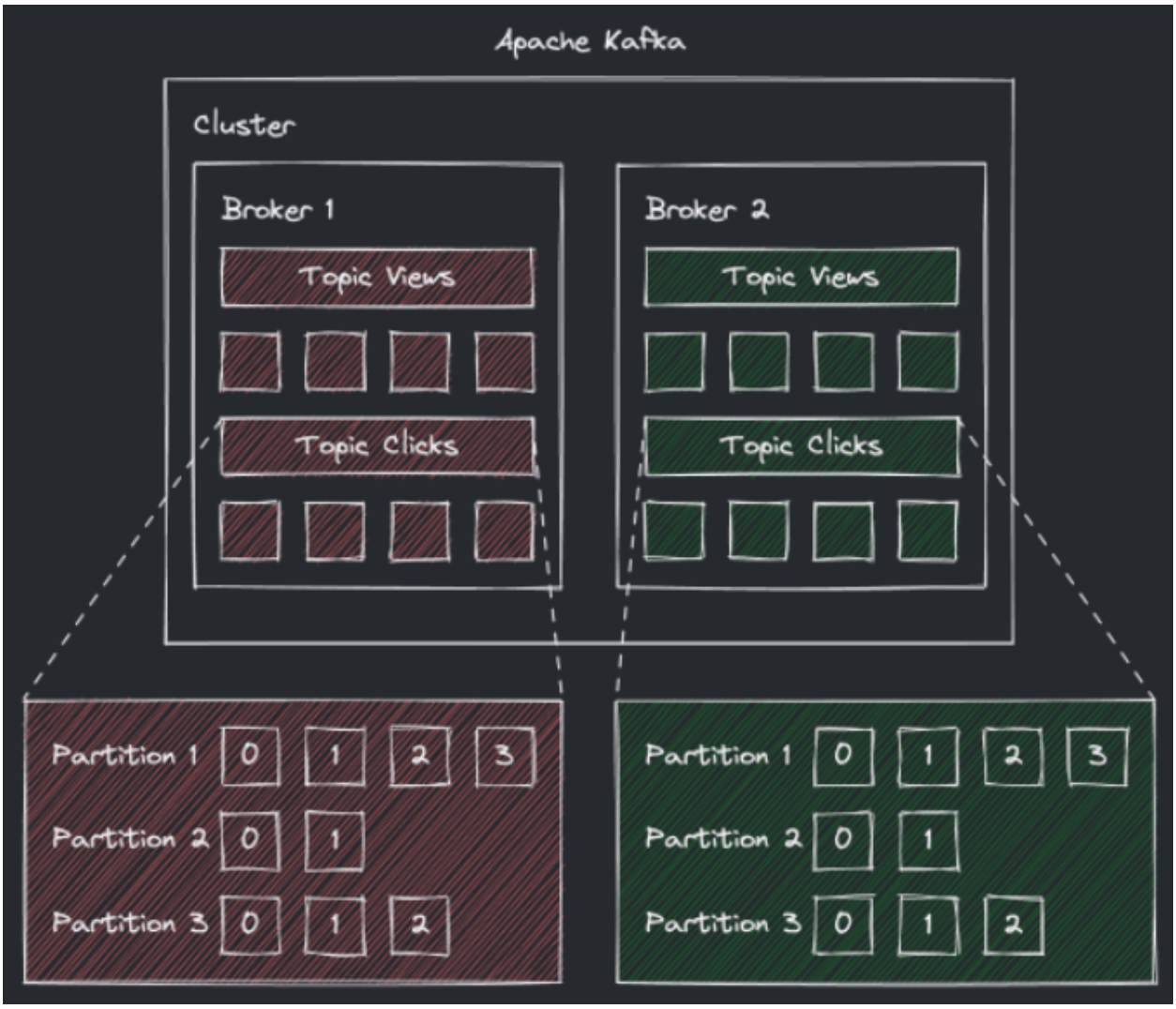
Первое сообщение в первую партицию первого топика записывается с оффсетом 0



Дальнейшие сообщения в ту же партицию будут иметь оффсеты 1, 2, 3 и т.д. (записи хранятся в порядке поступления)

Пример приложения

Собираем пользовательские логи, включающие данные о просмотрах и кликах по баннерам.



Распределение партиций по разным брокерам

Об одном полезном свойстве

Для каждого конкретного ключа хранится только последняя запись с этим ключом, остальные удаляются. Таким образом происходит оптимизация хранимого лога событий внутри Kafka. Консьюмеры начинают чтение с последнего значения по выбранному ключу и затем получают обновления.

Это можно использовать для восстановление кэша или получения разных статусов из источника правды в виде Кафки.

> RabbitMQ vs Kafka

Для чего лучше подходит Kafka

- Лучше всего для стриминга из A в B без сложной маршрутизации, но с максимальной производительностью
- Идеально для логирования событий, обработки потока и отслеживания последовательных изменений в системе
- Подходит для обработки данных в многоуровневых пайплайнах

Кратко: используем для хранения и последующей, возможно повторной, обработки и анализа потоковых данных.

Для чего лучше подходит RabbitMQ

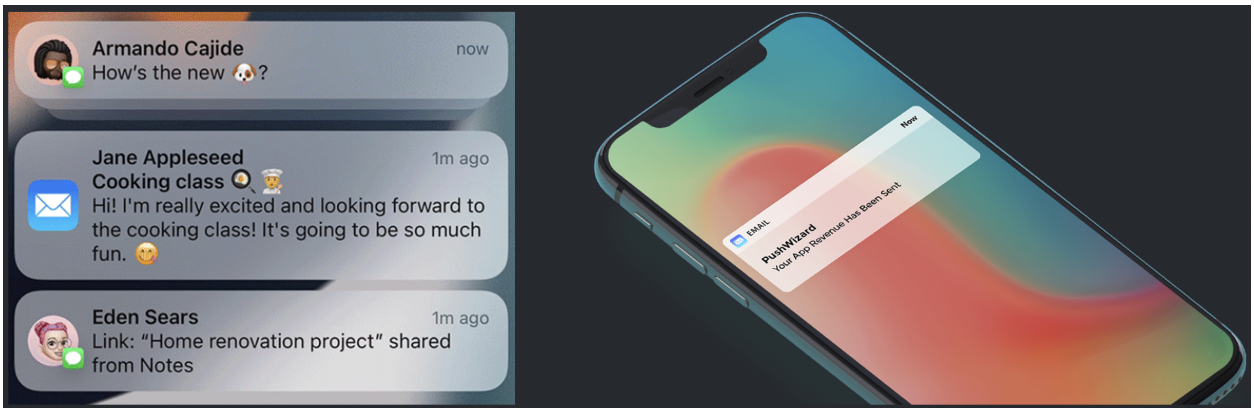
- Применяется для тяжелых фоновых задач, а также коммуникации внутри сложных приложений
- Идеально для веб-сервисов с низкими задержками благодаря делегированию нагрузки множеству воркеров
- Также подходит для фоновых и долгоиграющих задач вроде конвертации видео или обработки изображений

Кратко: используем с долгими фоновыми задачами и для общения и интеграции между различными сервисами.

> Сервис уведомлений

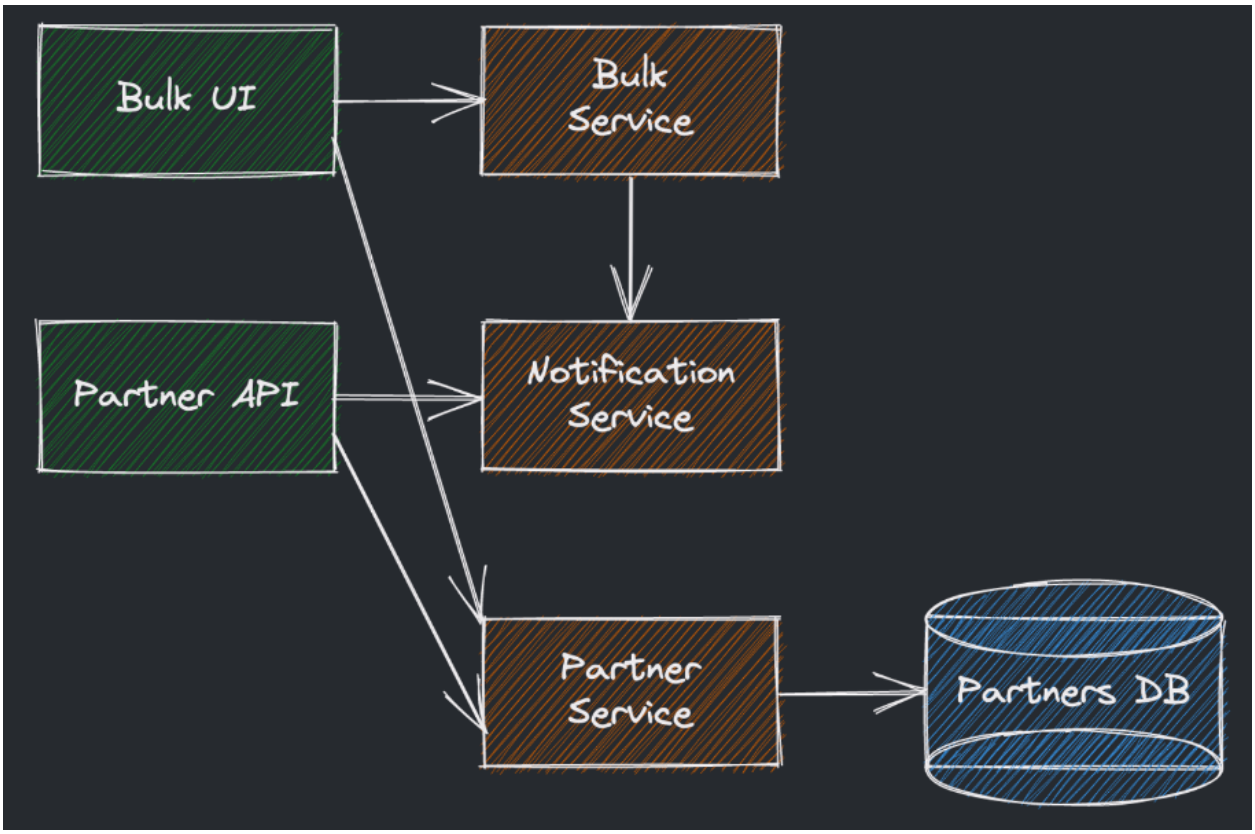
Зачем нужен такой сервис?

Чтобы приложения могли доставить нам ну очень важную информацию даже на утюг или холодильник.

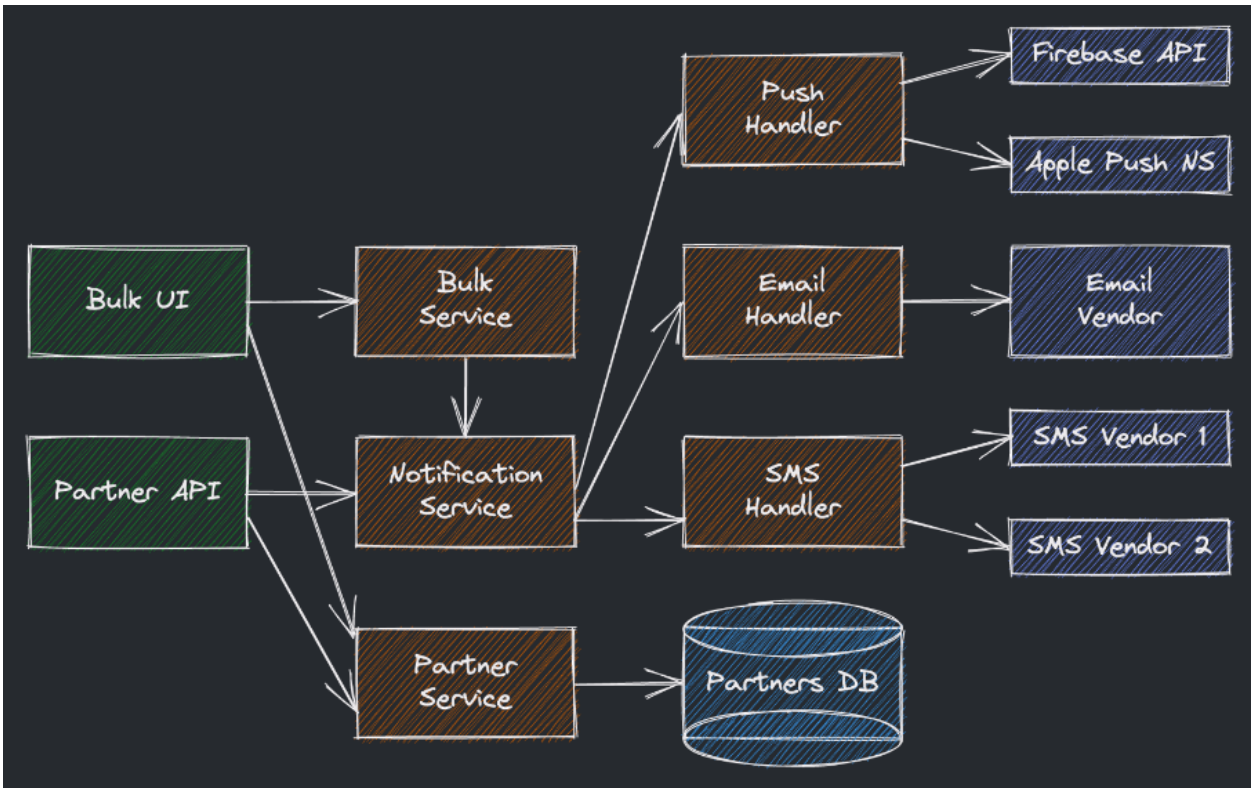


Какие сценарии отобразим?

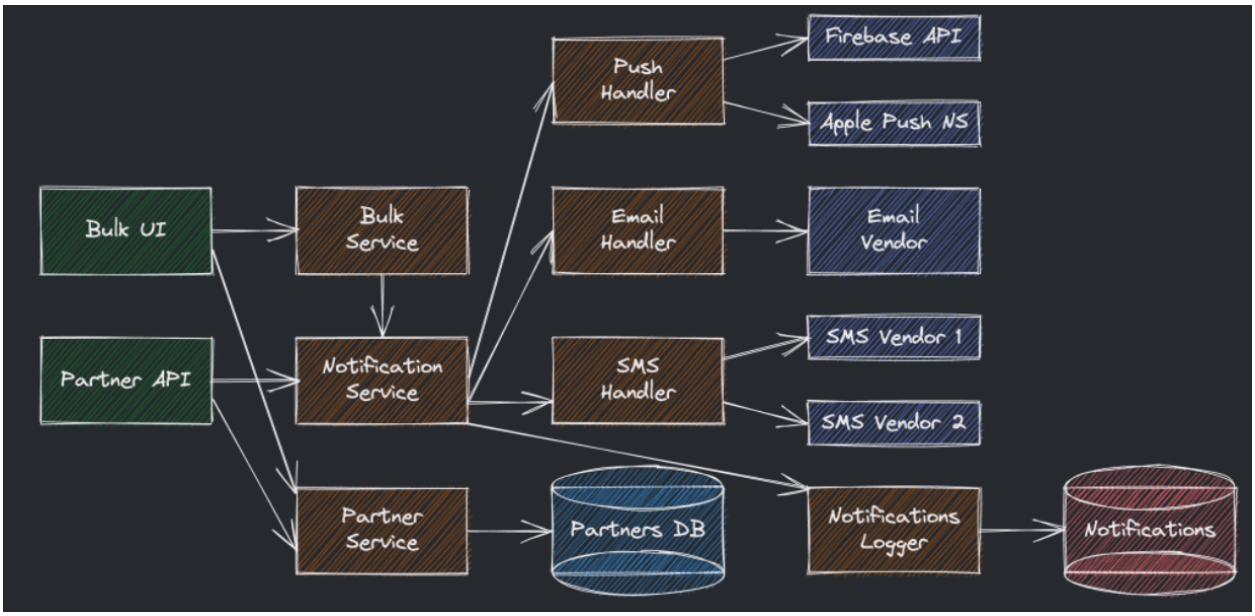
- Партнеры могут делать рассылки либо через интерфейс, либо через API



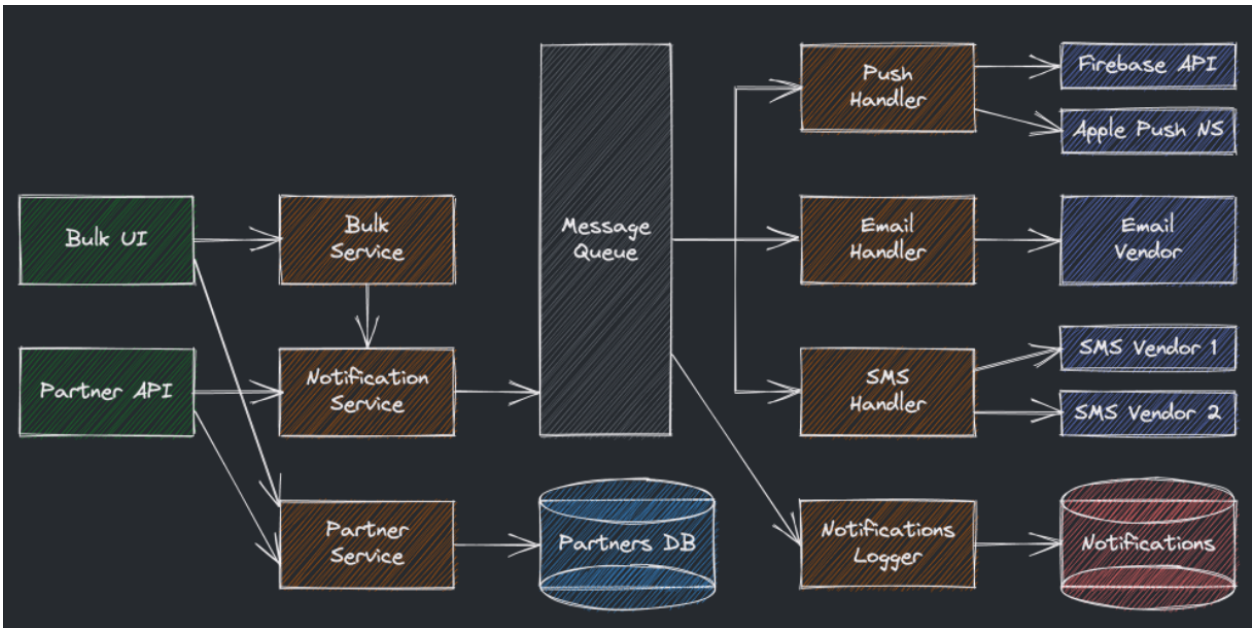
- Рассылки должны производиться на какие только возможно устройства пользователя



- Информация обо всех рассылках должна сохраняться на будущее на всякий случай



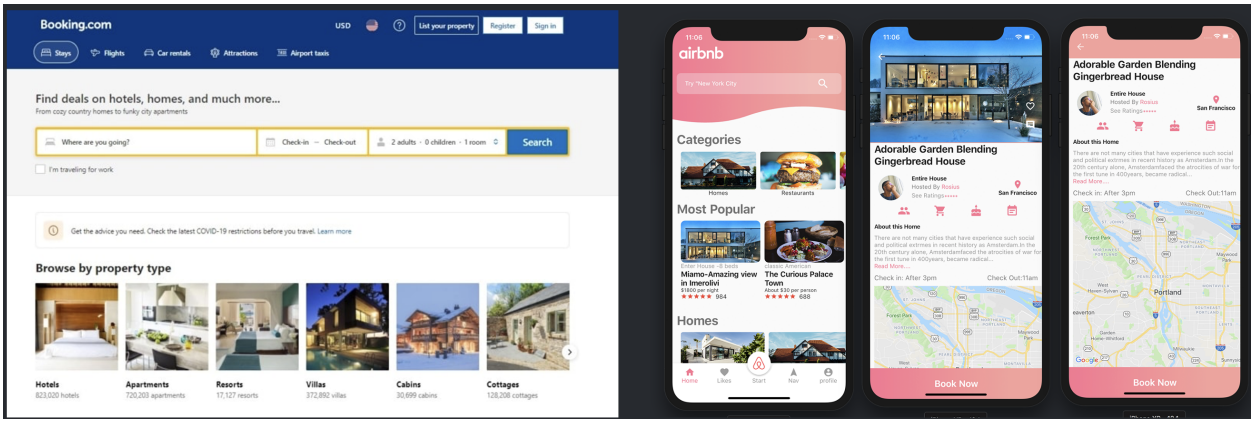
Добавим очередь сообщений: разъединяем зависимые сервисы, выделяем естественные фоновые процессы.



> Сервис бронирования

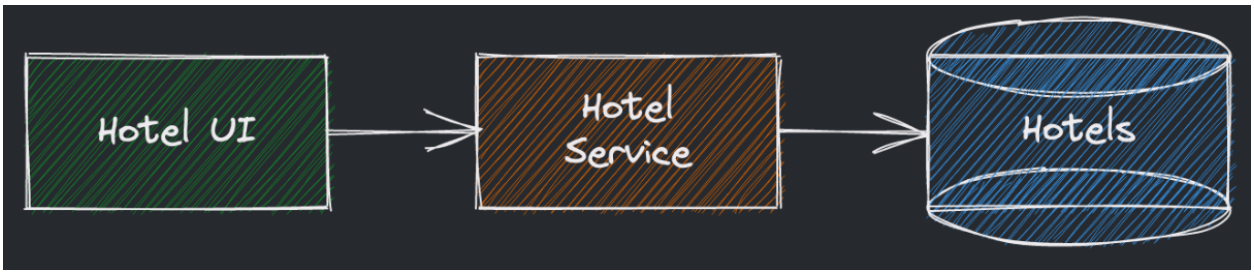
Зачем нужен такой сервис?

Чтобы было где пережидать периоды структурных трансформаций и поисков новых моделей бизнеса.

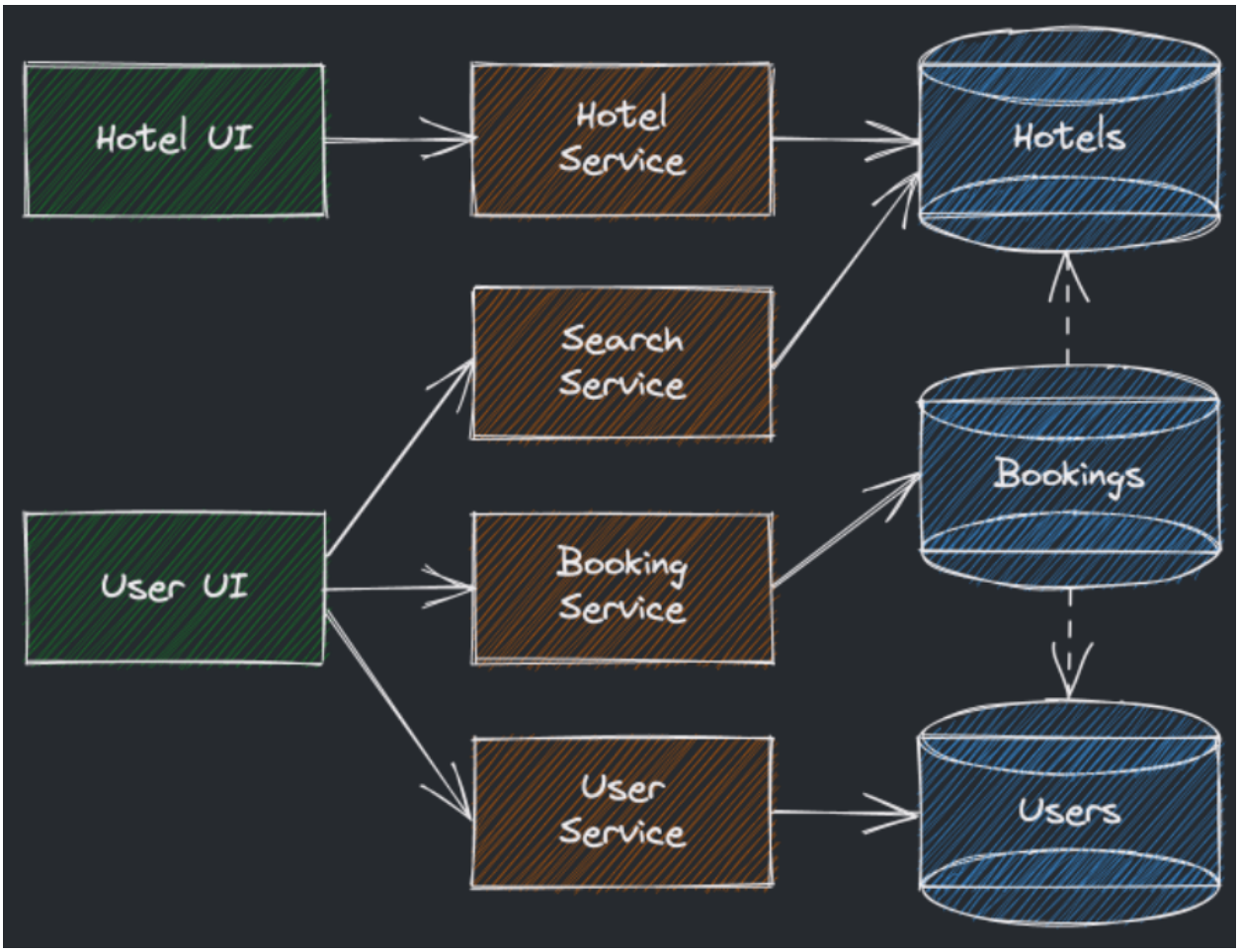


Какие сценарии отобразим?

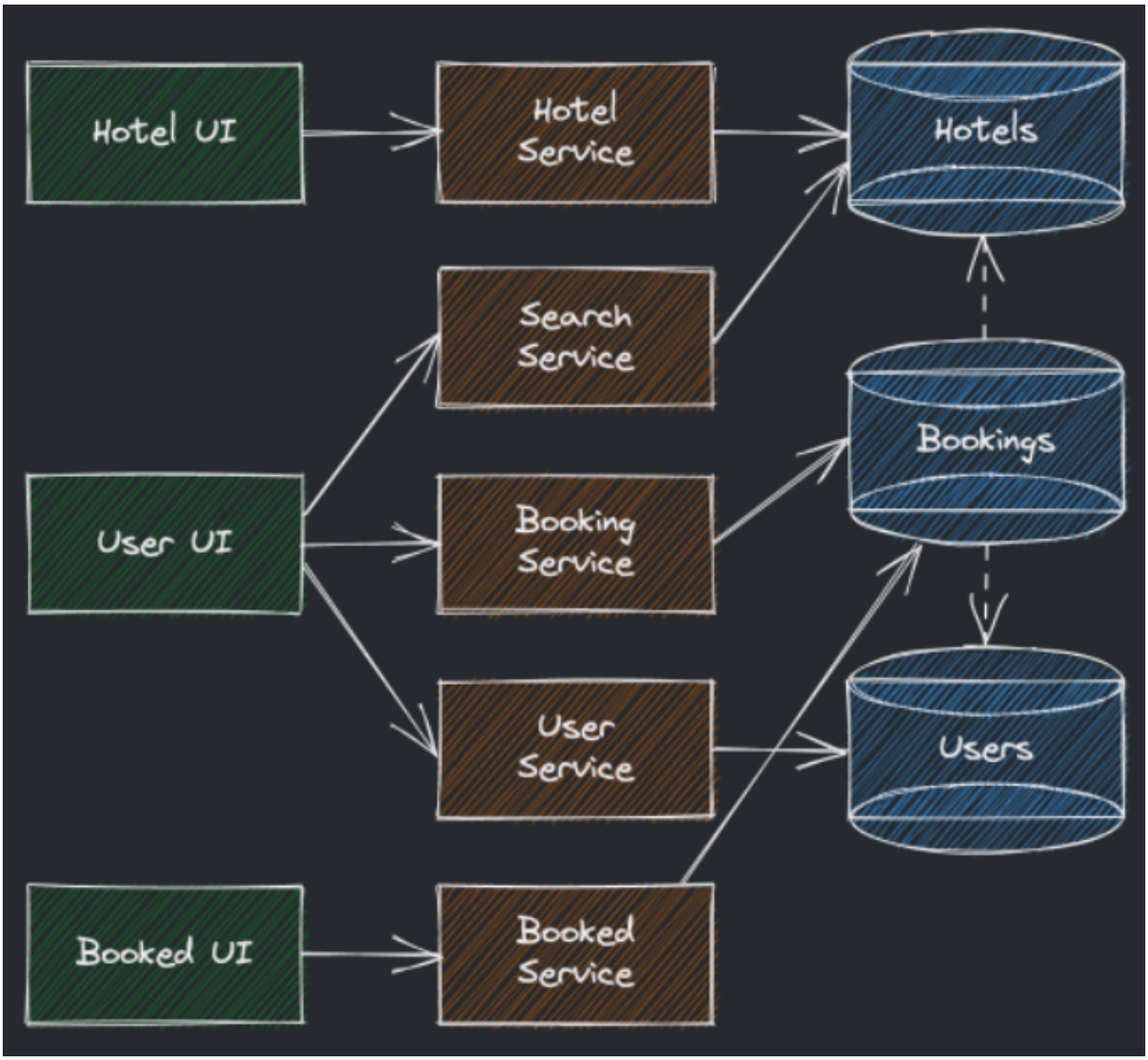
- Владельцы отелей регистрируются и пополняют жилой фонд



- Пользователи регистрируются, могут искать и бронировать жилье

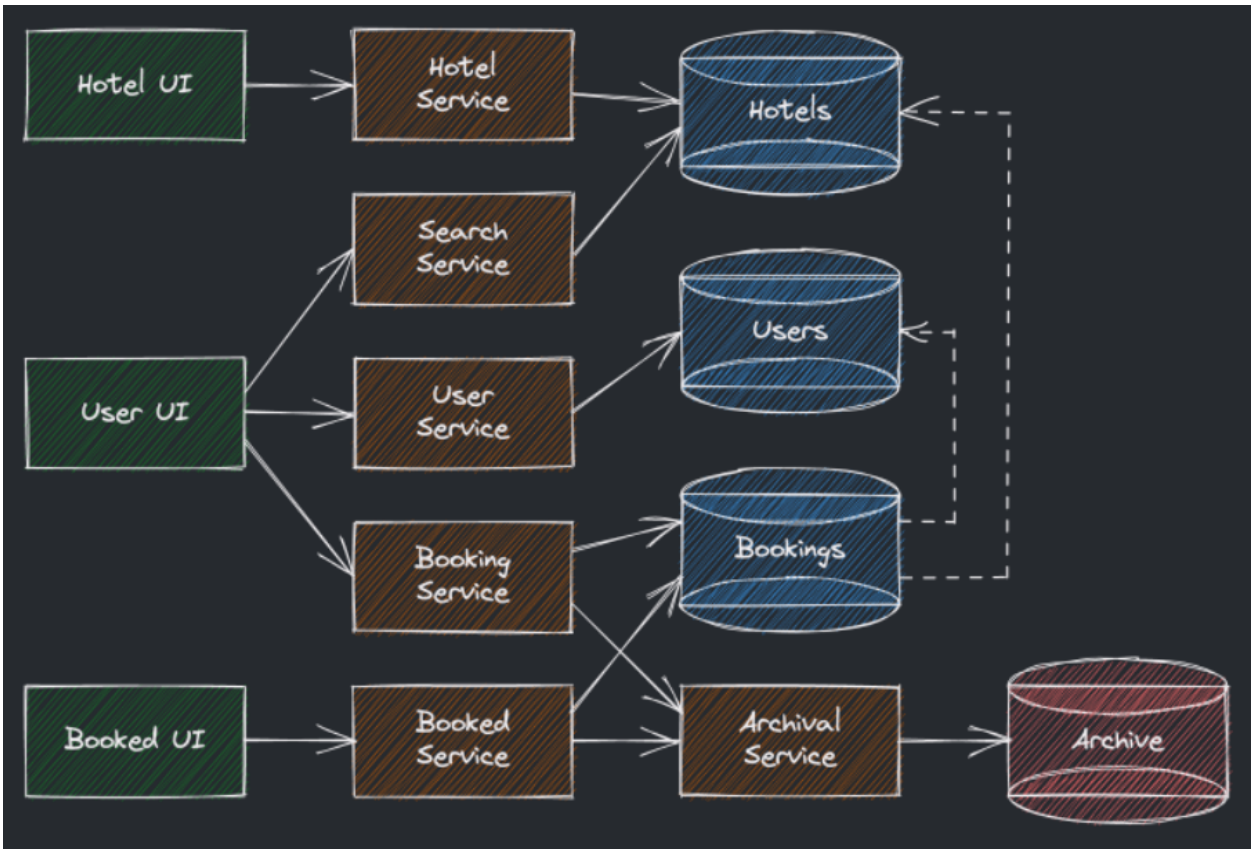


- И владельцы, и пользователи могут редактировать бронь

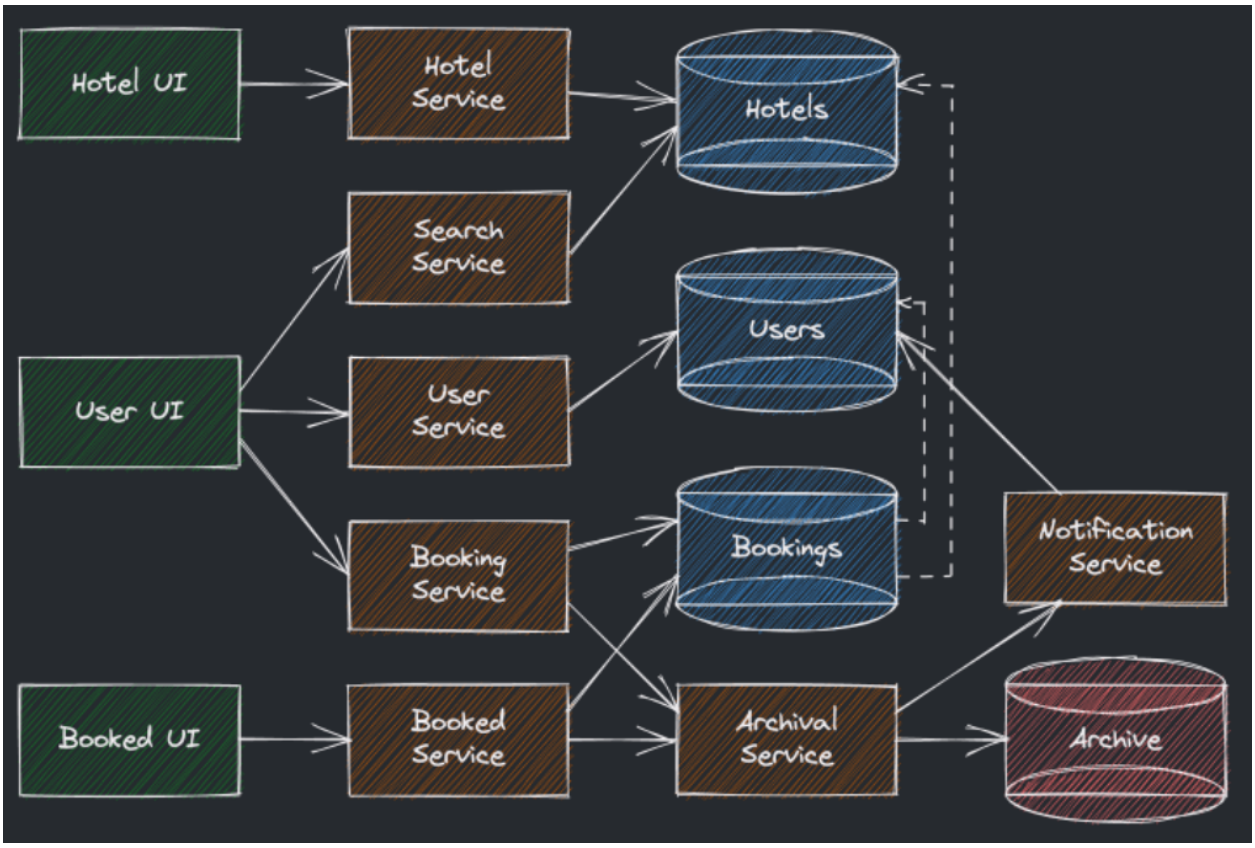


Дополнительно:

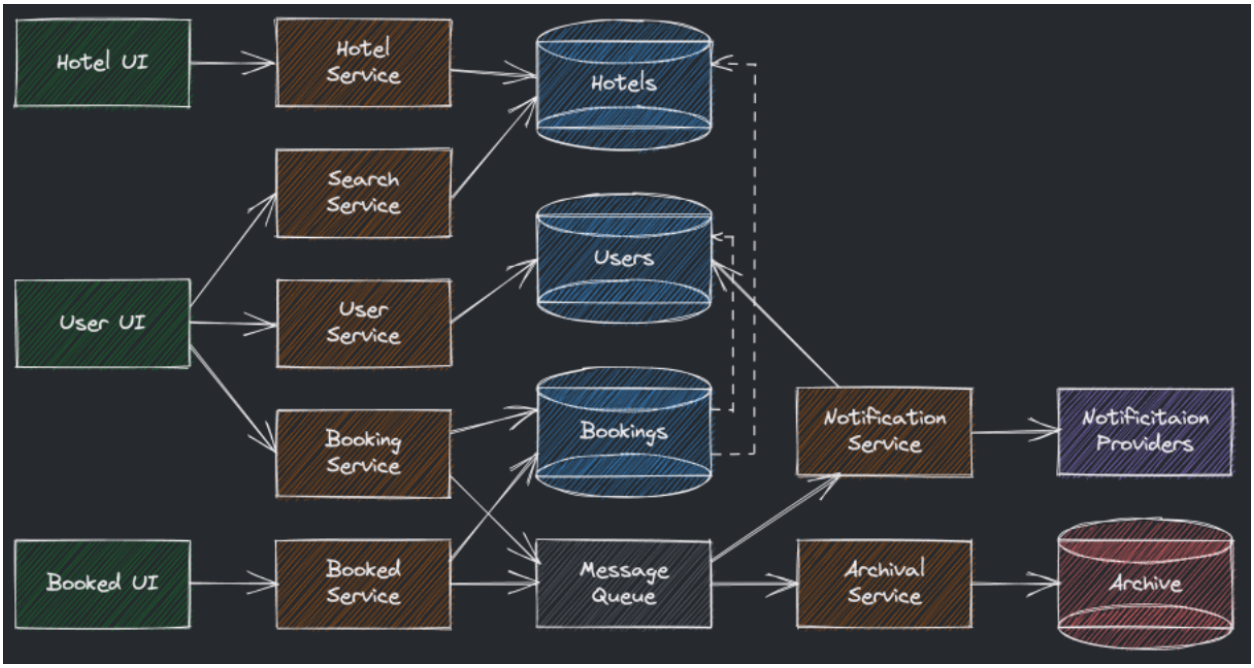
- После завершения брони её можно положить в архив



- При изменениях в брони уведомляем пользователя



- Добавляем очередь сообщений: отсоединяем добавленные в качестве приправы сервисы и общаемся с ними через очередь



> Полезные материалы

- RabbitMQ for beginners
- Apache Kafka for beginners