Архитектура банковских ПОИС, часть 2

Информационное обеспечение предметно-ориентированных ИС

Дальнейшее развитие SOA. Очередь сообщений.

Применяется в случае наличия нескольких приложений, асинхронно (получение не обязательно произойдет сразу после отправления) общающихся друг с другом с помощью сообщений, не привязанных к платформе. Для функционирования необходимы:

- один язык обмена сообщениями,
- определенный текстовый формат представления данных.

Архитектура системы построенной по принципу “очередь сообщений”

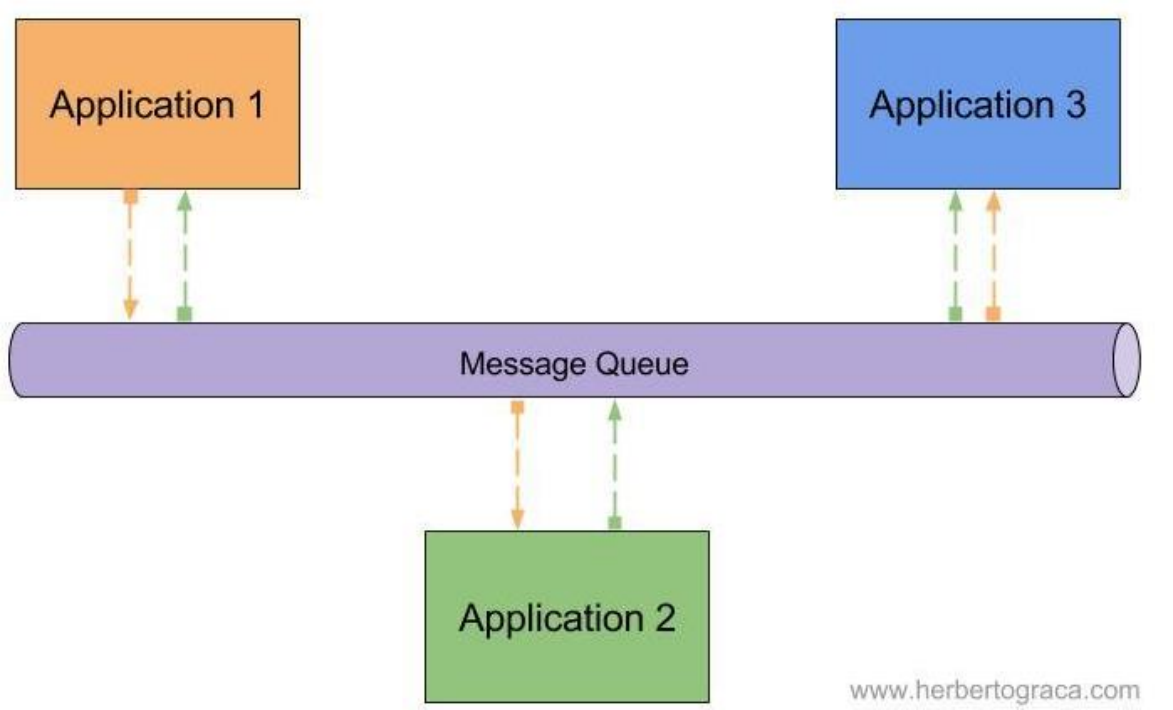
Очередь сообщений использует в качестве компонента инфраструктуры программный брокер сообщений (RabbitMQ, Beanstalkd, Kafka и т. д.).

Брокер сообщений является центральным звеном архитектуры. Его основные задачи:

- регистрация программной процедуры, которая «слушает» сообщения, помещённые в очередь;
- сохранение сообщения до тех пор, пока принимающее приложение не подключится;
- вызов зарегистрированной программной процедуры.

Потоки данных в общем виде

В общем виде, схему функционирования очереди сообщений можно показать так:



Типы настройки очереди сообщений (message queue).

1. **Запрос/Ответ.**
   Клиент шлёт в очередь сообщение, включая ссылку на *«разговор» («conversation» reference)*. Сообщение приходит на специальный узел, который отвечает отправителю другим сообщением, где содержится ссылка на тот же *разговор*, так что получатель знает, на какой *разговор* ссылается сообщение, и может продолжать действовать.
2. **Публикация/Подписка.**

   По спискам
   Очередь поддерживает списки опубликованных тем подписок (topics) и их подписчиков. Когда очередь получает сообщение для какой-то темы, то помещает его в соответствующий список. Сообщение сопоставляется с темой по типу сообщения или по заранее определённому набору критериев, включая и содержимое сообщения.

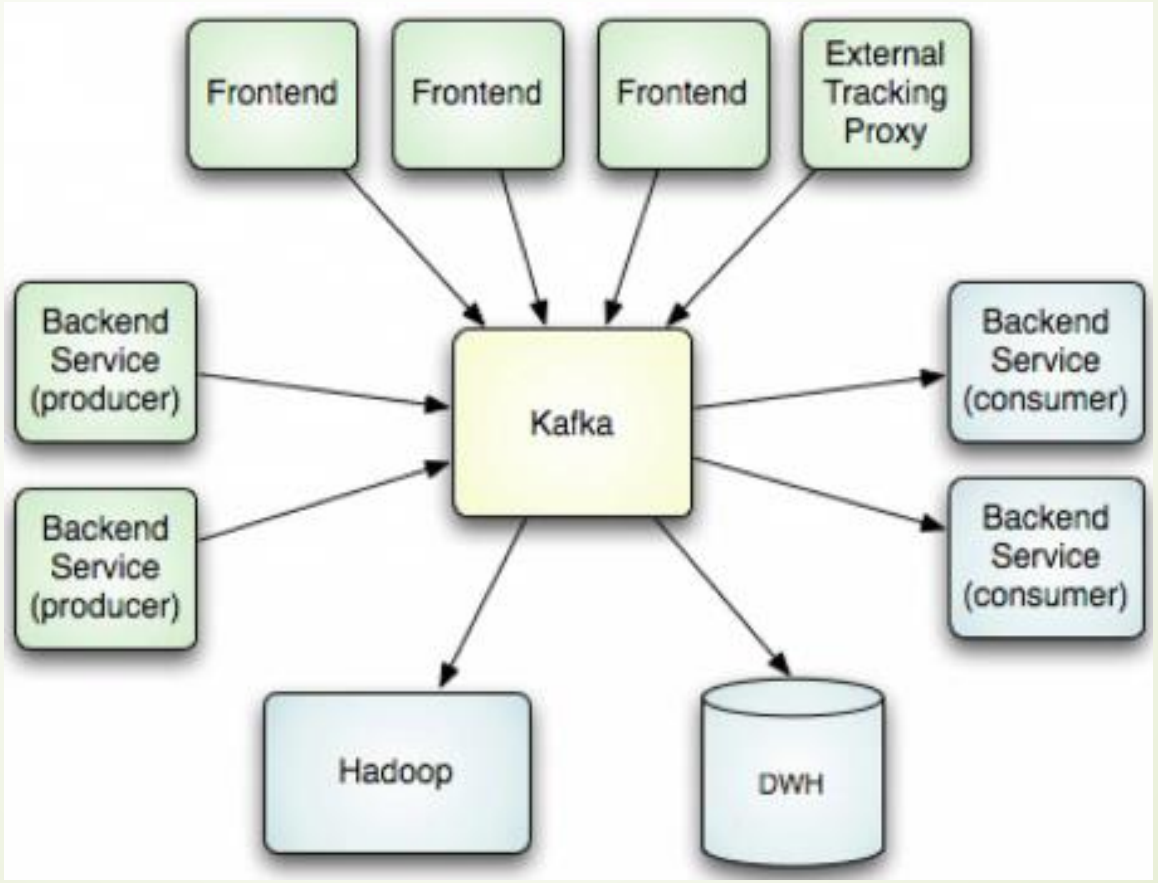
   На основе вещания
   Когда очередь получает сообщение, она транслирует его всем узлам, прослушивающим очередь. Узлы должны сами фильтровать данные и обрабатывать только интересующие сообщения.

Сценарии действий клиента очереди сообщений

1. В pull-сценарии клиент опрашивает очередь с определённой частотой. Клиент управляет своей нагрузкой, но при этом может возникнуть задержка: сообщение уже лежит в очереди, а клиент его ещё не обрабатывает, потому что не пришло время следующего опроса очереди.

2. В push-сценарии очередь сразу же отдаёт клиентам сообщения по мере поступления. Задержки нет, но клиенты не управляют своей нагрузкой.

Архитектура очереди сообщение на базе брокера Apache Kafka

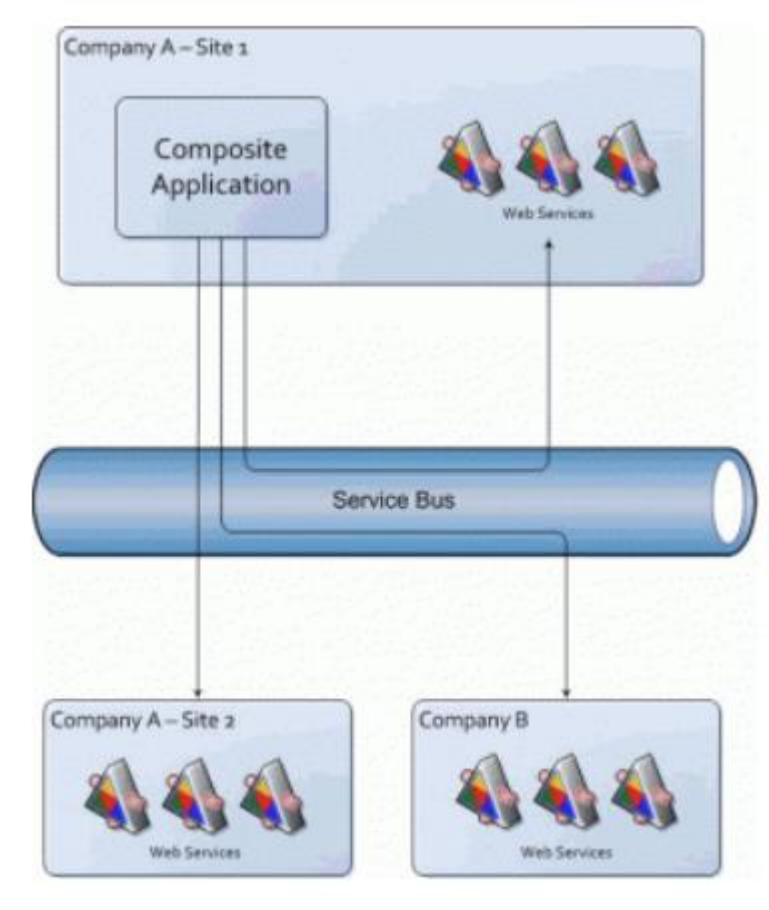


Эволюция очереди сообщений. Сервисная шина предприятия (ESB).

В этой архитектуре используется модульное приложение (composite application), обычно ориентированное на пользователей, которое общается с веб-сервисами для выполнения каких-то операций. В свою очередь, эти веб-сервисы тоже могут общаться с другими веб-сервисами, впоследствии возвращая приложению какие-то данные. Но ни приложение, ни бэкенд-сервисы ничего друг о друге не знают, включая расположение и протоколы связи. Они знают лишь, с каким сервисом хотят связаться и где находится сервисная шина.

Клиент (сервис или модульное приложение) отправляет запрос на сервисную шину, которая преобразует сообщение в формат, поддерживаемый в точке назначения, и перенаправляет туда запрос.

Типовая архитектура сервисной шины предприятия



Основные функции ESB

- отслеживать и маршрутизировать обмен сообщениями между сервисами (брокер);

- преобразовывать сообщения между общающимися сервисными компонентами (!!!)

- управлять развёртыванием и версионированием сервисов (брокер)

- управлять использованием избыточных сервисов (брокер)

- предоставлять стандартные сервисы обработки событий, преобразования и сопоставления данных, сервисы очередей сообщений и событий, сервисы обеспечения безопасности или обработки исключений, сервисы преобразования протоколов и обеспечения необходимого качества связи (брокер)

Недостатки ESB

Все они связаны с чрезвычайной сложностью управления такой структурой, как ESB (централизованная логика):

- единая точка отказа, способная обрушить системы связи всей компании.

- большая сложность конфигурирования и поддержки.

- шина так сложна, что для её управления требуется целая команда высококвалифицированных it-специалистов.

- высокая зависимость сервисов от ESB как внутри, так и снаружи IT-контура организации.

Архитектура микросервисов

Эта архитектура определяется как вариант сервис-ориентированной архитектуры программного обеспечения, ориентированный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей - микросервисов.

Главное различие микросервисов и шины в том, что ESB была создана в контексте интеграции отдельных приложений, чтобы получилось единое корпоративное распределённое приложение. Микросервисная архитектура создавалась в контексте быстро и постоянно меняющихся бизнесов, которые (в основном) с нуля создают собственные облачные приложения.

То есть в случае с ESB у нас уже были приложения, которые нам не «принадлежат», и поэтому мы не могли их изменить. А в случае с микросервисами мы полностью контролируем приложения (при этом в системе могут использоваться и сторонние веб-сервисы).

Перевод крутой статьи Фаулера о микросервисах: https://habr.com/ru/post/249183/

Свойства, характерные для микросервисов

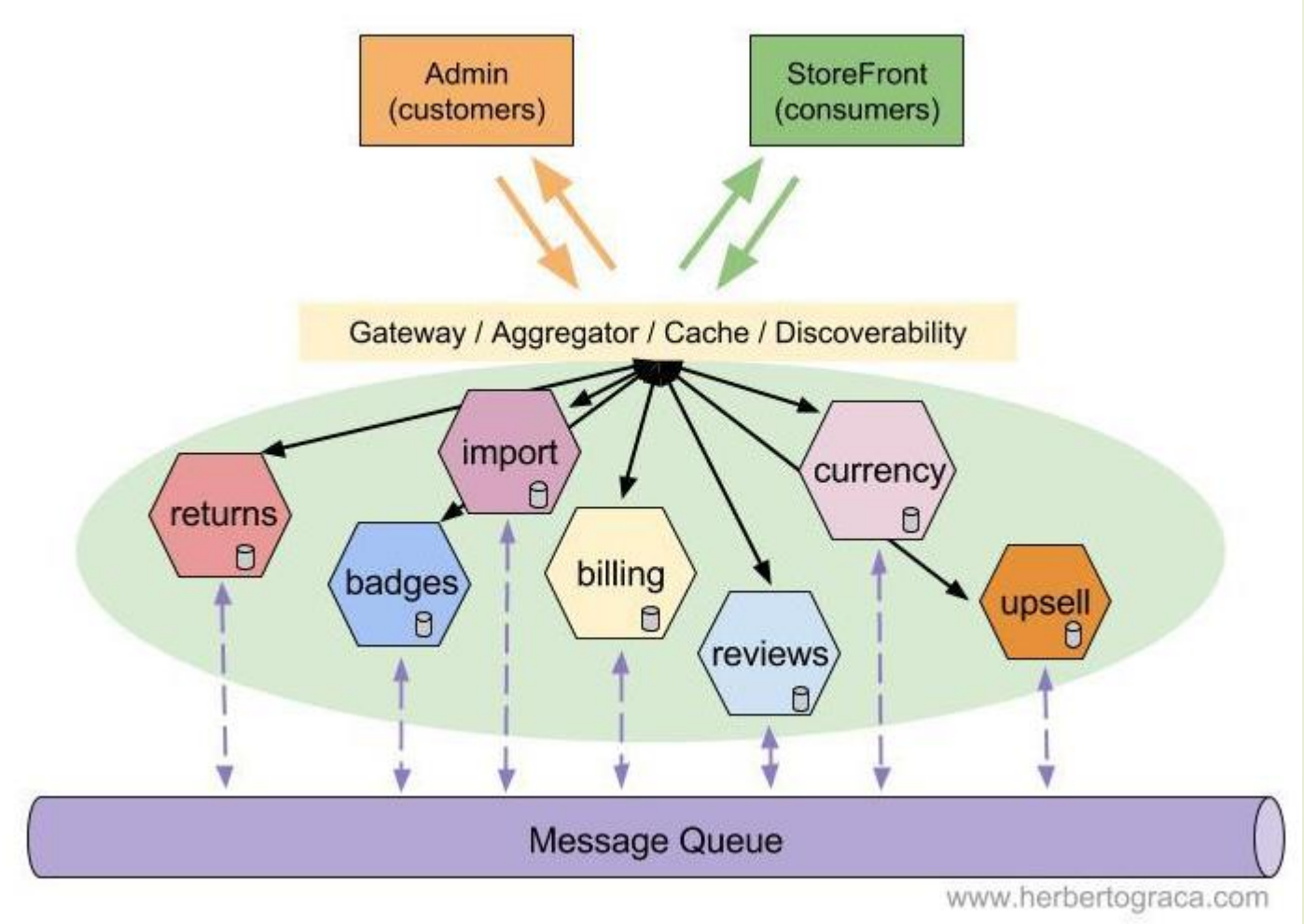
- модули можно легко заменить в любое время: акцент на простоту, независимость развёртывания и обновления каждого из микросервисов;

- модули организованы вокруг функций: микросервис по возможности выполняет только одну достаточно элементарную функцию;

- модули могут быть реализованы с использованием различных языков программирования, фреймворков, связующего программного обеспечения, выполняться в различных средах контейнеризации, виртуализации, под управлением различных операционных систем на различных аппаратных платформах: приоритет отдаётся в пользу наибольшей эффективности для каждой конкретной функции, нежели стандартизации средств разработки и исполнения;

- архитектура симметричная, а не иерархическая: зависимости между микросервисами одноранговые.

Типовая архитектура микросервисов



Принципы микросервисной архитектуры Ньюмана

- **Проектирование сервисов вокруг бизнес-доменов**
  Это может дать стабильные интерфейсы, высокосвязные и мало зависящие друг от друга модули кода.
- **Культура автоматизации**
  Это даст гораздо больше свободы, можно развернуть больше модулей.
- **Скрытие подробностей реализации**
  Это позволяет сервисам развиваться независимо друг от друга.
- **Полная децентрализация**
  Децентрализованное принятие решений и архитектурные концепции, организация сама превращается в сложную адаптивную систему, способную быстро приспосабливаться к переменам.

Принципы микросервисной архитектуры Ньюмана

- **Независимое развёртывание**
  Можно развёртывать новую версию сервиса, не меняя ничего другого.
- **Сначала потребитель**
  Сервис должен быть простым в использовании, в том числе другими сервисами.
- **Изолирование сбоев**
  Если один сервис падает, другие продолжают работать, это делает всю систему устойчивой к сбоям.
- **Удобство мониторинга**
  В системе много компонентов, поэтому трудно уследить за всем, что в ней происходит. Введение более сложных инструменты мониторинга, позволяющих заглянуть в каждый уголок системы и отследить любую цепочку событий.