УДК 004.05

МИКРОСЕРВИСНЫЕ АСИНХРОННЫЕ АРХИТЕКТУРЫ КАК РЕШЕНИЕ СЛОЖНЫХ БИЗНЕС-ПРОЦЕССОВ

П.Д. ТАЛАЙКО
(Представлено: канд. техн. наук, доц. И.Б. БУРАЧЕНОК)

Рассмотрены особенности взаимодействия программных компонентов в асинхронных микросервисных архитектурах. Произведено знакомство и спецификой работы с современными Message Brokers. Обоснована целесообразность использования асинхронного подхода в распределённых системах. Рассмотрен основной принцип работы одной из самых популярных очередей сообщений Kafka.

Часто при разработке программных продуктов приходится сталкиваться с реализацией каких-то бизнес-задач в рамках сложных процессов. IT-сообщество (от англ. Information Technology) придумало множество различных решений основанных на декомпозиции таких процессов. С появлением инструментов и вычислительных возможностей в построении высоконагруженных архитектур, многие разработчики разбили свои огромные приложения (монолиты) на модули, работающие над определенным подпроцессами, но столкнулись с рядом проблем в ходе создания микросервисной архитектуры (МА), а именно:

- усложненное взаимодействие модулей системы;
- управление транзакциями между модулями;
- дорогая стоимость поддержки (каждый модуль должен иметь свой набор ресурсов);
- усложнение разработки и запуска ПО (CI/CD (англ. Continuous Integration/Continuous delivery), Jenkins, TeamCity, Docker, OpenShift и т. д.).

Правильно обозначенные задачи, позволят правильно выстроить процессы и подпроцессы. Декомпозиция должна происходить строго по линии данных и функций, за которые отвечает данный программный компонент. Для примера можем провести модуль, отвечающий за авторизацию и аутентификацию пользователя в системе. Данный модуль назовём «security-service» и возложим на него определенные функции и данные. Данными, которыми будет оперировать модуль – эта информация о пользователе и его роли в системе. Может возникнуть справедливый вопрос, как нам получать данные пользователя в рамках другого сервиса и как взаимодействовать с «security-service» модулем? Ответ очень прост – вынести логику авторизации и аутентификации в общий инфраструктурный компонент и подключать его к различным модулям в зависимости от необходимости получения данных о пользователе и его ролях.

Инфраструктура микросервисной архитектуры – это набор сервисов или библиотек, которые направлены на вынесение реализации общей функциональности. К сервисам инфраструктура можно отнести сервис конфигураций «config-service» и сервис регистрации сервисов архитектуры «discovery-service». Основная концепция направлена на централизацию конфигураций с помощью «config-service» и обращение к сервису по его имени, независимо от места физического запуска сервиса, за это отвечает «discovery-service». Для создания инфраструктуры существуют готовые решения, и одно из таких представлено во framework Spring в модуле Cloud или коротко Spring Cloud [1].

После того как централизована конфигурация и к сервису можно обратиться по имени, не зная где данный сервис физически запущен, перейдём к главной теме асинхронному взаимодействие микросервисов между собой. Долгое время закладывалась концепция о клиент-серверном взаимодействии – «совершая запрос в синхронном режиме получаем ответ (результат)». Представим, когда необходимо выполнить ряд операций по одному запросу, допустим, пользователь в личном кабинете нажимает кнопку «Получать уведомления» и сервис получает событие о начале отправки уведомлений для конкретного пользователя. Уведомления могут приходит пользователю на почту, в телеграм канал, мобильный телефон и все источники должны уметь отправлять пользователю уведомление в соответствующий канал. Следует отметить, что после нажатия пользователем кнопки «Получать уведомления», отправится запрос на изменение статуса уведомления в систему, после чего, его данные попадут в выборку участвующих в рассылке уведомлений, а конкретный сервис, умеющий работать с мессенджерами, почтой, мобильными рассылками будет сообщать о происходящих событиях.

Каждый сервис может иметь собственный компонент представления (JSP, JS, HTML, CSS), через который происходит взаимодействие с пользователем.

Далее подробнее рассмотрим архитектуру, представленную на рисунке 1. Пользователь отправляет HTTP запрос и получает синхронно ответ HTTP 200 OK о включении его в список подписчиков уведомлений, сервис «user-service» изменяет в схеме <project_name>_USER базы данных информацию о статусе подписки, а другой модуль высылающий уведомления сервисам рассылки в фоновом процессе ожидает сообщение от message broker. При поступлении нового уведомления производит создание сообщений по

средствам выборки пользователей, нуждающихся в уведомлениях, и собственно контента уведомления. Последним этапом формируется запрос через другую очередь сообщений для конкретных рассыльщиков (телеграм, почта, мобильный сервис).

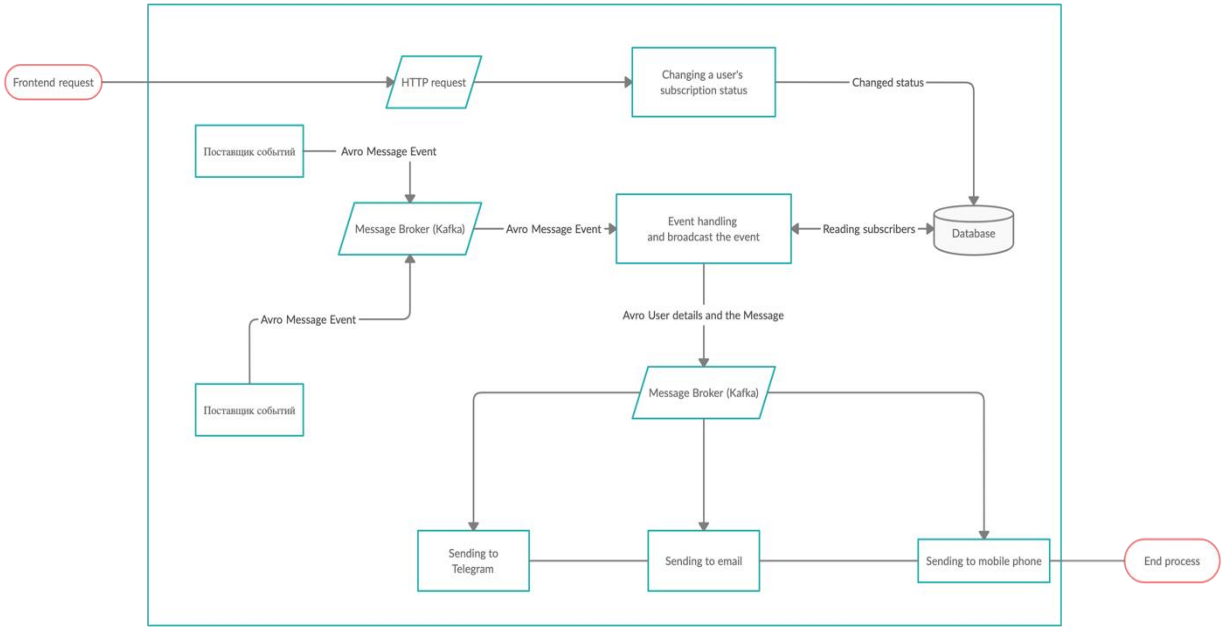

Рисунок 1. – Архитектура рассылки уведомлений пользователю

Далее рассмотрим, как могут взаимодействовать различные сервисы в микросервисной архитектуре. Выше было описано одно из возможных взаимодействий посредством общей базы данных, но в большинстве случаев необходимо просто передать другому сервису информацию, не отображая и не храня её.

Рассмотрим одну из самых известных очередей сообщений (message broker) Kafka. Схематичный принцип работы данного инструмента представлен на рисунке 2.

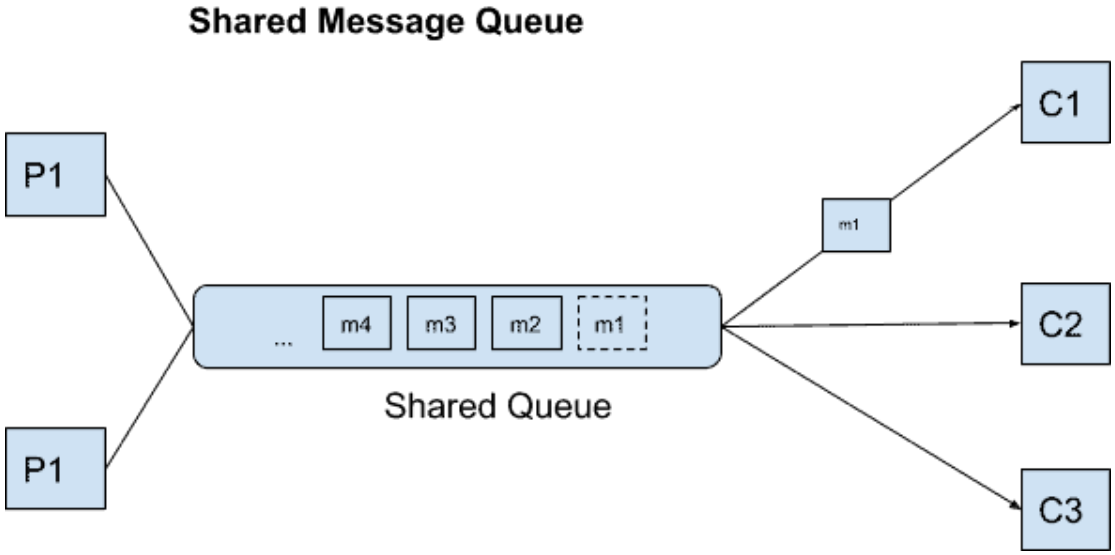

Рисунок 2. – Схематичный прототип работы Kafka

Kafka позволяет обмениваться сообщениями в больших объёмах и при этом хранить состояние очереди. Главная концепция заключается в наличии поставщиков, потребителей и групп потребителей. Лёгкая интеграция с любой платформой делает данную очередь сообщений инструментом по умолчанию во многих проектах. Необходимо чётко понимать то, какие сообщения необходимы каким сервисам и правильно

определять группы сервисов, так как от этого зависит получат ли они из очереди сообщения. Если обратиться к примеру архитектуры выше и объединить наших отправителей сообщений пользователю в одну группу, то сообщение будет отправляться только одним из сервисов, так как остальным сообщение из очереди не будет доступным. Элемент очереди представлен в виде набора byte, поэтому если работаем с определенным видом данных как String, Number, Object, то необходимо определить сериализаторы и десериализаторы для того или иного типа данных. Многие реализации producer-consumer под различные платформы для Kafka имеют встроенные инструменты сериализации и десериализации, поэтому работать с данным брокером становится только приятнее. Стоит отметить, что существует возможность передавать объекты по каналу, и в мире Java данным протоколом общения выступает Avro. Apache Avro позволяет определять протокол Java-объекта, который будет передаваться в очередь Kafka. Таким образом можно обмениваться целыми Java-объектами, что делает работу проще и понятнее. Kafka отлично подойдёт для архитектур где важно хранить состояние очереди, иметь возможность управлять стратегией получения сообщения и передачи их большего количества между сервисами.

В ходе данного исследования были выявлены основные преимущества асинхронных архитектур, а именно: обработка пользовательских запросов в фоне, горизонтальное масштабирование системы, большие возможности интеграции со сторонними системами. Микросервисные асинхронные архитектуры позволяют исполнить процессы, где важно эффективно выполнить задачу, но при этом не ожидать ответ, пока тот или иной подпроцесс выполняется. Важное преимущество такой архитектуры – это подключение к расширению или созданию новой функциональности другими разработчиками, так как большие задачи, как правило, реализуются с большими затратами человеческих ресурсов. Модули системы слабосвязанные и имеют протоколы связи, которые безболезненно могут быть заменены. Были исследованы возможные взаимодействия в распределённых системах.

ЛИТЕРАТУРА

1. Синхронные архитектуры. Основы Spring Cloud. / Создание микросервисов Сэм Ньюмен [Электронный ресурс]. – Режим доступа: https://vk.com/doc7608079_449808456?hash=330da3096bb49d5549 – Дата доступа: 29.08.2020.
2. Асинхронные архитектуры. Инфраструктурные сервисы архитектуры. Spring Cloud Config. Spring Cloud Foundry. Взаимодействие компонентов в микросервисных архитектурах. / Java в облаке Джош Лонг, Кеннет Бастанни [Электронный ресурс]. – Режим доступа: https://vk.com/doc26879026 473369384?hash=191b74be6a25cb6661 – Дата доступа: 13.06.2020.
3. Основные понятия и быстрый старт с Kafka / Документация Kafka [Электронный ресурс]. – Режим доступа: https://kafka.apache.org/documentation/#ecosystem – Дата доступа: 14.06.2020.
4. Spring Cloud Netflix Microservices / Статья Kirill Sereda [Электронный ресурс]. – Режим доступа: https://goo-gl.su/yQczFctA – Дата доступа: 15.06.2020.