DOI: 10.58168/MoInSyTe2024_429-436
УДК 004.9

ОБЗОР ТЕХНОЛОГИЙ ПЕРЕДАЧИ ИНФОРМАЦИИ МЕЖДУ МИКРО-СЕРВИСАМИ

А.А. Соловьев[1], В.И. Анциферова[1], В.К. Зольников[1], А.Н. Щеблыкин[1]

[1]ФГБОУ ВО «Воронежский государственный лесотехнический университет имени Г.Ф. Морозова»

Аннотация. В работе рассматриваются принципы работы технологий передачи данных, их плюсы и минусы, проблема выбора технологии передачи данных между сервисами, трудности использования и сложности поддержки каждой технологии самых популярных и эффективных технологиях: REST, Kafka, Grpc.

Ключевые слова: Kafka, RESTAPI, GraphQ, HTTP, метод, микро-сервис, партиция, офсет консюмер, топик.

OVERVIEW OF INFORMATION TRANSFER TECHNOLOGIES BETWEEN MICRO-SERVICES

A.A. Solovyev[1], V.I. Antsiferova[1], V.K. Zolnikov[1], A.N. Shcheblykin[1]

[1]Voronezh State University of Forestry and Technologies named after G.F. Morozov

Abstract. The paper discusses the principles of data transfer technologies, their pros and cons, the problem of choosing a data transfer technology between services, the difficulties of using and the difficulties of supporting each technology in the most popular and effective technologies: REST, Kafka, Grpc.

Keywords: Kafka, REST API, Graph, HTTP, method, micro-service, partition, offset consumer, topic.

В современном мире нам часто приходится сталкиваться с различными приложениями, о существовании сервисов которых даже не задумываемся, хоть ипользуемся ими каждый день.

© Соловьев А. А, Анциферова В. И., Зольников В. К., Щеблыкин А. Н., 2024

Каждый раз открывая приложение банка, чтобы вновь проверить баланс карты, оплатить ЖКХ или перевести деньги на необходимые нужды, мы не задумываемся о том, что происходит внутри этой иконки приложения банка. Так же, когда пользователь отправляет сообщение в мессенджере, ставит лайк или переводит деньги маме, он не задумывается, где хранятся эти сообщения, как осуществляется доставка данных на различные сервисы системы, он лишь хочется выполнить эти действия. Причем пользователю нужно видеть достоверную информацию: виден ли лайк всем его друзьям, успешен ли денежный перевод маме.

Все это обеспечивается приложением и сервисами внутри него. Как правило, банковское приложение имеет несколько сервисов: сервис-фронтэнд, отвечающий за общение с пользователем или попросту ui; сервис-бэкэнд, отвечающий за сохранение и обработку данных, переданных с ui. При ближайшем рассмотрении этой архитектуры можно увидеть, что бэкенд - сервис состоит из маленьких сервисов помощников, которые декомпозируют сложную логику. Это тренд называется микро сервисной архитектурой. На смену большим ресурсоемким монолитным программам пришли небольшие сервисы, облегчающие работу всей системе. И если в монолите все ясно, то как устроена связь между этими сервисами?В этом разработчикам помогают такие технологии как: REST, Kafka/RabittMQ, WebSoket,GraphQl, gRPC, SOAP и другие.

В данной статье я хочу осветить проблему выбора технологии передачи данных между сервисами, трудности использования и сложности поддержки каждой технологии, а также хочу поделиться опытом применения данных технологий на практике, как учебных, так и коммерческих данных. Работа будет сосредоточена на нескольких самых популярных и эффективных технологиях: REST, Kafka, Grpc.

Общение сервисов по протоколу HTTP(REST API)

Множество приложений используют данный протокол для общения между сервисами по причине его простоты. HTTP – широко распространённыйсетевой протокол прикладного уровня, изначально предназначенный для передачи гипертекстовых документов (то есть документов, содержащие ссылки, позволяющие организовать переход к другим документам). Аббревиатура HTTP расшифровывается как HyperTextTransferProtocol – «протокол передачи гипертекста».

HTTP-запрос состоит из четырёх элементов: метод, URI, версия HTTP и адрес хоста. Данные элементы обладают своими особенностями. Так, метод указывает, на то какое действие нужно совершить,а URI — это полный путь до конкретного файла на севере.

HTTP-ответ имеет три части: статус ответа, заголовки и тело ответа. В статусе ответа сообщается, всё ли прошло успешно или возникли ошибки. В заголовках указывается дополнительная информация, помогающая браузеру корректно отобразить файл. А в тело ответа сервер кладёт запрашиваемый файл.

На основе данного протокола был создан REST –архитектурный стиль разработки интерфейсов, позволяющий прозрачно и легко настроить взаимодействие между сервисами. RESTоперирует ресурсами, а это в свою очередьвсе, что разработчик хочет показать внешнему миру через приложение. Обычно разработчики сами определяют, какие ресурсы они хотят открыть для использования, но открыть ресурсы мало, их надо точно идентифицировать. Данный процесс можно выполнить при помощи уникальных urlи методов http.

Также стоит упомянуть про такие методы как:

GET — получение информации об объекте (ресурсе).

POST — создание нового объекта (ресурса).

PUT — полная замена объекта (ресурса) на обновленную версию.

PATCH — частичное изменение объекта (ресурса).

DELETE — удаление информации об объекте (ресурсе).

Стоит осветить, из чего состоит REST-запрос на сервер.

Как говорилось ранее,RESTоперирует ресурсами, но также их можно назвать конечными точками(endpoint) – адрес, по которому отправляется запрос.Один и тот же объект (ресурс) может иметь несколько конечных точек.

Например, чтобы отправить запрос на сервер, может использоваться конечная точка service/orders. Чтобы посмотреть список данных, можно использовать конечную точку service/orders/list, а чтобы создать новый объект — service /orders/create.

Параметры также можно передавать через url, к примеру, service/bills/{userId}/list, здесь userId уникальный идентификатор пользователя, для которого клиент запросил данные о счетах.

Также не стоит забывать о том, что можно передавать заголовки запроса, и, конечно, само тело. Формат тела запроса будет рассмотрен ниже.

После получения данного запроса, сервер его обработает и отправит ответ.

Сам ответ состоит также из заголовков, по большей части из тех, что уже были в запросе. Тело ответа –это информация, которую запрашивал клиент. Обычно для запроса и ответа используется JSON формат, так как его удобно читать и поддерживать, но RESTподдерживает и другие типы данных. Также в структуру ответа входит httpстатус код, который описывает состояние ответа.

Ответы вида 1xx — информационные.

Ответы вида 2xx говорят об успешном выполнении запроса. Например:

200 – OK. Если клиентом были запрошены какие-либо данные, то они находятся в заголовке или теле сообщения.

201 – OK. Создан новый ресурс.

Ответы вида 3xx обозначают перенаправление или необходимость уточнения. Например:

300 — на отправленный запрос есть несколько вариантов ответа. Чтобы получить нужный вариант, клиент должен уточнить запрос.

301 — запрашиваемый адрес перемещен.

307 — запрашиваемый адрес временно перемещен.

Ответы вида 4xx говорят о том, что при выполнении запроса возникла ошибка, и это ошибка на стороне клиента. Например:

400 – BadRequest. Запрос некорректный.

401 – Unauthorized. Запрос требует аутентификации пользователя.

403 – Forbidden. Доступ к сервису запрещен.

404 – Notfound. Ресурс не найден.

Ответы вида 5xx говорят об ошибке на стороне сервера. Например:

503 — сервис недоступен.

504 — таймаут (превышено допустимое время обработки запроса).

Все это дает гибкий способ описания apiдля работы сервисов друг с другом. RESTapiбыстро реализовывается и легко поддерживается. HoREST не единственный стиль написания api основанный на http.

GraphQL — это язык запросов и серверная среда для API с открытым исходным кодом. Он появился в Facebook в 2012 году и был разработан для упрощения управления конечными точками для API на основе REST. Мы не будем углубляться в принципы работы данной технологии, лишь продемонстрируем разницу между REST и GraphQL.

Главное отличие GraphQL от REST API состоит в том, что все данные клиент может получит всего одним запросом, даже если они будут располагаться в

разных источниках. А в REST API для этого придётся сделать выборку и извлекать их уже оттуда. То есть клиент может запросить по одной конечной точке любой интересующий его объект, что в REST APIневозможно. Для одного endpoin в REST существует только один тип объекта – это может быть счет, пользователь или даже массив счетов.

На первый взгляд это кажется очень удобным: клиент может отправить любой запрос, и получить нужные ему объекты, не вызывая сервис несколько раз. К примеру, сервис А хочет получить сущности пользователя, но без определенных полей, а также все счета, связанные с этим пользователем от сервиса В. При помощи GraphQL это можно сделать за один запрос, но используя REST так сделать не получится. Для REST придется осуществить два запроса, а после удалить поля из сущности пользователя. Кажется, что ответ очевиден – следует использовать GraphQL, но если посмотреть эту технологию со стороны сервера, то можно заметить сложности.

Первая проблема – это распознавание запроса. Что запросил клиент от сервера?Возможно, он запросил сущности пользователей, самих пользователей и их счета на оплату, а может быть только типы покупок. GraphQL предоставляет типы данных, которые лежат и обрабатываются на сервере, благодаря которым и строится ответ сервера. Но если посмотреть ближе, то мы увидим, что сервер превращается в SQL сервер. Если вернуться к началу, мы осознаем, что GraphQL это и есть язык запросов, а отсюда вытекает множество сложностей. Например, организация вывода данных, оптимизация работы с базой данных, так как при REST запросе мы можем заранее на этапе разработки понять, какая будет сложность у данного запроса, следует ли нам распараллелить выполнение запросов, надо ли пустить программу в асинхронное выполнение. При работе с GraphQL разработчику это придется делать в runtime.

Вторая проблема касается не только сервера, но и клиента. GraphQL всегда присылает httpстатус равным 200, что говорит об успешном выполнении. Но если на сервере упала ошибка, то все равно вернётся код 200, и разработчикам клиента всегда надо будет смотреть на тело ответа, чтобы понять, что им пришло, тогда как при REST, разработчик по статусу ответа сразу может понять, что произошло с запросом.

Когда же использовать GraphQL и REST? GraphQL следует использовать при написании программ, для которых количество запросов будет минимальным, так, за один запрос вы получите все интересующие клиент данные.В случае

если разработчики пишут внешнийapi, то следует использовать данную технологию. В других же случаях стоит рассматривать REST.

Технологии, использующие протокол http, хорошо себя показали за почти тридцатилетнее использование в разработке. Это быстро реализуемый, легко поддерживаемый и масштабируемый договор связи между двумя сервисами. Они не требую никаких дополнительных ресурсов от системы. Но у них есть один серьёзный недостаток: запрос может выполнится только один раз, а значит, если сервер, на который будет послан запрос недоступен, то запрос просто не попадет на него. Поэтому разработчики придумывают различные алгоритмы опроса сервера, добавляют распределители запросов, например, haproxy или заменяют общение по http на общение через брокеры сообщений.

Брокеры сообщений.

Если рассматривать рынок брокеров сообщений, то фаворитом выступает Kafka. Данная технология выступает и как броккер, и как хранилище сообщений.Kafkaлегко встраивается в систему, имеет множество библиотек для работы на разных языках программирования.Также стоит отметить её легковесность и возможность задать множество настроек, подходящих для вашей системы.

Давайте рассмотрим, что такое брокеры сообщений и зачем они нужны. Брокер сообщений обычно имеет 2 сущности потребителя и производителя. Потребитель читает данные, а производитель производит эти данные для потребителя. Сущности общаются между собой при помощи шины или какой-либо очереди, где хранятся сообщения. Потребитель читает голову очереди, а производитель пишет в её хвост. Данная очередь хранит сообщения на сервере, пока потребитель их не прочитает, далее брокер придвигает голову очереди вперед, тем самым предоставляя новые данные для потребителя. Смысл очередей прост: один сервис в неё пишет, другой – читает. Тем самым разработчик устраняет проблему http и недоступности сервисов. Программист уверен, что его сообщение всегда будет доставлено на сервис, так как брокер гарантирует, пока сообщение не прочитано, оно будет хранится в очереди.

Но как потребитель читает из очереди и как производитель пишет в эту самую очередь? Все также по tcp протоколу, все также через интернет. В этом случае возникает вопрос: если брокер будет недоступен, то как мы сможем отправлять данные? Возникает та же проблема доступа, что и с технологиями HTTP.

Действительно, полностью данную проблему решать нельзя, но можно себя обезопасить, с целью чего и используют Kafka. Kafkaимеет ещё один плюс:

все её очереди могут реплицироваться, тем самым при отказе одного сервера, можно будет обратиться к другому.

В данной статье я лишь частично коснусь реализации этой технологии, мы сосредоточимся на том, как Kafkaрешает проблемы передачи сообщений.

Kafka оперирует не очередями, а топиками. При этом в каждом топике может быть x партиций, а в каждой партиции будет n оффсетов – сообщений. Производитель пишет в топик сообщение, далее Kafkaопределяет, в какую партицию будет записано сообщение и с каким оффсетом. Если настроен механизм репликации, то в дальнейшем это сообщение будет передано другим репликам данной партиции. После того как все реплики обновят данные (но стоит заметить, что количество реплик, которых ждет продюсер тоже настраиваемое количество), производитель получит уведомление о сохранении сообщения. Конечно, стоит сказать, что процесс репликации достаточно сложен. Существуют так называемые ISR реплики, которые не отстают или отстают на настроенное количество сообщений от главной партиции. При отказе основной партиции сначала будут опрошены ISR реплики, а только потом остальные. Также стоит отметить, что партиции одного топика могут иметь лидирующие(главные) реплики на разных серверах.

Вернемся к сообщению. После того как продюсер его записал, консюмер может его считать. Консюмер обращается к Kafka, передает ей название топик и партиции, а также номер последнего прочтенного сообщения. Далее Kafkaсама прочитает сообщение, удалит его из партиции, передвинет оставшиеся сообщения и отдаст сообщение консюмеру. Если во время чтения потребитель будет недоступен, то Kafkaоткатит изменения, если же у потребителя настроено реплицирование, то Kafkaсама передаст чтение другой реплике потребителя. Стоит отметить, что данный механизм достаточно надежен, однако не убережёт, если у вас один сервер к Kafka.

В данной статье я попытался раскрыть технологии передачи данных между сервисами. Как мы смогли убедиться, нет универсального подхода, нужно комбинировать технологии на основе http и, например, брокеры сообщений. Так же не стоит забывать об обработке ответов и отправке запросов на стороне сервисов. Порой хватит алгоритма повторения запроса на сервер, с периодическим засыпанием программы. Также возможно создать общение между сервисами при помощи базы данных, когда один сервис считывает данные из таблиц, не нагружая запросами основной сервис.

Выбор коммуникации сервисов лежит на разработчиках и архитекторах. Тщательно взвешивая все стороны приложения, можно прийти к верному решению.

Список литературы

1. Гавриленко, М.Н. Автоматизация получения и обработки большого объема данных с использованием технологий Java и ApacheKafka / М.Н. Гавриленко ; науч. рук. Е.И. Сукач // Актуальные вопросы физики и техники : IX Республиканская научная конференция студентов, магистрантов и аспирантов (Гомель, 23 апреля 2020 г.) : материалы : в 2 ч. / М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины; редкол. : Д. Л. Коваленко (гл. ред.) [и др.]. – Гомель : ГГУ им. Ф. Скорины, 2020. – Ч. 1. – С. 270-272.

2. Нархид Н. ApacheKafka. Потоковая обработка и анализ данных / Н. Нархид, Г. Шапира, Т. Палино. – Санкт-Петербург : Питер, 2019. – 320 с.

3. Широкополосные беспроводные сети передачи информации / В.М. Вишневский [и др.]. - Москва: Техносфера, 2005. - 595 с.

4. Полуэктов А.В., Макаренко Ф.В., Ягодкин А.С. Использование сторонних библиотек при написании программ для обработки статистических данных // Моделирование систем и процессов. – 2022. – Т. 15, № 2. – С. 33-41.

References

1. Gavrilenko M.N. Automation of receiving and processing large amounts of data using Java and ApacheKafka technologies // Current issues in physics and technology : IX Republican scientific conference of students, magisters and postgraduates (Gomel, April 23, 2020) : proceedings : in 2 parts. – Gomel: GSU named after F. Skorina, 2020. – Part 1. – P. 270-272.

2. Narhid N. ApacheKafka. Stream processing and data analysis / N. Narhid, G. Shapira, T. Palino. – St. Petersburg : Piter, 2019. – 320 p.

3. Broadband wireless information transmission networks / V.M. Vishnevskiy et al. – Moscow: Tekhnosfera, 2005. – 595 p.

4. Poluektov A.V., Makarenko F.V., Yagodkin A.S. The use of third-party libraries when writing programs for processing statistical data // Modeling of systems and processes. - 2022. – Vol. 15, No. 2. – pp. 33-41.