ИКОНОМИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

КАТЕДРА „ИНФОРМАТИКА“

РЕФЕРАТ

по дисциплината

„Интернет технологии и комуникации“

на тема:

Облачни комуникационни модели в дистрибутирана система за управление на поръчки

Докторант:
Йордан Йорданов

Научен ръководител:
доц. д-р Павел Петров

Варна, 2022

Съдържание

Списък на съкращенията

REST	Representational State Transfer
GRPC	Google Remote Procedure Call
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
CNCF	Cloud Native Computing Foundation
WWW	World Wide Web

Въведение

Комуникационните модели са важен актив за много широкомащабни уеб приложения, в частност системи за електронна търговия или управление на поръчки/доставки. Възможностите на дистрибутираните системи да работят с огромен брой потребители едновременно са съществени.

Актуалността на изследваната тема се обуславя от тенденцията облачните технологии да се превръщат в основна платформа на продуктовите предприятията, които искат да внедряват и изпълняват своите дигитални бизнес услуги.

Обект на изследване в настоящия труд е информационна система базирана на архитектура, състояща се от много малки, независими микроуслуги. Всяка микроуслуга се изпълнява в отделен процес като контейнер, който е разположен в клъстер, управляван от инструмент за оркестрация, който отговаря за внедряване и управлението.

Целта на реферата е да представи сътрудничеството или т.н. информационно и икономическо взаимодействие между подсистемите, изискващо надеждни модели за комуникация. От архитектурна гледна точка, комуникационните протоколи са важно дизайнерско решение. В и извън клъстера, микро-услугите комуникират чрез технологии за изпращане и получаване на данни.

Основните точки, които са поставени за изпълнение на целта, са следните:

- да се изследват възможните комуникационни модели между клиентско приложение и микро-услугите от бекенда;
- да се предложат основни принципи и добри практики при изграждането на комуникация в приложения, базирани на облак;

Основната теза на изследването е, че едно от главните предизвикателства е да се внедрят бизнес процеси от край до край, като същевременно се поддържа последователност и съгласуваност в микроуслугите. Главна причина са възникнето на частични повреди в отделните компоненти на системата.

1. Синхронна комуникация между микроуслуги

Може да разграничим два основни вида комуникация, които са използвани между различните микро-услуги: синхронна и асинхронна комуникация. Синхронната се основава главно на протоколите HTTP/HTTPS и се характеризира със модела „заявка-отговор“.

1.1 Механизъм за комуникиране чрез трансфер на репрезентативно състояние

Прехвърляне на представително състояние (REST) е архитектурен подход за проектиране на уеб услуги, който обхваща основите на световна мрежа (WWW), която е съществена част от съвременното общество и икономика.

Представен е през 2000г. като част от дисертацията на Рой Т. Филдинг. REST е стил за изграждане на разпределени системи, базирани на хипермедия. Той е независим от протоколите на приложния слой. Въпреки това, най-често срещаните реализации на REST API използват HTTP като протокол на приложението. Ориентирана към ресурсите архитектура определя набор от ограничения:

- Клиент-сървър е стилът клиент-сървър е мотивиран от разделяне на притесненията. Той насърчава разделянето на потребителски интерфейси и съхранение на данни на системата. Той също така

опростява специален компонент дизайн и позволява независима еволюция на клиентски и сървърни компоненти.

- Безгражданство Безгражданството насърчава мащабируемостта и надеждността. Освен това улеснява разработването от страна на сървъра.
- Слоеста система Използването на многослойна система позволява набор от интересни архитектурни разширения, като например като наследено капсулиране, балансиране на натоварването и (споделено) кеширане.
- Униформен интерфейс е Единният интерфейс е най-важното ограничение на стила REST и определя ресурсите като централни единици, с които може да се оперира с помощта на фиксиран набор от глаголи (методите HTTP).

Основно предимство на REST е, че той използва отворени стандарти и не обвързва внедряването на API или клиентските приложения с конкретна реализация. Например, REST услугите в системата за управление са написани на ASP.NET, докато клиентските приложения използват React и JavaScript, които могат да генерират HTTP заявки и да десериализират отговори. Основните принципи при проектирането на RESTful API, използващи HTTP са:

- REST API са проектирани като ресурси до които клиентът има достъп.
- Ресурсът има идентификатор (URI), който уникално го идентифицира. Например, URI за конкретна клиентска поръчка може да бъде: *https://manager.com/orders/eu.123123.231*
- Клиентите взаимодействат с услугите, като обменят обекти от данни. Системата за управление на поръчките използва JSON като формат за обмен. Например, GET заявка към посочения по-горе

URI ще върне отговор: {"orderId":eu.123123.231,"orderValue":99.90,"productId":1}

- Приложните интерфейси, изградени върху HTTP, използват глаголи за извършване на операции с ресурси. Най-често срещаните операции са GET, POST, PUT, PATCH и DELETE.
  Table 2.1: A
- REST API използват модел на заявка без проследяване на състоянието. HTTP заявките трябва да са независими, запазването на информация за преходно състояние между заявките не е осъществимо. Единственото място, където се съхранява информацията, е в самите ресурси и всяка заявка трябва да бъде атомна операция. Това ограничение позволява уеб услугите да бъдат силно мащабируеми, тъй като не е необходимо да се запазва афинитет между клиенти и конкретни сървъри. Всеки сървър може да обработи всяка заявка от всеки клиент. Въпреки това други фактори могат да ограничат мащабируемостта. Например, много уеб услуги пишат в хранилище за данни, което може да е трудно за мащабиране. За повече информация относно стратегиите за мащабиране на хранилище за данни вижте Хоризонтално, вертикално и функционално разделяне на данни.
- REST API се управляват от хипермедийни връзки, които се съдържат в представянето. Например, следното показва JSON представяне на поръчка. Той съдържа връзки за получаване или

актуализиране на клиента, свързан с поръчката.

```
{
   "orderID":3,
   "productID":2,
   "quantity":4,
   "orderValue":16.60,
   "links": [
      {"rel":"product","href":"https://mang.com/customers/3",
"action":"GET" },
      {"rel":"product","href":"https://mang.com/customers/3",
"action":"PUT" }
   ]
}
```

- През 2008 г. Леонард Ричардсън предлага следния модел на развитие на уеб API:
- Ниво 0: Дефинирайте един URI и всички операции са POST заявки към този URI.
- Ниво 1: Създайте отделни URI за отделни ресурси.
- Ниво 2: Използвайте HTTP методи за дефиниране на операции с ресурси.
- Ниво 3: Използвайте хипермедия (HATEOAS, описано по-долу).

Ниво 3 съответства на наистина RESTful API според дефиницията на Филдинг. На практика много публикувани уеб API са някъде около ниво 2.

1.2 Механизъм за заявки към отдалечени процедури

gRPC е модерна, високопроизводителна рамка, която развива дистанционно извикване на процедури (RPC) протокол. На ниво приложение, gRPC рационализира съобщенията между клиенти и бек-енд услуги. Произхождащ от Google, gRPC е с отворен код и е част от Cloud Native Computing Foundation екосистема от облачни предложения.

Типичното клиентско приложение на gRPC ще разкрие локална функция да извиква друга функция на отдалечена машина като процес, който реализира бизнес операция. RPC абстрахира мрежовата комуникация и изпълнение от точка до точка.

gRPC е модерна, високопроизводителна рамка, която развива вековното дистанционно извикване на процедури (RPC) протокол. На ниво приложение, gRPC рационализира съобщенията между клиенти и бек-енд услуги. Произхождащ от Google, gRPC е с отворен код и е част от Cloud Native Computing Foundation(CNCF) екосистема от облачни предложения. CNCF счита gRPC за инкубационен проект. Инкубиране означава, че крайните потребители използват технологията в производствените приложения и проектът е здравословен брой сътрудници.

Типичното клиентско приложение на gRPC ще разкрие локална функция в процес, която реализира бизнес операция. Под завивките тази локална функция извиква друга функция на отдалечена машина. Какво изглежда, че локалното повикване по същество се превръща в прозрачно повикване извън процеса към отдалечена услуга. RPC водопроводът абстрахира мрежовата комуникация от точка до точка, сериализацията и изпълнение между компютри.

В приложенията, базирани на облак, разработчиците често работят на различни езици за програмиране, рамки и технологии. Тази оперативна съвместимост усложнява договорите за съобщения и необходимия

водопровод междуплатформена комуникация. gRPC осигурява „равномерен хоризонтален слой“, който ги абстрахира опасения. Разработчиците кодират в родната си платформа, фокусирани върху бизнес функционалността, докато gRPC се занимава с комуникационни водопроводи.

gRPC използва HTTP/2 за своя транспортен протокол. Въпреки че е съвместим с HTTP 1.1, HTTP/2 разполага с много разширени възможности:

- Двоичен протокол за кадриране за транспортиране на данни - за разлика от HTTP 1.1, който е базиран на текст.
- Поддръжка за мултиплексиране за изпращане на множество паралелни заявки през една и съща връзка - HTTP 1.1 ограничава обработката до едно съобщение за заявка/отговор в даден момент.

gRPC обхваща технология с отворен код, наречена Protocol Buffers. Те осигуряват висока ефективност и платформено-неутрален формат за сериализиране за сериализиране на структурирани съобщения, до които услугите изпращат взаимно. Използвайки междуплатформен език за дефиниране на интерфейс (IDL), разработчиците дефинират услуга договор за всяка микроуслуга. Договорът, реализиран като текстов .proto файл, описва методи, входове и изходи за всяка услуга. Същият файл на договора може да се използва за клиенти на gRPC и услуги, изградени на различни платформи за разработка.

Използвайки прото файла, компилаторът Protobuf, protoc, генерира както клиентски, така и служебен код за вашия целева платформа. Кодът включва следните компоненти:

• Строго въведени обекти, споделени от клиента и услугата, които представляват операциите на услугата и елементи от данни за съобщение.

• Силно въведен базов клас с необходимия мрежов водопровод, който отдалечената услуга gRPC може да наследява и разширява.

• Клиентска заглушка, която съдържа необходимия водопровод за извикване на отдалечената услуга gRPC.

По време на изпълнение всяко съобщение се сериализира като стандартно представяне на Protobuf и се обменя между клиента и отдалечената услуга. За разлика от JSON или XML, съобщенията на Protobuf се сериализират като компилирани двоични байтове.

Книгата gRPC за разработчици на WCF, достъпна от сайта на Microsoft Architecture, предоставя задълбочена информация покритие на gRPC и буфери на протоколи

Сравнете gRPC услугите с HTTP API

Jlkjlkjlk

1.3 Недостатъци на синхронната комуникация между микроуслуги

Всички услуги, осъществяващи синхронна комуникация имат много знания една за друга. Създава се тясна връзка между различните микроуслуги, което нарушава една от предпоставките за използване на микросервизи. Всеки път, когато бъде добавена нова услуга и тя трябва да бъде актуализирана за нещо, което се случва в системата, ще трябва да се направят промени в кода, за да бъде извикана и тази нова услуга. Така добавянето на нови услуги става все по-трудно. С течение на времето системата може да бъде натоварена на места, които не сме забелязали в началото.

Недостатъците на използването на синхронна комуникация ще бъде малък мост, към следващата точка, където ще разгледаме асинхронната комуникация.

2. Асинхронна комуникация между микроуслуги

Кратко въведение в асинхронната комуникация за микроуслуги, различните опции за използване. Протокол за комуникация е AMQP. Може да има един или множество приемници на съобщенията.

2.1 Въведение в проблема "Съгласуваност между услугите"

Предизвикателството е да се внедрят бизнес процеси от край до край, като същевременно се поддържа последователност в услугите. За да представим проблема напълно, е важно да разгледаме две диаграми на монолитната или ориентираната към услуги версии на системата.

- нито една услуга не трябва да включва таблици/хранилище от друга и никога не трябва да извиква директни заявки към тях
- комуникация, базирана на събития. модел за публикуване-абониране
- Предизвикателството относно комуникацията не е толкова в протоколите, а повече за стила, защото когато възникне повреда – колкото по-свързана е системата, толкова по-големи проблеми ще се получат
- Частични повреди, проектирате на системата, като се вземе предвид общите рискове

2.2 Асинхронна комуникация между различните микроуслуги с помощта на посредник на съобщения

Най-често срещаният подход е, използване на посредник за корпоративни съобщения с опашки и теми за публикуване-абониране.

Сервизната шина се използва за отделяне на приложения и услуги един от друг, осигурявайки следните предимства:

- Работа за балансиране на натоварването между конкуриращи се работници
- Безопасно маршрутизиране и прехвърляне на данни и контрол през границите на услуги и приложения
- Координиране на транзакционна работа, която изисква висока степен на надеждност.

3. Комуникационни модели за достъп до бекенда

С въвеждането на потребителски интерфейс трябва да представим как клиентските приложения взаимодействат с различните услуги. Ще проучим как предния край на системата достъпва до инфраструктурата на микроуслугите.

3.1 Директна комуникация на клиент с микроуслуга

Използва се, когато различни части от страницата на клиента изискват различни микроуслуги. Обикновено клиентът извиква балансьор на натоварване, който изисква данни от вътрешната микросервизна инфраструктура.

Някои от недостатъците на този подход:

- Прекалено много „обиколки“ в Интернет (извън вътрешната микросервизна мрежа)
- Микроуслугите трябва да бъдат изложени на „външния свят“
- Междусекторни проблеми като удостоверяване и оторизация
- Използване на синхронна комуникация като HTTP
- Различните клиентски приложения изискват различни API (уеб срещу мобилни клиенти)

3.2 Шлюз за приложете програмни интерфейси

Предоставя еднократна крайна точка за група микроуслуги. Наподобява модела за дизайн: „фасадата“. Известен е също като „backend for frontend“. Изгражда се за конкретните нужди на клиента, Действа като пълномощник между клиентите и микроуслугите. Може да осигури удостоверяване, кеширане и други проблеми.

API шлюзът може да се превърне в “анти-модел“ като пълно монолитно приложение: съдържащо твърде много крайни точки, обединяващо всички микроуслуги, унищожавайки техните предимства.

API шлюзовете също трябва да бъдат отделени за всеки клиент, разделен от логически групи въз основа на бизнес граници. протокол за пренос на данни могат да бъдат HTTP или gRPC

3.3 Комуникация в реално време

Комуникацията в реално време може да се постигне с HTTP уеб сокети, изградени с резервни механизми. Използва се, когато изпращаме данни от услугите директно към клиентите

ASP.NET Core има SignalR като комуникационна технология в реално време. SignalR е библиотека за ASP.NET разработчици, която опростява процеса на добавяне на уеб функционалност в реално време към приложенията. Уеб функционалността в реално време е възможността сървърният код да изпраща съдържание към свързани клиенти незабавно, когато стане достъпно, вместо сървърът да чака клиент да поиска нови данни. SignalR може да се използва за добавяне на всякакъв вид уеб функционалност "в реално време". Всеки път, когато потребител обновява уеб страница, за да види нови данни, то, тя е кандидат за използване на SignalR.

Заключение

Комуникацията на услугите се превръща в важно дизайнерско решение при конструирането на облачно приложение. В реферата проучвахме модели на комуникация, които са естествени за облачната инфраструктура: клиентите от предния край комуникират с микроуслуги от задния край, платформи за API шлюз и комуникация в реално време. Разгледахме как комуникират микроуслугите с други бек-енд услуги, както синхронната HTTP комуникация, така и асинхронни съобщения между услугите. Покрихме gRPC, модерна, високопроизводителна рамка, която развива дистанционно извикване на процедури.

Използвана литература

1. БИЖКОВ, Г., КРАЕВСКИ, В. (2007) *Методология и методи на педагогическите изследвания*. УИ „Св. Климент Охридски“.
2. ВАНКОВА, Д. (2014) *Делфи – методът, същност и изследователски опит*. МУ – Варна. Известия на съюза на учените – Варна. с. 59-66.
3. ГАВРАИЛОВ, Е. (2014) *Основи на научните изследвания.* УИ ВСУ „Черноризец Храбър“.
4. ГАНЧЕВ, Г., ДЕЛЧЕВ, М. (2013) *Методика на педагогическите изследвания*. [Онлайн] Достъпно на: https://obuch.info/metodika-na-pedagogicheskite-izsledvaniya.html [Достъпено: 20 декември 2021].
5. ДИМИТРОВ, Н. (2013) *Въведение в научните изследвания*. „Интелексперт-94“.
6. ИИКТ – БАН. (2020) *Ефективни методи и алгоритми за Монте Карло симулации, анализ на чувствителността и стохастични оптимизации.* [Онлайн] Достъпно на: http://ict.acad.bg/?page_id=557 [Достъпено: 22 януари 2022].
7. КОРОВКИНА, Н., ЛЕВОЧКИНА, Г. (2022) *Методика подготовки исследовательских работ студентов*. [Онлайн] Достъпно на: https://intuit.ru/studies/courses/11980/1160/info [Достъпено: 15 декември 2021].
8. НЕМИНСКА, Р. (2015) *Методи на интердисциплинарно обучение.* Българско списание за образование. Брой 2. с.115-125.
9. ОРЛОЕВ, Н. (2002) *Методология на научните изследвания*. РУ „А. Кънчев”.
10. ПАПАНЧЕВ, Т. (2015) *Обзор на методите за моделиране на надеждността на електронни изделия.* Сп. „Компютърни науки и комуникации”. БСУ – Бургас. Том 4. с. 34-43.