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 2](#_Toc99271352)

[Въведение 3](#_Toc99271353)

[1. Синхронна комуникация 4](#_Toc99271354)

[1.1 Механизъм за трансфер на репрезентативно състояние 4](#_Toc99271355)

[1.2 Механизъм за заявки към отдалечени процедури 8](#_Toc99271356)

[1.3 Сравнение на двата стила за реализация 9](#_Toc99271357)

[2. Асинхронна комуникация 10](#_Toc99271358)

[2.1 Въведение в проблема “Съгласуваност между услугите” 10](#_Toc99271359)

[2.2 Асинхронна комуникация между различните микроуслуги с помощта на посредник на съобщения 11](#_Toc99271360)

[3. Комуникационни модели за достъп до бекенда 11](#_Toc99271361)

[3.1 Директна комуникация на клиент с микроуслуга 11](#_Toc99271362)

[3.2 Шлюз за приложете програмни интерфейси 13](#_Toc99271363)

[4. Препоръки при проектиране 14](#_Toc99271364)

[Заключение 15](#_Toc99271365)

[Използвана литература 16](#_Toc99271366)

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 | |
| JSON | JavaScript Object Notation | |

Въведение

Комуникационните технологии са важен актив за много широкомащабни уеб приложения, в частност системи за електронна търговия или управление на поръчки. Те са част от световната мрежа, която сама по себе си представлява разпределена система от взаимосвързани ресурси.

**Актуалността на изследваната тема** се обуславя от тенденцията облачните технологии да се превръщат в инструменти за стратегическа трансформация и дигитализация на бизнеса. Облачните платформи позволяват мигновено пускане на иновативните идеи на пазара. Тези предимствата поставят компаниите една стъпка пред конкурентите.

**Обект на изследване** в настоящия труд е разпределена информационна система, базиранa на микроуслуги, работеща върху множество процеси и сървъри (хостове). Всяка услуга се изпълнява в отделен процес като контейнер, разположен в клъстер от виртуални машини. Екземплярите трябва да взаимодействат помежду си с помощта на протоколи за комуникация като HTTP, AMQP, TCP в зависимост от естеството на работа. Инфраструктурата е разгърната в облачната платформа Azure и се управлява от инструмент за оркестрация Kubernetes.

**Целта** на реферата е да представи информационното взаимодействие между подсистемите за управление на поръчките към клиенти, които комуникират чрез технологии за изпращане и получаване на данни.

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

-  да се разгледат актуалните комуникационни модели, които интегрират отделните части на софтуерният продукт;

-  да се предложат основни принципи и добри практики при изграждането на облачно базираните приложения;

**Основната теза** на изследването е свързана с предизвикателства при внедряването на бизнес процеси „от край до край“, като същевременно се поддържа последователност и съгласуваност в компонентите на системата. Архитектурата трябва да изпълни важни нефункционални изисквания като: висока степен на достъпност[[1]](#footnote-1), разширяване на мащаба[[2]](#footnote-2), при увеличаващ се потребителски трафик.

Клиент и услугите могат да използват различни видове комуникация, всеки един насочване към постигането на различни цели. Може да разграничим два основни вида комуникация, които се използват между компонентите на системата: синхронна и асинхронна комуникация.

# Синхронна комуникация

Уеб услугите са интерфейси, които са предназначени за комуникация между машини, за разлика от уеб сайтовете, които са насочени към взаимодействието с хората и се достъпват чрез браузър. Уеб услугите могат да обслужват различни видове клиенти, като например мобилни, базирани в потребителския браузър или други сървърни услуги.

В синхрония подход клиентът изпраща заявка към услуга, която я обработва и изпраща обратно отговор. Клиентският код може да продължи своята задача само след получаване на отговора.

## Механизъм за трансфер на репрезентативно състояние

Прехвърляне на представително състояние (REST) е архитектурен подход за проектиране на уеб услуги, който обхваща основите на световна мрежа.

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, като концептуалните идеи зад него са взети от HTTP и се основават на WWW.

Основно предимство на REST е, че той използва отворени стандарти и не обвързва внедряването на API или клиентските приложения с конкретна реализация.

REST API са проектирани около ресурси, които могат да бъдат обекти, данни или услуги, които могат да бъдат достъпни от клиента. Всеки ресурсът има уникален идентификатор (URI). Например, URI за конкретна клиентска поръчка: [*https://manager.com/orders/eu.123123.231*](https://manager.com/orders/eu.123123.231)*.* Уеб API използват JSON като формат за обмен. Той се фокусира единствено върху сбитото представяне на структурирани данни. Например, заявка към посочения по-горе URI ще върне отговор:

{

"orderId":eu.123123.231,

"orderValue":99.90,

"productId":1

}

Има различни HTTP методи, които осигуряват различна семантика, когато се прилагат към ресурс, както е показано в следната таблица.

|  |  |
| --- | --- |
| Метод | Описание |
| GET |  |
| HEAD |  |
| PUT |  |
| DELETE |  |
| POST |  |
| OPTIONS |  |

* 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.

## Механизъм за заявки към отдалечени процедури

gRPC е модерна, високопроизводителна рамка, която развива дистанционно извикване на процедури (RPC). 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.

Kлиентско gRPC приложение разкрива локална функция в уеб услуга, която реализира бизнес операция. Тази локална функция извиква друга функция на отдалечена машина.

В приложенията, базирани на облак, разработчиците често работят на различни езици за програмиране, рамки и технологии. gRPC осигурява   
„хоризонтален слой“, който абстрахира опасенията от несъвместимост.

gRPC използва HTTP/2 като транспортен протокол (съвместим също и с HTTP 1.1), който 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

## Сравнение на двата стила за реализация

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.

# Асинхронна комуникация

[NET-Microservices-Architecture-for-Containerized-NET-Applications.pdf](file:///C:\books\NET-Microservices-Architecture-for-Containerized-NET-Applications.pdf)

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.

## Въведение в проблема “Съгласуваност между услугите”

* 102 6 Concurrent and Scalable Storage Backends
* Distributed transactions

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

## 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.

# 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.

## Директна комуникация на клиент с микроуслуга

облачна система, предните клиенти (мобилни, уеб и настолни приложения) изискват комуникационен канал за взаимодействие с независими бек-енд микроуслуги. За да бъде нещата опростени, един клиент от предния край може директно да комуникира с микроуслугите на задния край.



С този подход всяка микроуслуга има публична крайна точка, която е достъпна от предни клиенти. В производствена среда, бихте поставили балансьор на натоварването пред микроуслугите, маршрутизиращи трафика пропорционално.

Макар и лесна за изпълнение, директната комуникация с клиента би била приемлива само за прости микросервизни приложения. Този модел тясно свързва клиентите от предния край с основните бек-енд услуги, отваряне на вратата за редица проблеми, включително:

* Податливостта на клиентите към рефакторинг на услуги в задния край;
* По-широка повърхност за атака, тъй като основните бек-енд услуги са директно изложени;
* Дублиране на междусекторни проблеми във всяка микроуслуга;
* Прекалено сложен клиентски код - клиентите трябва да следят множество крайни точки и да се справят с неуспехите устойчив начин.

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)

## 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

# Препоръки при проектиране

……….

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, E. (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.

1. висока степен на достъпност - компонент на технологична система, която елиминира единични точки на повреда, за да осигури непрекъснати операции или време на работа за продължителен период. Тя гарантира, че бази данни и приложения работят, когато е необходимо. [↑](#footnote-ref-1)
2. мащабируемост - способността на система, мрежа или процес да поддържа увеличаващ се обем на работа. [↑](#footnote-ref-2)