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 | |
| IDL | Interface Definition Language | |
| SOA | Service-oriented architecture | |

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 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, насочени към постигането на различни цели. Може да разграничим два основни типа, 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 HTTP[[3]](#footnote-3) заявка към услуга, която я обработва и изпраща обратно HTTP отговор. Клиентският код може да продължи своята задача само след получаване на отговора. HTTP заявките и отговорите имат обща структура: съдържат набор от заглавни редове, които включват информация за заявката, отговора и субекта.

## 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/бизнес обектите. Например, в системата за управление на поръчки основните субекти са потребители и поръчки. Създаването на поръчка може да се постигне чрез изпращане на HTTP POST заявка, която съдържа определена информацията. HTTP отговорът показва дали поръчката е направена успешно или не. Ресурсът не винаги се основава на един елемент от физически данни. Например, ресурсът за поръчка се внедрява вътрешно от няколко таблици в релационна база данни, но представен като едно цяло. REST е стил за моделиране на обекти и операциите, които приложението изпълнява върху тях. Обектите често се групират в колекции (поръчки, клиенти). Колекцията е отделен ресурс и свой собствен URI. Добра практика е да организираме URI в йерархия като например: [*https://manager.com/orders*](https://manager.com/orders/) връща колекцията от поръчки. Всеки елемент в колекцията има свой собствен уникален URI, като този за конкретна клиентска поръчка може да бъде [*https://manager.com/orders/eu.123123.231*](https://manager.com/orders/eu.123123.231)*.* Представянето на връзките между различните видове ресурси, като например доставки за поръчка: [*https://manager.com/orders/eu.123123.231/deliveries*](https://manager.com/orders/eu.123123.231/deliveries). Важно да отбележим, че това ниво на сложност може да бъде трудно за поддържане ако връзките между ресурсите се променят в бъдеще.

HTTP протоколът дефинира редица методи, показани в таблица 1.1, които осигуряват различна семантика, когато се прилагат към ресурс:

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

**Таблица 1.1**: Таблица на официалните методи на HTTP 1.1.

Ефектът от конкретна заявка зависи от това дали ресурсът е колекция или отделен елемент. Следващата таблица обобщава общите конвенции, приети от повечето RESTful реализации, използвайки примера за управление на поръчки.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ресурс | GET | POST | PUT | DELETE |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Таблица 1.2**: Таблица на общите REST конвенции.

Както бе споменато по-рано, клиентите и сървърите обменят представяния на ресурси. Например в POST, тялото на заявката съдържа представяне на ресурса за създаване. В GET заявка тялото на отговора съдържа представяне на извлечения ресурс.

В HTTP протокола форматите се определят чрез използване на типове медии, наричани още MIME. За недвоични данни повечето уеб API поддържат JSON (тип на медия = application/json) или XML (тип на носител = application/xml) като формат за обмен. Те се фокусира единствено върху сбитото представяне на структурирани данни. Например, заявка към посочения по-горе URI за детайли на поръчки, ще върне отговор:

{

"orderId":eu.123123.231,

"orderValue":99.90,

"productId":1

}

Сървърът информира клиента за резултата от заявка чрез използване на предварително зададени кодове на състоянието. Класовете кодове на състоянието могат да се видят в таблица 2.2.

|  |  |  |  |
| --- | --- | --- | --- |
| Статус код | Тип | Описание | Пример |
| 1xx |  |  |  |
| 2xx |  |  |  |
| 3xx |  |  |  |
| 4xx |  |  |  |
| 5xx |  |  |  |

**Таблица 1.3**: Таблица с диапазоните на кодовете на HTTP

REST е гъвкав архитектурен стил, който дефинира CRUD-базирани операции. Клиентите взаимодействат със сървъра през HTTP с модел на заявка/отговор. Той е широко разпространен, но следва да разгледаме по-нова комуникационна технология, набираща скорост в общността на облачните технологии: gRPC.

## 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, това е проект с отворен код и 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, който разполага с много разширени възможности:

* Двоичен протокол за транспортиране на данни, за разлика от HTTP1.1, който е текстов.
* Позволяващо заявки и отговори за асинхронно предаване на голям набор от данни (масиви)

gRPC е базирана на технология с отворен код, наречена Protocol Buffers. Файловете .**proto** осигуряват висока ефективност и платформено-неутрален формат за структуриране на съобщения. Използвайки междуплатформен език за дефиниране на интерфейс (IDL), разработчиците дефинират “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 може да генерира както клиентски, така и сървърен код за целевата платформа. Кодът включва следните компоненти:

• Строго обособени обекти, споделени от клиента и услугата, които представляват елементи от съобщението;

• Базов клас, който отдалечената gRPC услуга може да наследява;

Може да разгледаме следния пример за **order\_delivery.proto**.

syntax = "proto3"; // версия на синтаксиса

option csharp\_namespace = "Manager.Protos";

package order\_delivery; // идентификатор на пакета

import "google/protobuf/wrappers.proto";

service OrdersDeliveries {

rpc GetOrder (GetOrderRequest) returns (NullableOrder); // метод за извикване

}

message GetOrderRequest { // формат на съобщението

google.protobuf.StringValue order\_nr = 1;

}

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.

Технологията ни предоставя няколко типа:

* Unary RPC - The client sends a request message to the server and receives a response;
* Server streaming RPC - The client sends a request message to the server and receives a sequence of responses;
* Client streaming RPC - The client sends a sequence of messages and receives a single response from the server;
* Bidirectional streaming RPC The client and the server exchange messages in both directions;

## Сравнение на двата стила за реализация

Следната таблица сравнява различните архитектурни видове:

|  |  |  |
| --- | --- | --- |
|  | REST | gRPC |
|  |  |  |
| формат |  |  |
| случай на |  |  |
| употреба |  |  |
|  |  |  |

**Таблица 1.4**: Таблица

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

Важен момент при изграждането на микроуслуги е интеграцията помежду им. Комуникацията между вътрешните микроуслуги трябва да е сведена до минимум. Целта на всяка микроуслуга е да бъде автономна и достъпна за потребителя, дори ако другите услуги, които са част от приложението не работят. В случай, когато трябва да се извърши повикване от една микроуслуга към други (като изпълнение на HTTP заявка за данни), за да може да се предостави отговор, имате архитектура, която няма да бъде устойчива, когато някоя от частите се срине. Освен това, когато се създадът вериги от заявки/отговори, производителността на микроуслугите се намалява

## Асинхронна комуникация, базирана на съобщения

Асинхронните съобщения и управляваната от събития комуникация са от критично значение при разпространението на промените в множество микроуслуги и свързаните с тях модели на домейни. Когато настъпят промени, системата се нуждае от начин за съгласуване им в различните модели. Решението е евентуална последователност и комуникация, управлявана от събития, базирана на асинхронни съобщения.

Клиент прави заявка към услуга, като й изпраща съобщение. Тъй като това е асинхронна комуникация, клиентът приема, че отговорът няма да бъде получен веднага или че няма никакъв отговор. Съобщението се състои от заглавие (метаданни като информация за идентификация) и тяло. Съобщенията се изпращат чрез асинхронни протоколи като AMQP. Част от предпочитаната инфраструктура за този тип е брокер на съобщения, който е различен от тези, използвани в SOA. В „олекотен“ вариант, той действа като шина.

Налични са различни брокери на съобщения като всеки един има низ за връзка и потребителски достъп. Реалната част на процеса се състои в крайните точки, които произвеждат и консумират съобщения, тоест в микроуслугите. Важно правило, което системата за управление се опитва да спазва, доколкото е възможно, е да използва само асинхронни съобщения между вътрешните услуги, а синхронна комуникация: само от клиентските приложения към API Gateway.

Когато се използва асинхронна комуникация, управлявана от събития, микроуслуга публикува интеграционно събитие, когато нещо се случи в нейния домейн и друга микроуслуга трябва да е наясно с това. Пример за това е промяна на цената в микроуслуга от продуктов каталог. Микроуслугите се абонират за събитията, за да могат да ги получават асинхронно. Когато това се случи, получателите може да актуализират собствените си обекти. Тази система за публикуване/абониране се изпълнява чрез използване на реализация на шина за събития.

## 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

# 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

…**Tip**

Avoid requiring resource URIs more complex than collection/item/collection.

…….

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. Robert Vettor (2021) *Architecting Cloud-Native .NET Apps for Azure*. Microsoft. Redmond, Washington.
2. Steve Smith, (2021) *Architecting Modern Web Applications with ASP.NET Core and Azure*. Microsoft. Redmond, Washington.
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 непрекъснатiи операции или време на работа за продължителен период. [↑](#footnote-ref-1)
2. мащабируемост - способността на система, мрежа или процес да поддържа увеличаващ се обем на работа. [↑](#footnote-ref-2)
3. Протоколът за прехвърляне на хипертекст (HTTP) е мрежов протокол, от приложния слой на OSI модела, за пренос на информация. Използва TCP/IP . [↑](#footnote-ref-3)