ДОДАТОК А

Слайди презентації

Атестаційна робота магістра

Дослідження засобів і технологій міжсервісного зв’язку в мікросервісній архітектурі програмних систем

Виконав:

ст. гр. ІПЗм-17-2 Синкевич М.Е.

Керівник проекту: проф. Лесна Н.С.

Мета роботи. Об’єкт і предмет дослідження

Метою роботи є виявлення найбільш ефективних засобів і технологій міжсервісного зв’язку в мікросервісній архітектурі програмних систем.

Об’єктом дослідження є процес міжсервісного зв’язку в мікросервісній архітектурі програмних систем.

Пердметом дослідження є засоби і технології міжсервісного зв’язку в мікросервісній архітектурі програмних систем.

2

Актуальність теми. Наукова новизна

Актуальність теми атестаційної роботи обумовлена наявністю активної комунікації між сервісами у програмних системах з мікросервісною архітектурою, яка накладає часові затримки на обробку запитів при транспортуванні та обробці повідомлень, що спричиняє пряму залежність між продуктивністю системи та ефективністю технології міжсервісного зв'язку.

Наукова новизна. Запропонована методика оцінювання ефективності технологій міжсервісного зв'язку в мікросервісній архітектурі програмних систем.

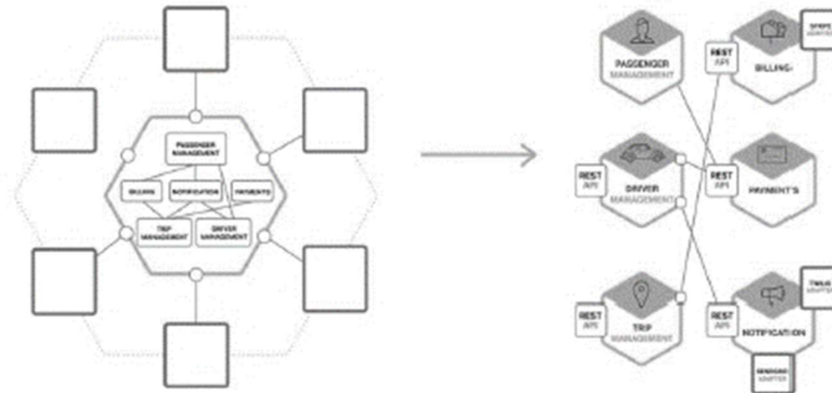
3

Постановка задачі

- провести аналіз проблем, що існують при міжсервісній комунікації та визначити причини їх виникнення;
- дослідити існуючі технології та протоколи передачі інформації;
- дослідити існуючі засоби серіалізації та десеріалізації даних;
- розробити методику оцінювання ефективності;
- виявити критерії оцінювання ефективності;
- розробити програмну систему для дослідження продуктивності обраних технологій;
- виявити найбільш ефективні технології міжсервісного зв'язку та засоби серіалізації даних у відповідності до обраних критеріїв;
- сформулювати рекомендації щодо впровадження REST та GraphQL з JSON у якості формату серіалізації та десеріалізації.

4

Причини виникнення проблем комунікації в мікросервісній архітектурі програмних систем



Формат сериалізації даних

Розмір	Найбільший	Проміжний	Проміжний	Найменший
Формат	XML	JSON	Thrift Binary	Protocol Buffers

Час	Найдовший	Проміжний	Проміжний	Найшвидший
Формат	XML	JSON	Protocol Buffers Thrift Binary	Protocol Buffers Thrift Binary

Критерії оцінювання ефективності

- підтримка мультиплатформності;
- незалежність від формату даних;
- стандартизація;
- когнітивна зрозумілість;
- простота використання;
- розповсюдженість;
- версіонування;
- кешування.

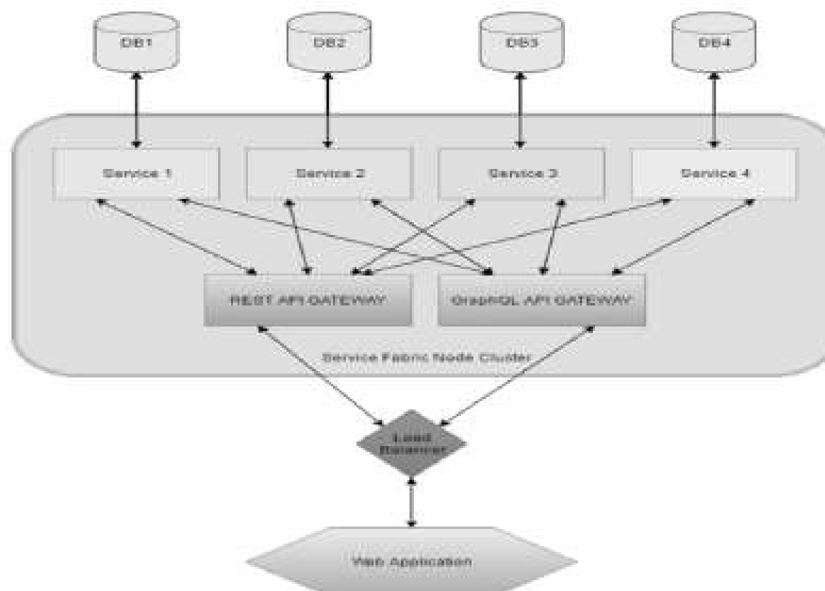
7

Засоби комунікації

- **REST** — підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь»;
- **GraphQL** — мова запитів для інтерфейсів прикладного програмування, що моделює дані у вигляді графів JSON-об'єктів;
- **SOAP** — це багатоплатформовий протокол зв'язку, що використовує XML для інкапсуляції повідомлень і для визначення, як їх передавати та отримувати, є розширенням XML-RPC;
- **Apache Thrift** — це мова опису інтерфейсів, що використовується для визначення і створення сервісів під різні мови програмування. Являє собою фреймворк до RPC;
- **gRPC** — є інструментом для крос-платформних віддалених викликів процедур, також як і Thrift. У gRPC Protocol Buffers використовується як формат серіалізації для повідомлення, а також як мова визначення інтерфейсу.

8

Структура програмної системи дослідження продуктивності технологій міжсервісного зв'язку в мікросервісній архітектурі



9

Обрані засоби:

- мова програмування Kotlin;
- середовище розробки IntelliJ IDEA;
- СКБД MariaDB;
- Google Cloud Platform.

Обрані технології

- Spring Framework;
- Spring boot;
- Hibernate;
- Axios;
- Kubernetes.

10

Структура таблиць, які містять тестові дані

Animals	
Type (char (6))	Name (char(6))

Persons	
ID (char (6))	Name (char(6))

Car	
Model (char (6))	Color (char(6))

Buildings	
Address(char (6))	Department (char(6))

11

Методика оцінювання ефективності засобів і технологій міжсервісного зв'язку

- здійснити вибір критеріїв оцінювання ефективності засобів і технологій міжсервісного зв'язку;
- виконати моделювання продуктивності для обраних за критеріями технологій;
- розробити рекомендації щодо використання розглянутих технологій міжсервісного зв'язку;
- здійснити вибір технології, спираючись на значення найбільш вагомих для системи критеріїв.

12

Задовільнення технологій обраним критеріям

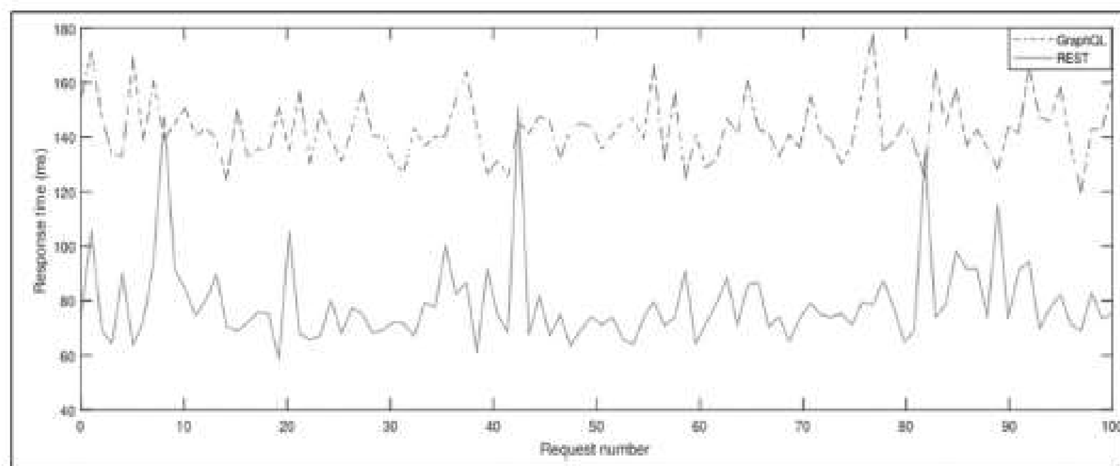
	REST	GraphQL	SOAP	Apache Thrift	gRPC
Мультиплатформність	+	+	+	—	—
Незалежність від формату даних	+	+	—	+	—
Стандартизація	—	—	+	—	—
Когнітивна зрозумілість	+	+	—	+	+
Простота використання	+	—	—	—	+
Розповсюдженість	+	+	—	—	+
Версіонування	+	—	+	+	—
Кешування	+	—	+	—	—

13

Результати моделювання

Перший сценарій

Час відповіді при запиті даних за допомогою GraphQL та REST інтерфейсів

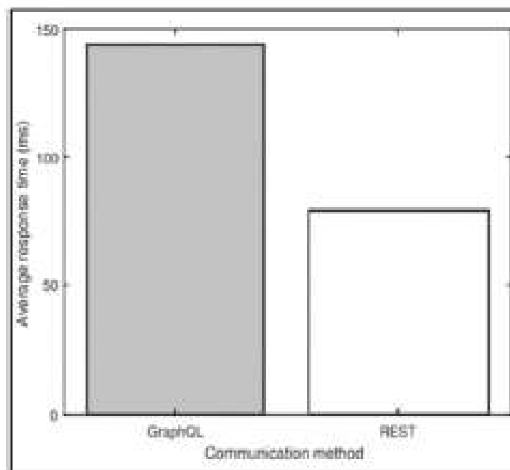


14

Результати моделювання

Перший сценарій

Порівняння середнього часу відгуку для GraphQL та REST

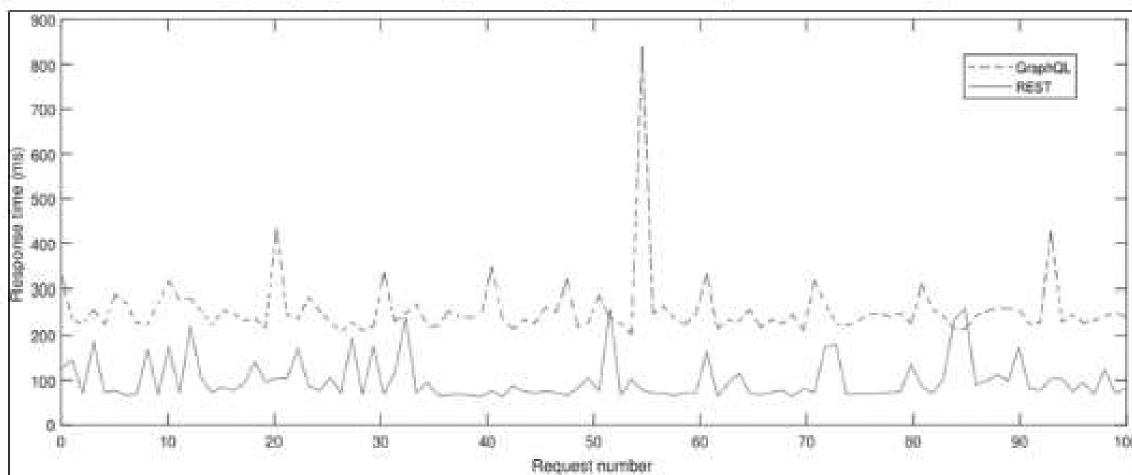


15

Результати моделювання

Другий сценарій

Час відповіді при запиті даних за допомого GraphQL та REST інтерфейсів

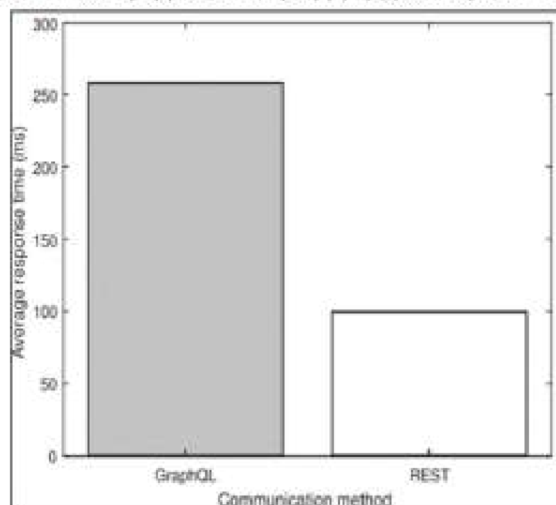


16

Результати моделювання

Другий сценарій

Порівняння середнього часу відгуку для GraphQL та REST

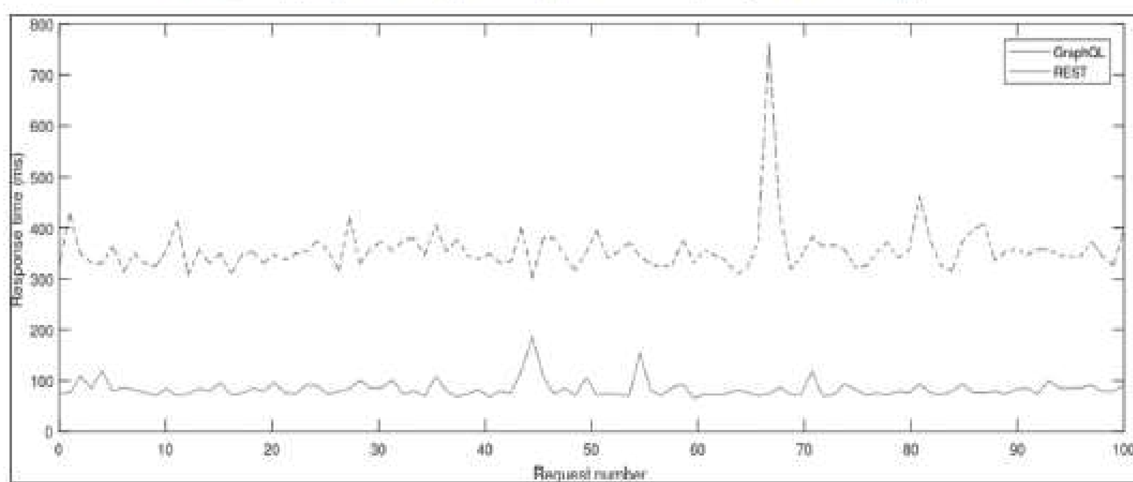


17

Результати моделювання

Третій сценарій

Час відповіді при запиті даних за допомого GraphQL та REST інтерфейсів

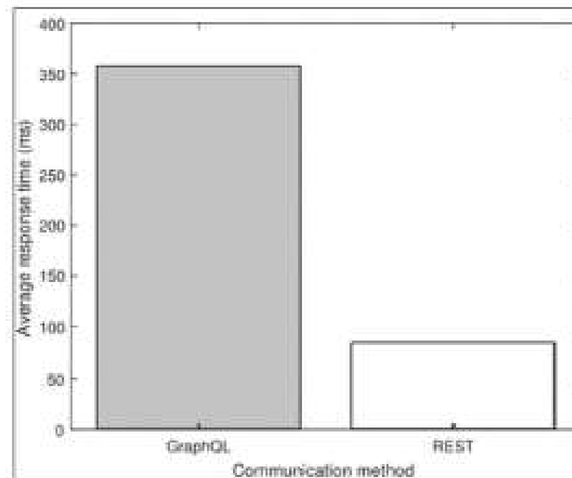


18

Результати моделювання

Третій сценарій

Порівняння середнього часу відгуку для GraphQL та REST



19

Рекомендації по використанню технологій міжсервісного зв'язку

- якщо однією з вимог до системи є жорстка стандартизація, використання XML прийнятне, а продуктивність не є першочерговою вимогою до системи, то SOAP буде найліпшим кандидатом, так як SOAP пропонує структурований спосіб відправки повідомлень між службами зі строгими правилами, що визначають структуру повідомлень і способи їх обробки;
- якщо продуктивність все ж є важливою вимогою, то SOAP не повинен бути обраним, так як він прив'язаний до використання XML, який, безумовно, має найгіршу продуктивність із усіх розглянутих форматів серіалізації. Високу продуктивність пропонують RPC технології, до яких відносяться Apache Thrift та gRPC;
- якщо важлива зрозумілість використання та мультиплатформовість технології то REST може бути використаний для встановлення міжсервісного зв'язку;
- GraphQL пропонує абстрактне уявлення про ресурси даних як вузли в графі, де будь-які вузли можуть бути об'єднані для формування запиту.

20

Апробація роботи

- міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті», опубліковано тези доповіді;
- стаття «Дослідження засобів і технологій міжсервісного зв’язку в мікросервісній архітектурі» у міжнародному науковому електроному журналі «Наука онлайн».

Висновки

- Розглянуті існуючі технології міжсервісної комунікації та засоби серіалізації і десеріалізації даних, виявлені їх недоліками та перевагами.
- Розроблена методика оцінювання ефективності засобів і технологій міжсервісного зв’язку, сформульовані критерії оцінювання ефективності засобів комунікації та їх вплив на вибір засобу.
- Розроблена програмна система, що дозволяє провести моделювання продуктивності обраних технологій міжсервісної комунікації.
- За допомогою розробленої програмної системи проведено дослідження продуктивності наступних засобів міжсервісної комунікації: REST та GraphQL з використанням JSON якості формату серіалізації та десеріалізації даних.
- Базуючись на результатах проведеного дослідження, виявлено найбільш ефективні технології міжсервісного зв’язку.
- Розроблено рекомендації щодо впровадження технологій міжсервісного зв’язку з урахуванням ваги кожного з обраних критеріїв.

22

ДОДАТОК Б

Лістинг коду програми

PersonApp.kt

```
@EnableSwagger2
@EnableFeignClients
@EnableCircuitBreaker
@EnableDiscoveryClient
@SpringBootApplication
class PersonApp

fun main(args: Array<String>) {
    runApplication<PersonApp>(*args)
}
```

PersonController.kt

```
@RestController
class PersonController(
    private val personService: PersonService,
    private val personConverter: PersonToDtoConverter
) {

    @GetMapping("/person")
    fun list() = personService.all()
        .map(personConverter)
}
```

PersonClient.kt

```
@FeignClient("person-service")
interface PersonClient {

    @GetMapping("/person")
    fun payment(): PersonDto

    data class PersonDto(
        val id: Int,
        val name: String
    )
}
```

.gitlab-ci.yml

```
image: alpine:latest

variables:
KUBERNETES_VERSION: 1.10.9
HELM_VERSION: 2.11.0
```

```
DOCKER_DRIVER: overlay2

stages:
- build
- package
- production

build_jar:
stage: build
image: openjdk:8-alpine
script: "chmod a+x gradlew && ./gradlew build"
artifacts:
paths:
- build/libs/*.jar
  only:
    - branches

build_image:
  stage: package
  image: docker:stable
  services:
    - docker:stable-dind
  script:
    - build_image
  only:
    - branches

production:
  stage: production
  script:
    - deploy
  environment:
    name: production
    url: http://$CI_PROJECT_PATH_SLUG.$AUTO_DEVOPS_DOMAIN
  only:
    refs:
      - master
  except:
    variables:
      - $KUBECONFIG == null
```

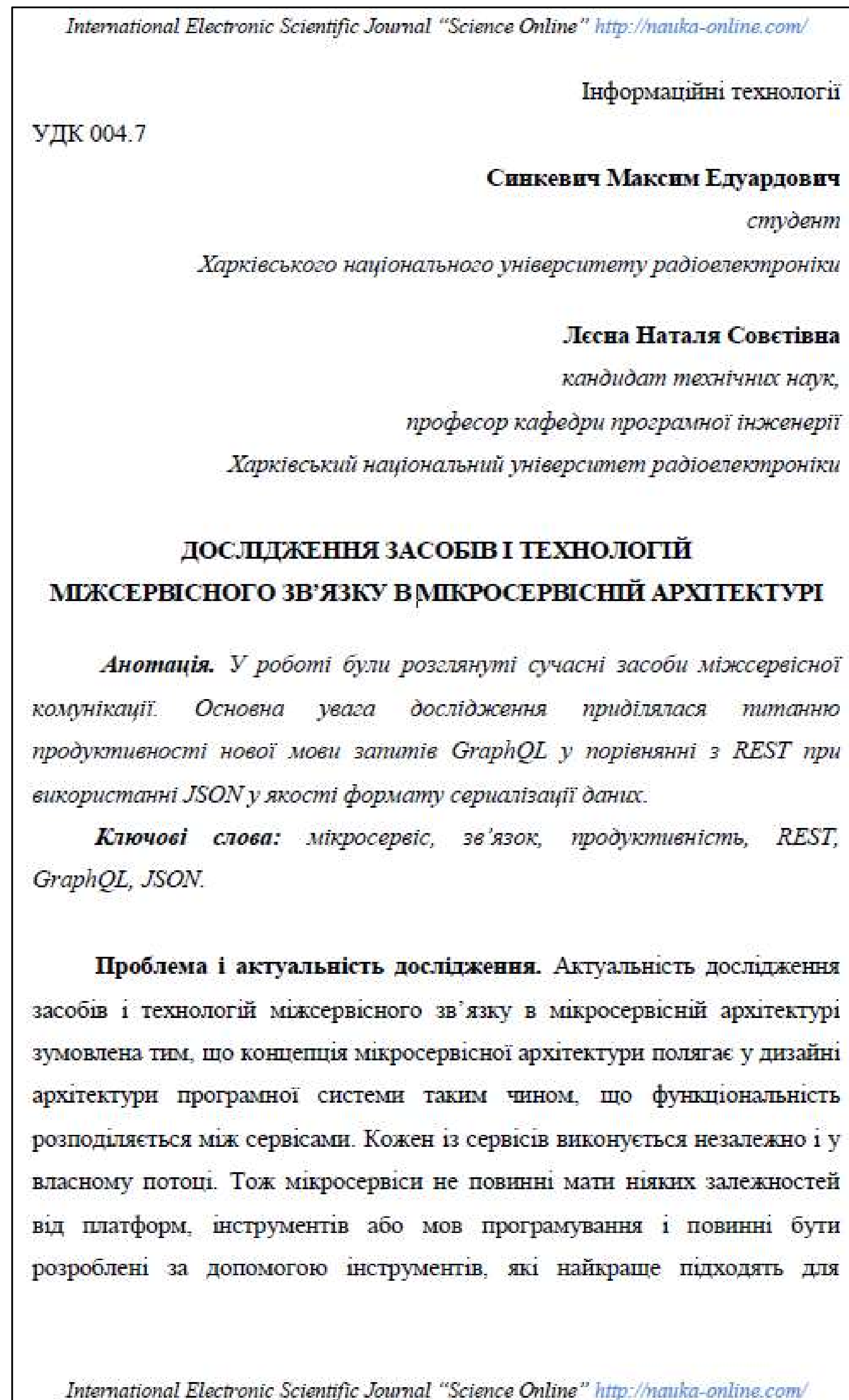
Dockerfile

```
FROM openjdk:8-jre-alpine
VOLUME /tmp
COPY build/libs/* app.jar
ENTRYPOINT ["java”,"-jar","/app.jar"]
```

ДОДАТОК В

Наукові публікації

В.1 Стаття у міжнародному науковому журналі Наука Онлайн



International Electronic Scientific Journal "Science Online" http://nauka-online.com/

Інформаційні технології

УДК 004.7

Синкевич Максим Едуардович

студент

Харківського національного університету радіоелектроніки

Лєсна Наталя Совєтівна

кандидат технічних наук,

професор кафедри програмної інженерії

Харківський національний університет радіоелектроніки

ДОСЛІДЖЕННЯ ЗАСОБІВ І ТЕХНОЛОГІЙ МІЖСЕРВІСНОГО ЗВ'ЯЗКУ В МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

Анотація. *У роботі були розглянуті сучасні засоби міжсервісної комунікації. Основна увага дослідження приділялася питанню продуктивності нової мови запитів GraphQL у порівнянні з REST при використанні JSON у якості формату серіалізації даних.*

Ключові слова: *мікросервіс, зв'язок, продуктивність, REST, GraphQL, JSON.*

Проблема і актуальність дослідження. Актуальність дослідження засобів і технологій міжсервісного зв'язку в мікросервісній архітектурі зумовлена тим, що концепція мікросервісної архітектури полягає у дизайні архітектури програмної системи таким чином, що функціональність розподіляється між сервісами. Кожен із сервісів виконується незалежно і у власному потоці. Тож мікросервіси не повинні мати ніяких залежностей від платформ, інструментів або мов програмування і повинні бути розроблені за допомогою інструментів, які найкраще підходять для

International Electronic Scientific Journal "Science Online" http://nauka-online.com/

Рисунок В.1 — Сторінка 1 статті

конкретної цілі. Найбільш імовірно, що кожен з сервісів використовується декількома різними клієнтами, серед яких інші мікросервіси системи, що розроблені і працюють на різних платформах. Тому важливо використовувати комунікаційні протоколи та засоби, що можуть ефективно використовуватися як мобільним додатком чи web застосунком, так і компонентами системи.

Огляд поточного стану об'єкта дослідження. У мікросервісній архітектурі сервіси взаємодіють між собою за допомогою передачі повідомлень мережею. Загальноприйнятими методами реалізації такої взаємодії є передача повідомлень за допомогою викликів REST API або моделі підписник-видавець, де сервіси можуть виконувати підписку на канал та отримувати нотифікацію про те, що нове повідомлення було опубліковано до каналу [1]. Проте існує декілька розповсюджених альтернатив, таких як SOAP, gRPC, Thrift, GraphQL та інші. Зв'язок між сервісами багаторазово досліджувався.

Ефективність обміну повідомленнями в розподілених системах є важливою темою, яка висвітлюється у багатьох роботах. Наприклад, Тихоміровс та Гребіс представили дослідження продуктивності веб сервісів при використанні REST та SOAP, у якому REST проявив себе краще [2]. Немає чіткої відповіді на питання, який метод комунікації найкращий для мікросервісів. Хоча фреймворки gRPC і Thrift потенційно пропонують найкращу продуктивність, вони додають залежності до архітектури мікросервісів, змушуючи її розвиватися на певних мовах; список мов, що підтримуються цими фреймворками, обмежений. REST, який є повністю незалежним від платформи, оскільки він є лише архітектурним стилем, може бути реалізований на будь-якій платформі, з будь-якою мовою, використовуючи будь-який транспорт і будь-яку серіалізацію [3]. Проте REST не має структури, яку пропонують фреймворки і протоколи, і часто слід використовувати інструменти третіх

Рисунок В.2 — Сторінка 2 статті

сторін для відстеження його інтерфейсів. SOAP пропонує структурований спосіб відправки повідомлень між службами зі строгими правилами, що визначають структуру повідомлень і способи їх обробки. Основний недолік SOAP полягає в тому, що він прив'язаний до використання XML, який, безумовно, має найгіршу продуктивність із всіх перелічених форматів серіалізації. GraphQL пропонує новий спосіб запиту даних у форматі, який легко зрозуміти. GraphQL пропонує абстрактне уявлення про ресурси даних як вузли у графі, де будь-які вузли можуть бути об'єднані для формування запиту [4]. Якщо фреймворк GraphQL може проаналізувати і серіалізувати дані без великих втрат продуктивності в порівнянні з архітектурою REST з використанням JSON, це може бути життєздатною альтернативою.

Мета дослідження. Дана робота містить дослідження засобів міжсервісної комунікації для визначення ефективного способу міжсервісної взаємодії. Об'єктами дослідження обрано фреймворк GraphQL та архітектурний стиль REST. У якості формату сериалізацих обрано JSON. Метою є створення прототипу мікросервісної системи, призначеної для порівняльного аналізу продуктивності міжсервісної комунікації при використанні REST та GraphQL.

Дослідження продуктивності міжсервісної комунікації при використанні REST та GraphQL. Критерієм продуктивності міжсервісної комунікації є час, який витрачається на транспортування повідомлення за обраним протоколом та технологією. Для порівняння продуктивності GraphQL та REST розроблено тестове середовище та виконано заміри часу проходження запитів через систему.

Тестове середовище містить чотири мікросервіси та сервер баз даних. Кожен з сервісів має підтримку REST та GraphQL запитів. Мікросервіси в системі призначені для виконувати тільки доступ до бази даних і не виконувати жодних фактичних обчислень. Кожний сервіс має

Рисунок В.3 — Сторінка 3 статті

доступ до даних, що доступні тільки для цього конкретного сервісу. Розглянемо тест, що складається з отримання досить невеликої кількості даних від кожного сервісу з використання інтерфейсів REST і GraphQL. Таблиці баз даних містять по 5 записів. Для отримання достовірних результатів надіслано 100 запитів один за одним з секундною затримкою між ними. Затримка впроваджується для того, щоб бути впевненим, що навантаження на систему має мінімальний вплив на результат. За час відповіді взято повний час знаходження запиту у системі, включаючи час, який займає виконання обробки запиту мікросервісом та включаючи час на витяг та обробку даних з бази даних. Час відповіді для кожного запиту до кожного із інтерфейсів зображено на рисунку 1.

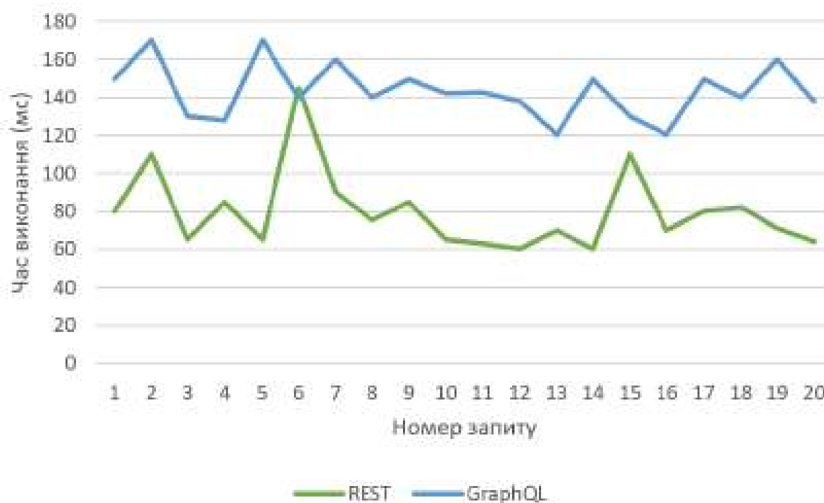


Рис. 1. Час відповіді при запиті даних за допомого GraphQL та REST інтерфейсів

Найшвидший час відповіді для REST становить 58,53 мс, а найповільніший — 151,08 мс. Взаємодія з використанням GraphQL має значно гірший результат з найшвидшим часом відгуку у 119,86 мс і найповільнішим часом відгуку у 177,72 мс. Середній час відгуку для REST дорівнює 78,94 мс, а для GraphQL — 142,92 мс. Таким чином GraphQL є повільнішим в середньому на 81,05%.

Рисунок В.4 — Сторінка 4 статті

Аналогічні тести проведені з використанням більших об'ємів даних: по 25 та 50 запитів у таблицях баз даних. У всіх випадках GraphQL показав довший час відповіді ніж REST.

Висновки. Результати тестів показують, що GraphQL не може конкурувати з архітектурою REST з використанням JSON формату з точки зору часу відгуку. Навіть при отриманні незначного обсягу даних обробка GraphQL викликає просідання часу відповіді. Так як GraphQL прив'язаний до використання JSON у якості формату повідомлень, то він обмежений у техніках серилізації. Проблема незручності REST інтерфейсу полягає у тому, що кожен ресурс відображається на URI і якщо різні програми-споживачі мають потребу у специфічній структурі даних, то для кожного такого випадку необхідно створювати новий специфічний REST ендпоінт. У противагу, при використанні GraphQL потрібно створити тільки докладну структуру, що описує наявні типи і поля. Використовуючи такий ресурс кожен клієнт-споживач може точно вказати, які поля він бажає отримати. Це виключає великі інтерфейси, які важко підтримувати і які не мають чітких правил щодо того, як обробляти декілька версій одного API. GraphQL є гарною ідеєю для використання у сценарії, коли запитується невеликий обсяг даних. Але якщо продуктивність є важливим критерієм при виборі технології, то GraphQL не є сильним претендентом.

Література

1. C. Richardson. Building microservices: Inter-process communication in a microservices architecture [Електронний ресурс]. — Режим доступу: https://www.nginx.com/blog/building-microservices-inter-process-communication (дата звернення: 23.05.2017).
2. J. Grabis J. Tihomirovs. Comparison of soap and rest based web services using software evaluation metrics. Information Technology and Management Science, 19, December 2016 — 92 с.

Рисунок В.5 — Сторінка 5 статті

3. S. Newman. Building Microservices. O'Reilly Media Inc., USA, 2015. — P. 49.
4. Graphql offcial webpage with documentation [Електронний ресурс]. — Режим доступу: http://graphql.org (дата звернення: 23.05.2017).

Рисунок В.6 — Сторінка 6 статті

В.2 Тези доповіді на міжнародному молодіжному форумі «Радіоелектроніка та молодь у XXI столітті»

ТЕХНОЛОГИИ МЕЖСЕРВИСНОЙ СВЯЗИ В МИКРОСЕРВИСНОЙ АРХИТЕКТУРЕ ПРОГРАММНЫХ СИСТЕМ

Синкевич М.Э.
Научный руководитель — проф. Лесная Н.С.
Харьковский национальный университет радиоэлектроники
(61166, Харьков, пр. Науки 14, каф. Программной инженерии,
тел.(057) 702-13-06)
e-mail: maksym.synkevych@nure.ua, тел. 095-579-43-02

The given work considers the interservice communication methods used in the microservice architecture of software systems. Since the connection between microservices should be efficient and reliable, with a large number of small services interacting to perform a single transaction, this can be a challenge, therefore, it is necessary to identify ways to achieve better efficiency. Consider the main approaches to the process organizing of interaction and identify a compromise between asynchronous and synchronous messaging which significantly affect the work of the software.

Микросервисная архитектура структурирует приложение как набор сервисов, обладающих следующими свойствами: легкие в поддержке и тестировании, слабо сцепленные, независимо разворачиваемые [1]. В классических вариантах сервисно-ориентированной архитектуры модули сами по себе могут быть достаточно сложными программными системами, а взаимодействие между ними зачастую возлагается на стандартизированные протоколы (такие, как SOAP, XML-RPC), которые обычно тяжеловесны. В микросервисной же архитектуре приложения выстраиваются из компонентов, которые выполняют относительно простые функции и взаимодействуют с использованием экономичных сетевых коммуникационных протоколов (например, в подходе REST с использованием JSON, Protocol Buffers, Thrift). За счёт повышения гранулярности модулей архитектура нацелена на увеличение связности и на уменьшение степени сцепления, что разрешает проще добавлять и изменять функции в системе. Несмотря на преимущества архитектурного стиля он не лишен недостатков. Среди основных проблем при внедрении можно отметить: если в модулях, выполняющих несколько функций, взаимодействие локально, то микросервисная архитектура накладывает требование атомизации модулей и взаимодействия их в сети; отсутствие стандартизации и необходимость согласования форматов обмена между сервисами; баланс нагрузки и отказоустойчивость. Для них существуют решения, используя которые можно пойти на те или иные уступки. Существует два основных шаблона обмена сообщениями, которые микросервисы могут использовать для связи с другими микросервисами. Синхронное общение — сервис вызывает API, предоставляемый другим сервисом, используя протокол, такой как HTTP или gRPC. Этот вариант является синхронной схемой обмена сообщениями, потому что

151

Рисунок В.7 — Сторінка 1 тез

вызывающая сторона ожидает ответа от получателя. Асинхронное общение — сервис отправляет сообщение без ожидания ответа, и один или несколько сервисов обрабатывают сообщение асинхронно.

Важно различать асинхронный ввод-вывод и асинхронный протокол. Первый означает, что вызывающий поток не блокируется, второй — что отправитель не дожидается ответа от получателя.

Существуют компромиссы для обоих шаблонов. Запрос и ответ - это хорошо понятная парадигма, поэтому разработка API может показаться более естественной, чем разработка системы обмена сообщениями. Однако асинхронный обмен сообщениями имеет некоторые преимущества, которые могут быть полезны в микросервисной архитектуре: отправителю сообщения не нужно знать о потребителе; используя модель издатель-подписчик, множество потребителей могут подписаться на получение событий; если потребитель отказывает, отправитель по прежнему может отправлять сообщения, они будут получены, когда потребитель восстановится; чувствительность — вышестоящий сервис может ответить быстрее если он не дожидается нижестоящих вервисов; выравнивание нагрузки — очередь сообщений может выступать в качестве буфера для выравнивания рабочей назрузки. Однако существуют также некоторые проблемы с эффективным использованием асинхронного обмена сообщениями: связь с инфраструктурой обмена сообщениями — использование конкретной инфраструктуры для обмена сообщениями может вызвать тесную связь с ней, в результате чего будет сложнее при необходимести переключиться на другую; сквозная задержка может становиться высокой при заполнении очередей; при высокой пропускной способности денежные затраты на инфраструктуру для системы обмена сообщениями могут стать значительными; сложность — обработка асинхронных сообщений является нетривиальной задачей накладывающей свои ограчинения; если сообщения требуют семантики очереди, то она может стать узким местом в системе.

Принято считать, что построение микросервисной архитектуры основано на тех же принципах, что и REST сервис с JSON форматом взаимодействия. Это самый распространенный метод, но, как можно заметить, он не единственный. Комбинируя между собой подходы используя для определенных операций сихронные вызовы, для других — асинхронные можно достичь наиболее эффективного взаимодействия между сервисами.

Список источников:

1. Newman S. Building Microservices / Newman S. — O'Reilly Media, 2015 p 304.

Рисунок В.8 — Сторінка 2 тез

ДОДАТОК Г

Електронні матеріали