НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»

УДК _____

«До захисту допущено»

Завідувач кафедри
Стіренко С.Г.
(підпис) (ініціали, прізвище)

“_____” _____ 20**20** р.

Магістерська дисертація

зі спеціальності: *123. Комп'ютерна інженерія*
(код та назва напряму підготовки або спеціальності)

Спеціалізація: *123. Комп'ютерні системи та мережі*

на тему: Метод оптимізації обчислювальних ресурсів на основі спеціалізованих гібридних рішень

Виконав: студент VI курсу, групи ІВ-81 мн
Тихонов Костянтин Олександрович
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник професор, д.ф.-м.н., с.н.с. Гордієнко Ю.Г.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант нормоконтроль, д.т.н., професор Кулаков Ю.О.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали)
(підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність *123. Комп'ютерна інженерія*
(код і назва)

Спеціалізація *123. Комп'ютерні системи та мережі*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Стіренко С.Г.
(підпис) (ініціали, прізвище)

« » _____ 20***20*** р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Тихонов К. О.
(прізвище, ім'я, по батькові)

1. Тема дисертації «Метод оптимізації обчислювальних ресурсів на основі спеціалізованих гібридних рішень»

Науковий керівник дисертації професор, д.ф.-м.н., с.н.с. Гордієнко Ю.Г.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «***28***» ***10 2019*** р. № ***3770-с***

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження методи оптимізації програмних додатків та систем

4. Предмет дослідження методи та підходи до оптимізації програмних рішень за допомогою різних, гібрідних підходів

5. Перелік завдань, які потрібно розробити: огляд існуючих систем, та рішень для оптимізації програмних додатків

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	д.т.н., професор Кулаков Ю.О.		

7. Дата видачі завдання 13.02.2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Дослідження літератури та документації	20.02.2019	Викон.
2	Огляд існуючих рішень	15.03.2019	Викон.
3	Аналіз теоретичних методів	01.12.2019	Викон.
4	Проектування системи	12.01.2020	Викон.
5	Програмна реалізація системи	01.02.2020	Викон.
6	Тестування та виправлення помилок	08.02.2020	Викон.
7	Оформлення документації	17.04.2020	Викон.

Студент _____ _____

(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____

(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Метод оптимізації обчислювальних ресурсів на основі спеціалізованих гібридних рішень

студентом: Тихоновим Костянтином Олександровичем

Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 76 аркуші основного тексту, 20 ілюстрації, 10 таблиць. При підготовці використовувалася література з 18 різних джерел.

Актуальність. Великою проблемою сучасного світу є проблема розширення та оновлення додатків різної направленості. Швидкість розвитку сучасного світу перевищує людську можливості до передбачення напрямку цього розвитку. Саме для цього необхідно поліпшити процес міграції на новіші, більш швидкі рішення існуючих проблем. Частковим рішенням було використання архітектурних шаблонів. Але це тільки частині рішення для ситуацій в якій присутня сувора ієрархія. Тому сучасний підхід шукає ще більш гнучкі рішення.

Мета і завдання дослідження. Метою магістерської роботи є спрощення системного підходу до розробки програмного забезпечення та вирішення питання міграції до без серверного рішення

Для досягнення мети дослідження поставлено і вирішено такі завдання:

- дослідження структури та принципів побудови мікро-сервісних та без серверних архітектури
- Пошук найоптимальнішого набору інструментів для реалізації цих архітектур
- розробка програмної моделі гібридного без серверного додатку
- ілюстрація роботи моделі та аналіз отриманих результатів

Об’єкт дослідження – процес роботи додатків в гібридній без

серверній системі

Предмет дослідження – взаємодія програмних компонентів в системі яка базується на мікро сервісній архітектурі та використовує без серверний підхід

Методи досліджень. Для досягнення поставлених в магістерській роботі задач, використано знання з курсів - технологія хмарних обчислень, технологія розподілених обчислень, проектування комп'ютерних систем, та основи ООП.

Наукова новизна одержаних результатів роботи полягає у наступному:

- запропоновано спосіб аналізу існуючої системи та використання модульної структури додатку для оптимізації обчислень
- розроблено програмний продукт для аналізу роботи побудованої гібридної системи

Проведене дослідження дає змогу оптимізувати роботу існуючих програмних систем які базуються на мікро сервісній архітектурі та потребують додаткові разові обчислення.

Особистий внесок здобувача. Магістерське дослідження є самостійно виконаною роботою, в якій зображено особистий авторський підхід та особисто отримані теоретичні та прикладні результати, що відносяться до вирішення задачі оптимізації серверних обчислень. Формулювання мети та завдань дослідження проводилось спільно з науковим керівником.

Практична цінність. Отримані результати можуть використовуватися у майбутніх дослідженнях за напрямками:

- вдосконалення системи додатків
- аналіз проектування мікро-сервісних рішень

Апробація результатів. В рамках досліджень, що були виконані для

магістерської дисертації було зроблено доповідь в студентській науково-практичній конференції:

1. Kostyantyn Tykhonov, Yurii Gordienko. Method for optimization computing resources on the basis of specialized hybrid solutions. “The International Conference on Security, Fault Tolerance, Intelligence” (ICSFTI2020).

Основні положення, що виносяться на захист:

Метод оптимізації обчислювальних ресурсів програмними методами з використанням, різних архітектурних рішень до побудови додатку.

Ключові слова: гібридні системи, оптимізація, мікро-сервіс

Abstract

The urgency of the problem. The big problem of the modern world is the problem of expanding and updating applications of various directions. The speed of development of the modern world exceeds the human capacity to predict the direction of this development. That is why it is necessary to improve the migration process to newer, faster solutions to existing problems. A partial solution was to use architectural templates. But this is only part of the solution for situations in which there is a strict hierarchy. Therefore, the modern approach is looking for even more flexible solutions.

Purpose and tasks of the research. To achieve the objectives set in the master's thesis, used knowledge from the courses - cloud computing technology, distributed computing technology, computer systems design, and the basics of OOP.

To achieve the goal of the study, the following tasks were set and solved:

- Research of structure and principles of construction of microservice and serverless architecture
- Finding the best set of tools to implement these architectures
- Development of a software model of a hybrid without a server application

- Illustration of the model and analysis of the results

Object of study. The object of research is the process of application operation in a hybrid without a server system The subject of research - the interaction of software components in a system based on microservice architecture and uses a server-free approach

Subject of study. Game theories are used to achieve the goals set in the master's thesis.The scientific novelty of the obtained results is as follows:

- The method of analyzing the existing system and using the modular structure of the application to optimize calculations is proposed
- Developed a software product for the analysis of the constructed hybrid system
- The study allows to optimize the operation of existing software systems based on microservices

The novelty of the results. The master's research is an independently performed work, which reflects the personal author's approach and personally obtained theoretical and applied results related to solving the problem of server computation optimization. The formulation of the purpose and objectives of the study was carried out jointly with the supervisor.

The practical significance of the results of the work.

The obtained results can be used in future research in the following areas:

- Improvement of existing systems
- Analysis of designing micro service solutions

Approbation of results. As part of the research that was performed for the master's dissertation, a report was made at the student scientific-practical conference:

1. Kostyantyn Tykhonov, Yurii Gordienko. Method for optimization computing resources on the basis of specialized hybrid solutions. “The International Conference on Security, Fault Tolerance, Intelligence” (ICSFTI2020).

The main provisions to be defended:

A method of optimizing computing resources by software methods using various architectural solutions to build an web application.

Keywords: hybrid systems, serverless, optimization

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ

API – Application programming interface – прикладний програмний інтерфейс

AWS - Amazon Web Service - веб сервіси амазону

CCC - Code Carbon Copy - паперове копіювання коду

CVS - Control Version System - система контролю версій

DH - Docker Hub

FaaS - Function as a Service - Функція як сервіс

HTML – Hyper Text Markup Language – Мова розмітки гіпертексту

HTTP – Hyper Text Transfer Protocol – Протокол передачі гіпер-текстових документів

JSON - JavaScript Object Notation – Запис об’єктів JavaScript

PIC - platform independent code - код, який не залежить від середи розробки

MVC — Model-View-Controller - модель, зображення, контролер

REST – Representational State Transfer – «Передача репрезентативного стану»

URL – Uniform Resource Locator – Єдиний вказівник на ресурс

XML – Extensible Markup Language – Розширювана мова розмітки

XSLT - (eXtensible Stylesheet Language Transformations) - розширюємо мова трансформації стилів

ВСТУП

Сучасний напрямок розвитку програмного забезпечення не передбачає переробку і змінення архітектури, якої притримувалися при побудові додатку. Тенденція до інкапсуляції природно привела нас до постійного виділення окремих операцій в модулі та побудову систем які базуються на конкретній архітектурі. Змінювати фундамент програмного рішення це завжди починати з нуля, через це ми втрачаємо гнучкість рішень, але вирішуємо низку проблем.

Великі комерційної системи мають або модульний, або монолітний характер і будь-який з перерахованих вище підходів призводить до збільшення кількості коду викликаного постійним додаванням великого об'єму нового функціоналу. Будь-яка система з часом прагне до розширення або гине.

Передбачаючи таку тенденцію до зростання, отже збільшення навантаження на апаратну частину і отже підвищення вимог до обчислювальних потужностей, з'явилося безліч способів оптимізувати обчислень. Сучасні рішення дотримуються конкретної архітектури – мікро-сервісної або без-серверної. Але, важко повністю мігрувати на нову архітектуру маючи вже реалізоване рішення. Для цього необхідно використовувати поетапну міграцію частини функціоналу. Після детального аналізу використання певних функції програми необхідно розділити їх і тим самим оптимізувати їх використання для оптимізації навантаження на конкретний сервіс. В комбінації з хмарними обчисленнями ми можемо забезпечити плавну міграцію з однієї архітектури на іншу, або знизити навантаження на апаратну частину за допомогою підрахунку деяких операції в хмарних системах.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Методи оптимізації обчислювальних ресурсів

Основна мета оптимізації — підвищення продуктивності програми та економія обчислювальних ресурсів.

При аналізі наукових робіт на теми пов'язані з оптимізацією обчислювальних систем можна виділити три підходи:

1. Програмний підхід до оптимізації обчислювальних систем

Базується на аналізі та оптимізації коду програми [4, 6]

2. Оптимізація обчислювальних ресурсів за допомогою зміни та налаштування технічної бази на якій розміщено додаток

Базується на оптимізації використання серверних ресурсів. [5]

3. Оптимізація програмної Інфраструктурний всієї системи.

Ґрунтується на використанні спеціалізованих рішень для покращення та оптимізації систем для розміщення та встановлення додатків.

1.1.1 Програмний підхід до оптимізації обчислювальних ресурсів

Оптимізація знаходження та виправлення дублікатів програмного коду

Одна з проблем не оптимізованої програми — код який дублює вже реалізований функціонал програми.

Дублікат коду — елемент програми, який виконує ті самі команди з такою ж послідовністю з мінімальними відмінностями.

Пошук та видалення дублікатів — важливе завдання в процесі оптимізації програмних додатків. Об'єм коду впливає на розмір додатку та збільшує час необхідний для масштабування додатку.

Дублікати коду з'являються через практику копіювання фрагментів коду, та використання скопійованих фрагмента з мінімальними відмінностями.

Цей процес призводить до формування двох груп скопійованого коду:

1. Схожі за змістом

Схожі за змістом фрагменти коду — це програмні рішення, фрагменти, які повністю повторюють вже реалізовані команди. Вони є ідентичними за написанням.

2. Схожі за функціональністю

Схожими за функціональністю є фрагменти коду, які вирішують одні і ті ж самі завдання за допомогою різних підходів (бібліотек, архітектурних підходів, тощо).

В літературі, є консенсус щодо важливості виявлення та адресації дублікатів коду. Ручний пошук клонів коду дуже ускладнюється в великих системах і необхідна автоматична підтримка процесу аналізу та оптимізації коду. [1]

Для автоматизації виявлення дублікатів використовується декілька підходів:

1. Текстове порівняння

Цей підхід базується на базується на текстовому порівняні цілих рядків коду. Для підвищення продуктивності, рядки розподіляються за допомогою хеш-функцій для рядків і порівнюються рядки лише в одному розділі.

Результати аналізу зображуються у вигляді графіку, де кожна крапка позначає пару клонованих ліній. Клони шукаються за допомогою знаходження певних шаблонів поведінки в отриманому графіку. [2]

2. Порівняння за допомогою жетонів (токенів)

За основу використовуються текстове порівняння, але Але головна відмінність у використанні дерева суфіксів.

По-перше, кожна послідовність лексем для цілого рядка підсумовується так званим функтором, який резюмує конкретні значення ідентифікаторів та літералів [3]. Функтор однозначно характеризує цю послідовність лексем. Призначення функторів може розглядатися як ідеальна хеш-функція. Конкретні значення ідентифікаторів і літералів зображаються як параметри цього функтора. Дві лінії вважаються клонами, якщо вони відповідають їх функторам і кодуванню параметрів

Головне завдання після знаходження клонів — їх видалення. Є декілька підходів до цього завдання [4]:

1. Метод вилучення:

Метод вилучення полягає в тому, щоб замінити код-дублікат викликом новоствореної функції які містить спільний код фрагментів клону.

2. Метод вилучення та становлення:

Це метод через який клонований код, визначений у дочірніх класах, замінюється викликом функції новоствореному методу, що містить загальний код, і цей метод підтягується до найнижчого загального батьківського класу.

Також є ще ряд підходів до вилучення дубльованого коду — метод вилучення спільних функцій, метод вилучення класу та суперкласу, алгоритм заміни[5] та інші [4].

Паперове копіювання коду

CCC (**Code Carbon Copy**) - додаток вирішує проблему постійного переносу старого програмного коду в нові, сучасні рішення. Дуже часто цей перенос функціоналу призводить до втрати часу на механічну роботу.

Додаток CCC - може перенести функцію додатка донора в додаток реципієнт.

Розробник, який використовує CCC має ідентифікувати функцію для переносу в додатку донорі і місце де ця функція буде розміщена в додатку для реципієнт.

Донор та реципієнт повинні мати єдину програмну базу (написанні на одній і тій же мові)[6]

Додаток запрограмовано символічно записувати кожне значення, що обчислюється програмою. Програма фіксує усі вирази як функції, які мають значення та константи як параметри і трансформує їх байт код для представлення одержувачу [6].

В програмі існують вхідні і вихідні адаптери коду[6].

Оптимізація коду за допомогою машинного навчання

Компілятор — опора комп'ютерних наук [7, 8]. Переклад та оптимізація — це дві основні задачі компілятора.

Машинне навчання — це розділ інформаційних технологій про побудову систем, які можуть навчатися на проаналізованих даних. Навчання - покращення показника ефективності при виконанні конкретної задачі. Здатність систем машинного навчання передбачати події на основі попередньої інформації може використовуються для пошуку точки даних з найкращим результатом, найближчим до точки оптимізації.

Контрольоване навчання — це тип машинного навчання, при якому модель формується на основі мічених даних. Машина, на якій виконується навчання, отримує мічені об'єкти як

Оптимізація Spring Boot та віртуальної машини Java

Програмний код, на рівні бібліотеки та віртуальної машини Java хоча і може, мабуть, бути поліпшений, але це зовсім не тривіальна задача, яка не входить в порядок денний більшості програмістів, котрі використовують ці рішення.

Уся оптимізація на цьому рівні складається з уникнення помилок початківця. І більшість сучасних бібліотек ставлять не високий вхідний поріг для користувача. І на велику кількість помилок, що призводить до втрат пам'яті, чи дуже навантажує процесор виключена на етапах проектування фреймворку.

Алгоритми комплексної оптимізації використання обчислювальних ресурсів в хмарних системах

Для оптимізації обчислювальних ресурсів в роботі використовуються хмарні системи та програмні рішення адаптовані для роботи в таких системах.

Для роботи використовується система OpenStack.

OpenStack - це інтернет платформа яка використовується для створення обчислювальних хмар і хмарних сховищ.

Для оптимізації розроблено алгоритм балансування навантаження в хмарному сховищі даних. Стандартний алгоритм, що використовується в системі OpenStack, є неефективним тому, що не враховує маршрутизацію віртуальної і топологію локальної мережі, а також віддаленість віртуальної машини, яка виконує обробку запитів користувачів [9].

1.1.2 Оптимізація обчислювальних ресурсів за допомогою зміни та налаштування технічної бази на якій розміщено додаток

Оптимізація розподілених запитів між кластерами відмовостійкої обчислювальної системи

Об'єктом для дослідження та оптимізації буде обчислювальна система, яка складається з M локальних кластерів та загальнодоступного кластеру, які об'єднують m серверів.

Підвищення надійності та продуктивності розподілених обчислювальних систем, котрі об'єднують в цілісну систему декілька окремих кластерів, досягається динамічним розподілом запитів між ними враховуючи зміну навантаження кластерів чи їх відмови.

При оптимізації структури розглянутої обчислювальної систем шукається число серверів в локальних кластерах, число серверів в загальнодоступному кластері і кратність резервування комутаційних вузлів, що забезпечує максимум надійності системи при обмеженні вартість її реалізації та умови стаціонарності функціювання її вузлів.[10]

Мета оптимізації структури - мінімізація середнього часу перебування запитів в системі при обмеженні коштів на її побудову.

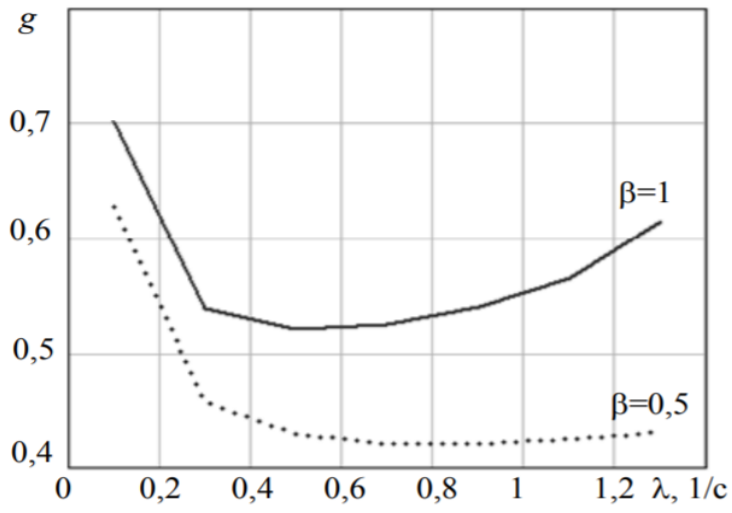


Рис. 1.1 Оптимальна кількість запитів, перерозподілена через мережу

Оптимізація відбувається шляхом перерозподілу потоку запитів які надходить в кластер. Шукається їх частка, що перерозподіляється через мережу в загальнодоступному кластері при якій мінімізується середній час перебування запитів.

1.1.3 Оптимізація інфраструктури веб додатків

Прискорення та оптимізація контейнеру Docker

Docker - програмне забезпечення для автоматизації розгортання і управління додатками в середовищах з підтримкою контейнеризації. Дозволяє «упакувати» додаток з усім його оточенням і залежностями в контейнер, який може бути перенесений на будь-яку Linux-систему з підтримкою cgroups в ядрі, а також надає середовище з управління контейнерами.

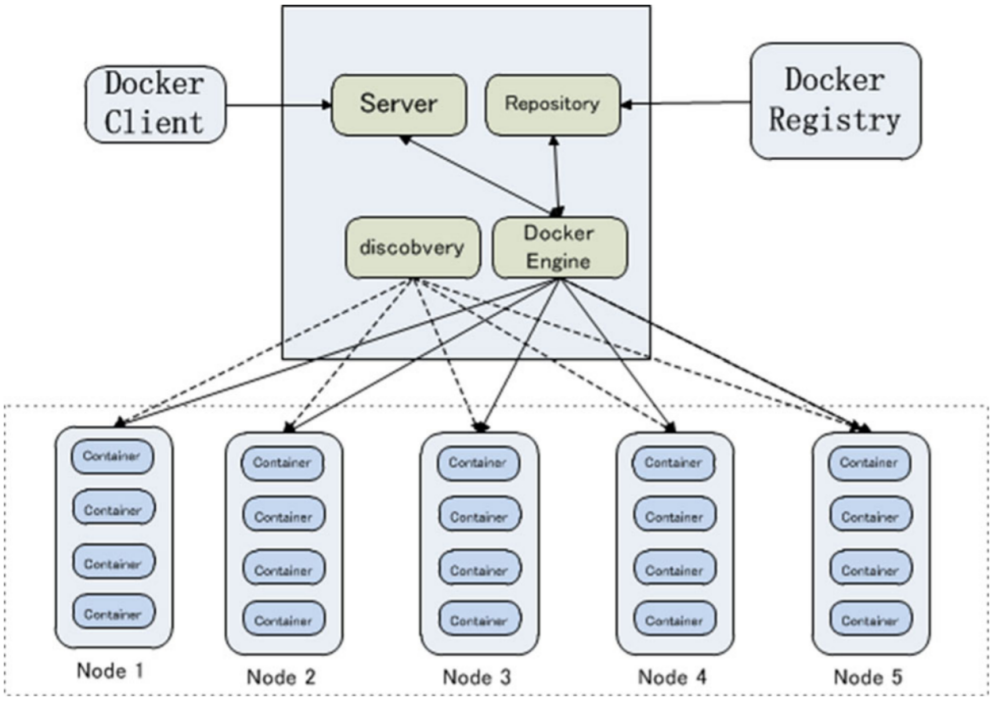


Рис 1.2 Структура системи Docker

Кластер контейнеру Docker використовує клієнт-серверну архітектуру, що можна побачити на рис.1.2.

Один з підходів до покращення продуктивності роботи контейнеру Docker – це удосконалення алгоритму розкладу. Новий алгоритм може базуватися на технології рою (swarm) та використовувати стратегію мульти-обєктної оптимізації, що враховує використання процесора та пам'яті кожного вузла, витрату часу передача зображень в мережі, асоціацію між контейнерами та кластеризація контейнерів.

В стані рою усі ноди діляться на два типи — менеджер та робітник. При цьому повноцінний кластер може існувати без робітничих нод. За замовчуванням менеджери також можуть виконувати роль робітника. [11]

У хмарному середовищі часто використовуються як контейнери, так і VM стеки разом, оскільки контейнери розміщуються поверх наданого віртуалізованого середовища [4]. Підсумовуючи це, контейнери - це по суті легкі віртуальні машини[3], де а працездатність контейнера майже у всіх випадках дорівнює або краща, ніж у віртуальної машини[4]. Порівняння інфраструктури показано на рисунку

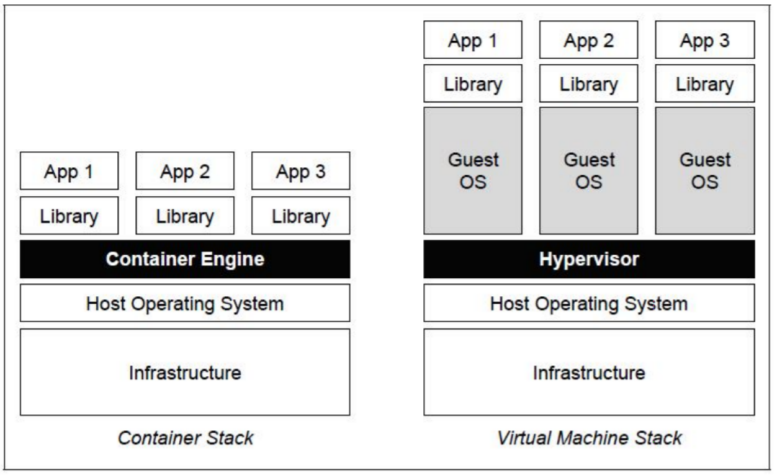


Рис 1.3 Порівняння стеків контейнера та віртуальної машини

Найбільш вивчений і успішний метод – це метод оптимізація загального призначення, техніка оптимізації колонії (ACO). ACO бере це натхнення в поведінці мурах, де мурахи відкладають слід який складається із спеціальної речовини з феромонами та позначають більш сприятливий шлях, яким слід йти іншим мурахам. Подібні механізм використовується для вирішення задач оптимізації [12]

Основні принципи підвищення продуктивності контейнерів:

1. Оптимізація вузьких місць
2. Відштовхуватися від потоку даних

Методи оптимізації обчислювальних ресурсів Docker:

Оптимізація перед побудовою образу на сервері:

1. Оптимізація Dockerfile
2. Перевірка релевантності контексту
3. Оптимізація часу затримки при оновленні контейнеру

Оптимізація побудованого образу

1. Оптимізація відношень між хостом та контейнером
2. Оптимізація даних про продуктивність контейнера
3. Використання AMP для полегшення даних про продуктивність

Зберігання низької ваги образу Docker

Маючи справу з зображеннями Docker, побудуйте Dockerfile. Dockerfile - це набір інструкцій, що описують способи побудови зображення. Dockerfile визначає набір деталей:

1. Файли які мають входити до контейнеру
2. Необхідні змінні середовища
3. Комманди які мають виконуватися

4. Кроки необхідні для інсталяції програмного додатку
5. Конфігурації мережі

Одна з частин файлу яка впливає на продуктивність це файловий контекст. Контекст окреслює вказані файли, необхідні для створення контейнера. Контекст можна зменшити додавши непотрібні файли до файлу - .dockerignore. Цей файл видалить обрані файли з контексту що зменшить його розмір. Перш за все необхідно проаналізувати додаткові бібліотеки, які можуть бути додані до контексту. Модифікований образ зі зменшеною вагою можно використовувати в інших проектах без перезавантаження тому що Docker використовує спільну файлову систему.

Зменшення часу затримки у мережі

У процесі побудови образу є аспекти, які використовують Інтернет (мережу) і залежать від швидкості в мережі. Якщо використовується великий образ контейнеру, це призводить до проблем з продуктивністю.

При побудові зображення Docker на локальному комп'ютері, система перевірятиме базове зображення, з якого створюється контейнер. Якщо вказане базове зображення не знайдено локально, Docker - за замовчуванням - спробує отримати зображення з Docker Hub, у якого є затримка продуктивності.

Docker Hub - це сховище, на якому зберігаються образи які використовуються для побудови контейнерів.

При використані сторонніх сервісів, а саме Docker Hub, виникає питання в їх надійності та стабільності. Необхідно врахувати наявність Docker Hub та ризики, пов'язані зі створенням сильної залежності від послуги цього сервісу. Саме для цього можливо використовувати свій приватний сервіс для зберігання образів.

Підвищення продуктивності роботи контейнера

Аналіз матричних даних контейнера

Що стосується вимірювання продуктивності контейнерів, нам знадобляться показники, які допомагають зрозуміти свою поточну ефективність. На щастя для нас, Docker надає нам інструменти для вилучення даних з наших запущених контейнерів для налагодження продуктивності.

1. **Команда docker stats**

Docker надає нам просту команду, яка називається docker stats, щоб отримати показники про запущені контейнери. Оскільки статистика докера настільки проста у використанні, давайте переглянемо те, що дає нам команда docker stats, і тоді ми зможемо зрозуміти, що ми можемо зрозуміти через дані.

```
CONTAINER ID NAME CPU % MEM USAGE / LIMIT MEM %
c796e97878c2 yourcontainer1 1.17% 35.71MiB / 1.952GiB 1.79%
2c02afe562b8 yourcontainer2 0.04% 9.344MiB / 1.952GiB 0.47%
0133d95251a1 yourcontainer3 0.00% 6.363MiB / 1.952GiB 0.32%
```

Рис 1.4 - приклад роботи docker stats

Платформа для аналізу Spring додатку - JMH Core. Для використання платформи для тестування необхідно додати дві залежності - jmh-core та jmh-generator-annprocess. Для ініціалізації бенчмарку необхідно створити клас з анотацією @Benchmark.

Старт Spring додатку (WebFlux) з простим контролером:

Таб. 1.1

Дані отримані після запуску базового додатку Spring за допомогою WebFlux

Платформа	**Режим**	**К-сть**	**Оцінка**	**Помилка**	**Частини**
Benchmark.init	ss	10	2.818 +-	0.287	s/op

WebFlux - це програмна бібліотека для реалізації реактивних веб додатків.

Старт Spring додатку (WebMVC) з простим контроллером:

Таб. 1.2.

Дані отримані після запуску базового додатку Spring за допомогою WebFlux

Платформа	**Режим**	**К-сть**	**Оцінка**	**Помилка**	**Частини**
Benchmark.init	ss	10	3.238 +-	0.317	s/op

Тестування додатку з індексацією контексту

Для тестування веб додатку необхідно додати залежність spring-context-indexer.

Таб. 1.3

Дані отримані після запуску базового додатку Spring за допомогою WebFlux та з контекстними залежностями

Платформа	Режим	К-сть	Оцінка	Помилка	Частини
Benchmark.init	ss	10	3.011 +-	0.112	s/op

Використання ледачої ініціалізації

За замовчуванням, Spring створює всі поодинокі біни під час запуску додатку контексту веб додатку. Головна причина - перевірка помилок під час запуску.

Для старту ледачого елемента Bean використовується анотація @Lazy. Анотація @Lazy використовується в конфігураційному класі (з анотацією @Configuration).

Анотація @Lazy може бути використана в @Component та @Service, та при ініціалізації, комбінуючи з анотацією @Autowired.

Таб. 1.4

Дані отримані після запуску базового додатку Spring за з використанням ледачого старту.

Платформа	Режим	К-сть	Оцінка	Помилка	Частини
Benchmark.init	ss	10	2.821 +-	0.127	s/op

Прискорення за допомогою вимкнення верифікації

Додавання параметру -noverify

Таб. 1.5.

Дані отримані після запуску базового додатку Spring за з використанням параметру -noverify

Платформа	Режим	К-сть	Оцінка	Помилка	Частини
Benchmark.init	ss	10	2.591 +-	0.067	s/op

Вимкнення JMX для прискорення веб додатку

Вимкнення jmx - -Dspring.jmx.enabled=false

Таб. 1.6.

Дані отримані після запуску базового додатку Spring за з використанням параметру jmx - -Dspring.jmx.enabled=false

Платформа	Режим	К-сть	Оцінка	Помилка	Частини
Benchmark.init	ss	10	2.867 +-	0.093	s/op

Вимкнення Logback

Для вимкнення Logback необхідно модифікувати dependency - додати exclusion для spring-boot-starter-logging

Таб. 1.7.

Дані отримані після запуску базового додатку Spring з виключенням залежності spring-boot-starter-logging

Платформа	Режим	К-сть	Оцінка	Помилка	Частини
Benchmark.init	ss	10	2.904 +-	0.097	s/op

Вимкнення Jackson

Для вимкнення Jackson необхідно модифікувати dependency - додати exclusion для spring-boot-starter-json

Таб. 1.8.

Дані отримані після запуску базового додатку Spring з виключенням залежності spring-boot-starter-json

Платформа	Режим	К-сть	Оцінка	Помилка	Частини
Benchmark.init	ss	10	2.774 +-	0.093	s/op

Вимкнення Hibernate Validator

Для вимкнення Hibernate Validator необхідно модифікувати dependency - додати exclusion для spring-boot-hibernate-validation

Таб. 1.9.

Дані отримані після запуску базового додатку Spring з виключенням залежності spring-boot-hibernate-validation

Платформа	Режим	К-сть	Оцінка	Помилка	Частини
Benchmark.init	ss	10	2.854 +-	0.081	s/op

1.2 Аналіз структури веб додатку

Для оптимізації обчислювальних ресурсів необхідно розуміти архітектурні підходи, що використовуються при побудові додатку.

Програма має базуватися на сталих архітектурних рішеннях та використовувати перевірені шаблони програмування. Розуміння та аналіз цих підходів та архітектур є першим етапом до оптимізації роботи програми.

1.2.1 Основні шаблони які використовуються для проектування веб додатків

Для проектування веб додатку необхідно обрати один з багатьох шаблонів для проектування.

MVC — Model-View-Controller

MVC - фундаментальний шаблон програмування, який допомагає відділити бізнес логіку від інтерфейсу користувача. Схематичне зображення роботи за шаблоном MVC зображене на рис. 1.5.

Головна задача шаблону проектування MVC – послаблення зв'язку між різними рівнями додатку та спрощення подальшої розробки. Цей шаблон широко застосовується при розробці веб-додатків які базуються на технології J2EE. Але традиційний MVC, не є універсальним рішенням складних задач розробки.(13)

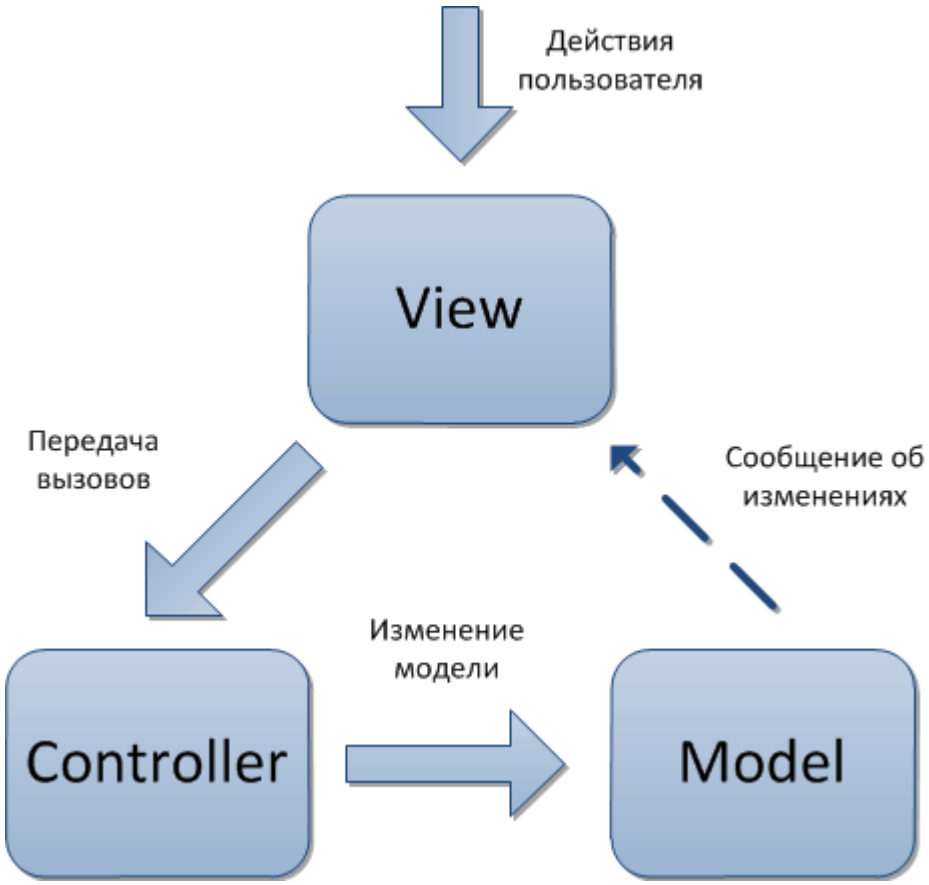


Рис. 1.5 - Схема алгоритму реалізованому за паттерном MVC

Додаток побудований по шаблону MVC має три рівні, котрі закладні в його назві:

1. View (Зображення, Вид) відповідає за зображення даних Моделі, - як правило, генеруючи HTML, які ми бачимо в своєму браузері.
2. Model (Модель) інкапсулює (об'єднує) дані програми, в цілому вони будуть складатися з POJO («Старих добрих Java - об'єктів», або бінов).
3. Controller (Контролер) обробляє запит користувача, створює відповідну Модель і передає її для зображення в Вид.

MVP - Model-View-Presenter (MVP)

MVP - шаблон проектування походить від MVC. Призначений для побудови інтерфейсу користувача.

Елемент Presenter - виконує роль посередника і відповідає за управління подіями призначеного для користувача інтерфейсу.

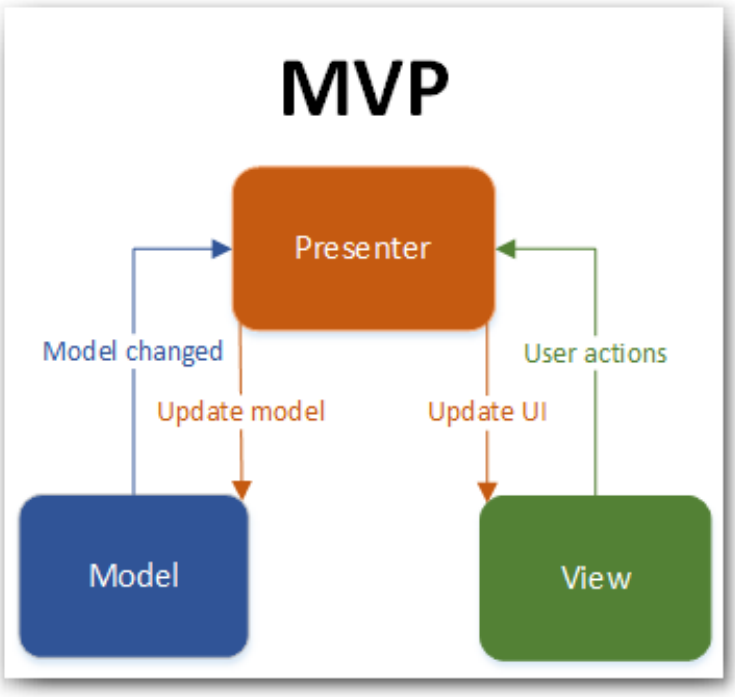


Рис. 1.6. Схематичне зображення шаблону MVP

На рис. 1.6 схематично зображено відношення між різними елементами шаблону MVP.

Passive View

У цьому уявленні інформація про модель відсутня. Але замість відображення інформації з моделі він гадає прості властивості для всієї інформації, яку необхідно відобразити на екрані.

Supervising Controller

Ця варіація MVP поінформована про існування моделі і відповідає за зв'язування даних з відображенням

Найпростіший додаток Spring складається лише з одного классу який знаходиться в кореневому каталозі src

Базується одному з двох фреймворків для автоматизації збірки проектів - Maven або Gradle. Та має конфігураційний файл - application.properties (це стандартна назва, яка може бути змінена за допомогою JAVA_OPT).

Подальша шлях для побудови будь-якого додатку обирає сам розробник. Ніяких заборон майже не існує. Ми будемо розглядати лише один архітектурний підхід до побудови Spring додатків, а саме MVC (Model, View, Controller)

1.2.2 Програмний аналіз структури додатку

Аналіз програмного код для оцінки продуктивності

Інструмент компілятора для прогнозування ефективності реалізовано за допомогою системи SUIF (Stanford University Infrastructure Function).

Система SUIF - це компіляційна інфраструктура, що базується на конкретному програмному представленні, що також називається SUIF (проміжний формат стенфордського університету). Акцент робиться на максимізації повторного використання коду. Система базується на двох фундаментальних концепціях: які масштабують програмні рішення. Система забезпечує набір попередньо визначених вузлів та залежностей. [14]

Пошук вихідного коду за допомогою коментарів в системі для контролю за версіями CVS

CVS(англ. Control Version System - система для контролю версій) – це система контролю версій, що широко використовується в спільнотах які працюють над програмами з відкритим кодом (10)

Для пошуку рядків коду за допомогою коментарів CVS, використовується карта значень, яка зберігає коментар як ключ і відповідний рядок коду як значення відповідно до цього коментаря.

На відміну від команди cvs annotate, яка показує останню редакцію, карта значень записує всі зміни у коді.

1.3 Підходи до генерації програмного коду

Генерація програмного коду, яка базується на компонентах

Генерація, що базується на компонентах використовує список інструкцій для генерації, та список параметрів для генерації вихідного коду.

Розробник вказує генератору, що необхідно створити за допомогою інструкцій для генерації. За допомогою, набору інструкції, вказується список цільових компонентів, які необхідно створити.

Набір інструментів, що використовується для генерації буде скорочувати час, який розробник втрачає на генерацію.

Генерація програмного коду за допомогою UML діаграм

Основна ідея інструменту - генерація коду, що базується на специфікаціях UML.

UML (англ. Unified Modeling Language - уніфікована мова моделювання) - мова графічного опису для об'єктного моделювання в області розробки програмного забезпечення, для моделювання бізнес-процесів, системного проектування та відображення організаційних структур.

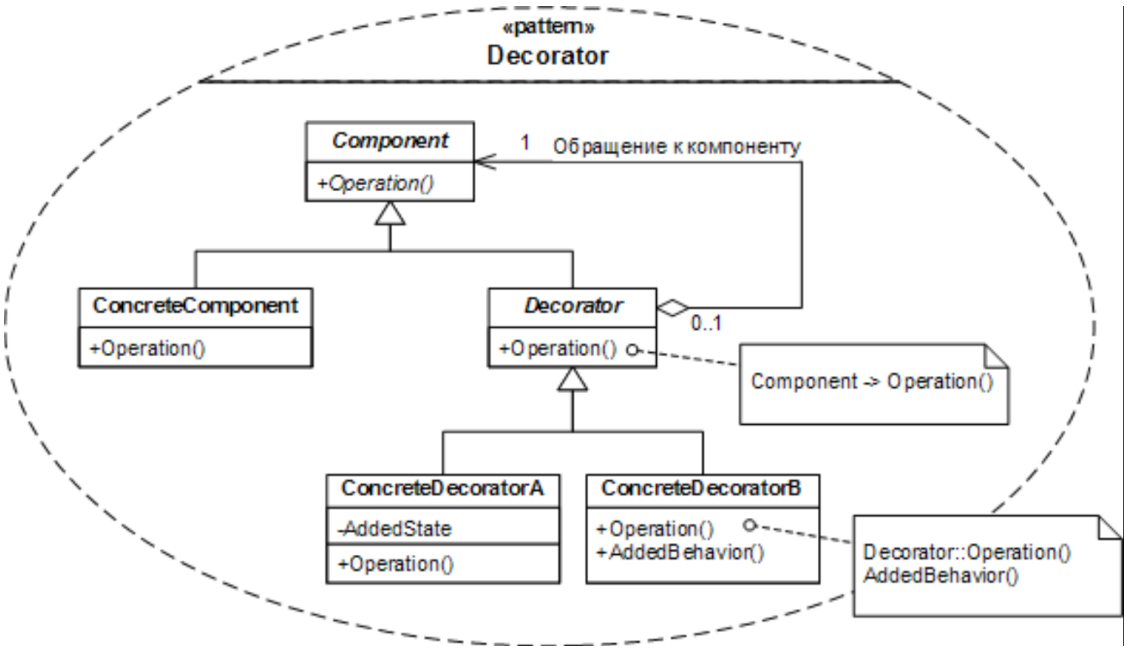


Рис. 1.7. Приклад шаблону декоратор у форматі UML діаграми

За основу для побудування, очікується, що специфікації будуть максимально

багатими та еластичними щодо мови перекладу. Описи моделей можуть

залежати або не залежить від цільової мови. Описи цільової мови зберігаються в атрибутах, визначених у спільних профілях, а рішення, які є залежними від цільової мови зберігаються в атрибутах, визначених у профілях, специфічних для цільової мови.

Генерація коду проводиться за допомогою двох ланцюгових перетворень.

Перше перетворення подібне до серіалізатора XMI (англ. XML Metadata Interchange) в моделі UML [3] з тією різницею, що проміжний файл коду генерується для кожного елемента моделі. Потім цей елемент моделі визначається як окремий файл-об'єкт. Усі дані, що зберігаються у моделі UML, переписуються у вигляді файлу XML (англ. Extensible Markup

Language) заздалегідь визначеного формату, який називається платформою незалежного коду. Шаблони для першої трансформації є незалежними та інваріантними щодо мови перекладу.

Друге перетворення здійснюється за допомогою процесора XSLT та модульно написаних шаблонів XSLT для кожної мови перекладу.

XSLT - (eXtensible Stylesheet Language Transformations) - мова перетворення XML-документів. Специфікація XSLT входить до складу XSL і є рекомендацією W3C.

Вхід у цю трансформацію - файл PIC (platform independent code) та шаблони, застосовані до нього. Вихід процесора XSLT - це файл вихідного коду цільової мови як результат специфікацій шаблону. Завдання щодо зміни коду виконується на завершальній фазі, оскільки PIC перетворюється на цільову мову за допомогою шаблонів XSLT.

ВИСНОВКИ ДО РОЗДІЛУ 1

Вступна частина першого розділу була присвячена короткому огляду предметної області, що розглядається в даній роботі.

Була введена класифікація типів методів з оптимізації обчислювальних ресурсів та наведене стисле пояснення основних підходів до оптимізації.

В пункті 1.1 розглянуті основні методи для оптимізації обчислювальних ресурсів. Проаналізовані базові напрямки оптимізації на прикладі останніх робіт по темі.

Проаналізована оптимізація програмного коду для підвищення продуктивності роботи додатку та для економії обчислювальних ресурсів. Проаналізовані методи які використовуються для виправлення головних проблем та недоліків, які можуть виникнути при процесі розробки програмних рішень.

Розглянуто методи з оптимізації контейнерів для зберігання додатків. Проаналізовані сучасні рішення для підвищення продуктивності їх роботи

В пункті 1.2 розглянуте питання автоматизації аналізу коду програми. Розглянуті рішення для інтерпретації та зчитування вихідного коду програми.

В пункті 1.3 розглянуто основні підходи до автоматизації написання та генерації програмного коду. Описані основні алгоритми, що використовуються при генерації.

Серед проаналізованих підходів та рішень для оптимізації обчислювальних ресурсів було обрано декілька підходів до генерації та

аналізу вихідного коду. Метод який буде використовуватися для оптимізації відноситься до програмного рішення.

При дослідженні предметної області та пошуку готових існуючих рішень для подальшого порівняння — аналогів не знайдено.

РОЗДІЛ 2

ПРОЕКТУВАННЯ МЕТОДУ ОПТИМІЗАЦІЇ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ

2.1 Огляд існуючих засобів реалізації

В розділі будуть розглянуті основні програмні рішення, продукти та підходи, що будуть використовуватися для реалізації обраного підходу. Буде проаналізована фундаментальна основа, яка використовується для реалізації моделі, котра буде підтверджувати ефективність методу оптимізації.

2.1.1 WebSocket

WebSocket - протокол зв'язку поверх TCP-з'єднання, призначений для обміну повідомленнями між браузером і веб-сервером в режимі реального часу.В даний час в W3C здійснюється стандартизація API Web Sockets. Чорновий варіант стандарту цього протоколу затверджений IETF.

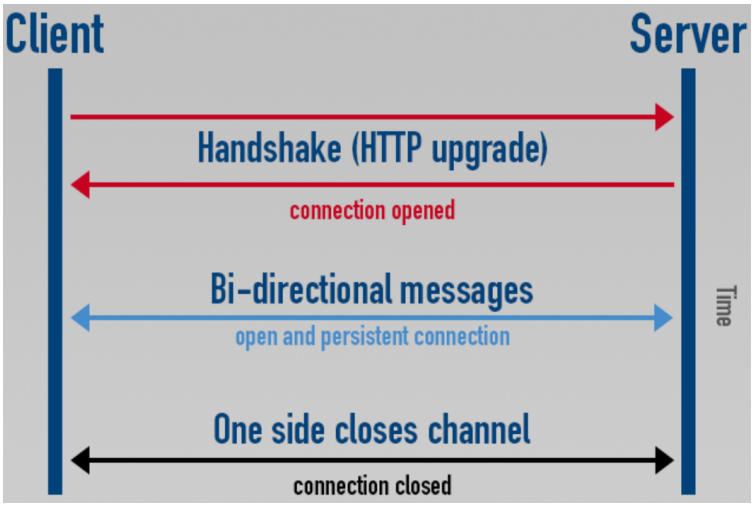


Рис. 2.1 Модель спілкування між клієнтом і сервером при підключені через WebSocket

Підключення через WebSocket ініціалізується за допомогою рукостискання (handshake), яке може налаштовує з'єднання між клієнтом і

сервером, яке може бути як одностороннім так і двостороннім, що можно побачити на рис 2.1.

2.1.2 HTTP

HTTP (HyperText Transfer Protocol) - протокол прикладного рівня передачі даних спочатку - у вигляді гіпертекстових документів в форматі «HTML», зараз використовується для передачі довільних даних. (16)

Код відповіді (стану) HTTP показує, чи був успішно виконаний певний HTTP запит. Коди згруповані в 5 класів: інформаційні, успішні, перенаправлення, помилки клієнта і помилки сервера

Список усіх типів HTTP протоколів:

1. Інформаційні - 1XX (100….103)

Інформування про процес передачі.

2. Успішні - 2XX (200...206)

Інформування про випадки успішного приймання та обробки запиту клієнта. Залежно від статусу, сервер може ще передати заголовки і тіло повідомлення.

3. Сповіщення про перенаправлення - 3XX (300...308)

Повідомляє клієнту, що для успішного виконання операції необхідно зробити інший запит (як правило по іншому URI).

4. Клієнтські - 4XX (400...417)

Вказівка помилок з боку клієнта. При використанні всіх методів, крім HEAD, сервер повинен повернути в тексті листа гіпертекстове пояснення для користувача.

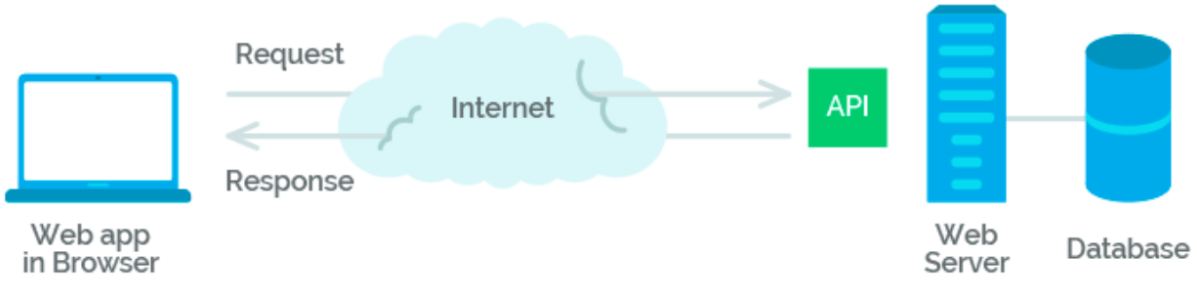
5. Серверні - 5XX (500...505)

Інформування про випадки невдалого виконання операції з вини сервера. Для всіх ситуацій, крім використання методу HEAD, сервер повинен

включати в тіло повідомлення пояснення, яке клієнт буде відображати користувачеві.

2.1.3 REST

REST (Representational State Transfer, «передача репрезентативного стану») — спосіб реалізації архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів. Був задокументований Роєм Філдінгом, одним із творців протоколу HTTP.



Р

Рис 2.2 Схематичний принцип роботи системи побудованої за принципами REST

Для побудови системи, яка базується на архітектурному підході REST мати можливість відправляти запити та отримувати відповіді, що зображено на рис 2.2. Спілкування проходить по одному з 8 стандартних HTTP повідомлень.

Для кожної одиниці інформації визначається 5 дій, а саме:

1. GET /{item} - отримати список об'єктів
2. GET /{item}/{id} - отримати специфічний об'єкт
3. PUT /{item} - оновити значення об'єкту

4. POST /{item} - створити новий об'єкт
5. DELETE /{item}/{id} - видалити об'єкт

2.1.4 RKT

CoreOS - легка операційна система з відкритим вихідним кодом на базі ядра Linux. Призначена для створення інфраструктури комп'ютерних кластерів, особливу увагу приділено автоматизації, спрощення провадження додатків, безпеки, надійності і масштабованості. В якості операційної системи CoreOS надає лише мінімальну функціональність, необхідну для розгортання додатків всередині програмних контейнерів, засоби виявлення сервісів і передачі налаштувань.

2.1.5 Бібліотека Spring Framework (Java)

Spring - це вільно поширена бібліотека яка була створена Родом Джонсоном. Він був спроектований з метою усунути складнощі розробки корпоративних додатків і зробити можливим використання простих компонентів Java Bean для досягнення всього того, що раніше було доступно тільки з використанням EJB. [17]

Вся логіка роботи Spring MVC побудована навколо DispatcherServlet, який приймає і обробляє всі HTTP-запити (з UI) і відповіді на них. Робочий процес обробки запиту DispatcherServlet проілюстрований на наступній діаграмі:

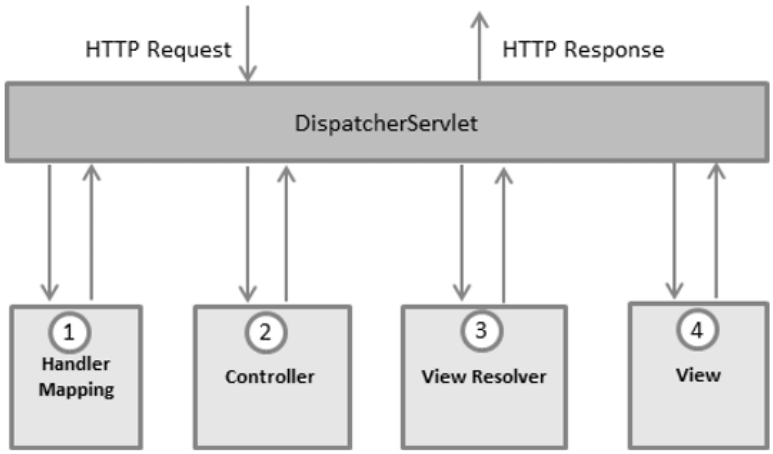


Рис 2.3 Спілкування з додатком через DispatcherServlet

Нижче наведена послідовність подій, відповідна входить HTTP-запиту:

Після отримання HTTP-запиту DispatcherServlet звертається до інтерфейсу HandlerMapping, який визначає, який Контролер повинен бути викликаний, після чого, відправляє запит в потрібний Контролер.Контролер приймає запит і викликає відповідний службовий метод, заснований на GET або POST. Викликаний метод визначає дані Моделі, засновані на певній бізнес-логікою і повертає в DispatcherServlet ім'я Віда (View).За допомогою інтерфейсу ViewResolver DispatcherServlet визначає, який Вид потрібно використовувати на підставі отриманого імені.Після того, як Вид (View) створений, DispatcherServlet відправляє дані Моделі у вигляді атрибутів в Вид, який в кінцевому підсумку відображається в браузері.

Всі вищезгадані компоненти, а саме, HandlerMapping, Controller і ViewResolver, є частинами інтерфейсу WebApplicationContext extends ApplicationContext, з деякими додатковими особливостями, необхідними для створення web-додатків.

2.1.6 Spring Actuator

Actuator займається моніторингом додатка, збиранням показників, аналізом трафіку або стану бази даних.

Основна перевага цієї бібліотеки полягає в тому, що ми можемо отримати інструменти виробничого класу без необхідності реально реалізовувати ці функції.

Привід використовується в основному для викриття оперативної інформації про запущене додаток - здоров'я, метрики, інформація, дамп, env тощо. Він використовує кінцеві точки HTTP або боби JMX, щоб ми могли взаємодіяти з ним.

Як тільки ця залежність опиняється на шляху до класу, нам доступні кілька кінцевих точок. Як і у більшості модулів Spring, ми можемо легко налаштувати або розширити його багатьма способами.

2.1.7 JHipster

JHipster генератор коду високого рівня, побудований на великому списку передових інструментів і платформ розробки.
Основними компонентами інструменту є:

1. Yeoman, інтерфейсний інструмент для створення
2. Spring Boot
3. AngularJS, выдающаяся структура Javascript
4. Swagger
5. Maven , Npm

2.1.8 Бібліотека JavaPoet

JavaPoet розроблений компанією Square, яка надає API для генерації вихідного коду Java. Бібліотека може генерувати примітивні типи, посилальні типи і їх варіації (такі як класи, інтерфейси, що перераховуються типи, анонімні внутрішні класи), поля, методи, параметри, анотації і Javadocs.

JavaPoet автоматично управляє імпортом залежних класів. Він також використовує шаблон Builder, щоб вказати логіку для генерації коду Java.

Специфікація основних методів JavaPoet

1. methodBuilder () - метод для генерації об'єкта Builder для методів
2. addStatement () - метод для додавання виразу до блоку коду
3. beginControlFlow () - метод для генерації початку нового блоку коду
4. endControlFlow() - метод, що закінчує блок коду
5. addModifiers() - метод для додавання модифікаторів доступу

2.1.9 Docker

Docker - програмне забезпечення для автоматизації розгортання і управління додатками в середовищах з підтримкою контейнеризації.

У контейнерів (майже завжди) є хости. Вони потребують того щоб обчислювальні машин працювали, але не сподівайтеся, що вони будуть оптимально налаштовані з коробки. У роботі з контейнерами потрібно проаналізувати необхідні ресурси, які будуть використовуватися контейнером.

Контейнери будуються із шарів та використовують єдину файлову систему. Це означає, серед іншого, під час створення зображення Docker (як частини вашого Dockerfile) воно залишає кеш для підвищення продуктивності. Важливо, що ці кешовані шари є добавками, а це означає, що ви можете їх тільки додавати.

2.1.10 Kubernetes

Kubernetes - відкрите програмне забезпечення для автоматизації розгортання, масштабування додатків, що були розміщені в контейнерах, та управління ними. Підтримує основні технології контейнеризації,

включаючи Docker, rkt, також можлива підтримка технологій апаратної віртуалізації.

Запити та обмеження

Kubernetes дозволяє налаштовувати обмеження на такі ресурси процесора, пам'яті та локальне ефемерне сховище (бета-функція в v1.12). Такі ресурси, як CPU, є стислими, це означає, що контейнер буде обмежений за допомогою політики управління процесором. Інші ресурси, як-от пам'ять, контролюються за допомогою Kubelet і примусово зупиняються, якщо вони переходять межу. Використовуючи різні конфігурації запитів і обмежень, можна досягти різних якостей обслуговування для кожного робочого навантаження.

Обмеження

Обмеження - це верхня межа робочого навантаження, дозволеного споживання. Переступивши потрібний граничний поріг, Kubelet змусить вбити pod. Якщо не встановлено обмежень, робоче навантаження може споживати всі ресурси на даному вузлі. Якщо працює кілька платформ, які не мають обмежень, ресурси будуть розподілятися на основі найкращих зусиль.

Запити

Є три основні аспекти в оцінці якості обслуговування (QoS), які можна отримати за допомогою ресурсів та лімітів - найкраща конфігурація QoS залежатиме від потреб платформи.

Гарантовані QoS

Гарантовані значення QoS можуть бути отримані за допомогою встановлення тільки лімітів. Це означає, що контейнер може використовувати усі ресурси.

Налаштування запитів та лімітів

Ключ до налаштування запитів та лімітів полягає в знаходженні переломного моменту в роботі одного pod (частки). Використовуючи кілька різних технік по тестовому навантаженню дає змогу зрозуміти різні стани при яких додаток закінчує своє виконання.

2.1.10 AWS

AWS (Amazon Web Services) - комерційне публічне хмара, яка підтримується і розвивається компанією Amazon з 2006 роки. Надає передплатникам послуги як по інфраструктурної моделі (віртуальні сервери, ресурси зберігання), так і на рівня платформи (хмарні бази даних, хмарне сполучна програмне забезпечення, хмарні обчислення, засоби розробки).

AWS Lambda дозволяє автоматично запускати програмні коди без необхідності у виділенні серверів або управлінні ними. Достатньо написати програмний код і завантажити його в Lambda.

AWS Lambda автоматично збільшує кількість ресурсів які виділяються для додатку, запускаючи програмний код у відповідь на кожен тригер. Всі запущені коди виконуються паралельно, при цьому кожен тригер обробляється індивідуально, що забезпечує масштабування відповідно до робочої навантаженням.

Рис 2.4 Приклад алгоритми роботи системи AWS lambda

Можна налаштувати обробку даних в AWS Lambda відразу після їх завантаження в Amazon S3. Наприклад, в Lambda можна створювати мініатюри зображень, змінювати кодування відео, індексувати файли, обробляти журнали, перевіряти зміст, а також агрегувати і фільтрувати дані - і все це в режимі реального часу.

2.2 Огляд архітектурних рішень при побудові веб додатків

2.2.1 Без-серверні обчислення

Бессерверні обчислення (англ. Serverless computing) - модель хмарних обчислень, в якій платформа динамічно керує виділенням машинних ресурсів. Іноді бессерверной обчислення також називають

«Функція як послуга» (англ. Function as a Service, FaaS), тому що одиницею додатку є функція, яка виконується платформою. (18)

Дослідження принципів побудови без сервісних програмних рішень

Ситуації в котрих використовується без-сервісна архітектура:

1. Коли існує загальна хмарна інфраструктура, яка не потребує забезпечення серверів (бази даних на вимогу, послуги контейнерів тощо)

2. Коли ця хмарна інфраструктура функціонує як одна з таких служб, як AWS Lambda, Google Cloud Functions та Azure Functions.

Для аналізу роботи архітектури яка використовує функцію як послугу візьмімо Spring Cloud Data Flow. Основні принципи без-серверної архітектури будуть спільними з іншими рішеннями для різних платформ та мов програмування

Spring Cloud Data Flow використовує термін “завдання” та описує їх як "короткочасні мікросервіси". Функції, в FaaS, по суті і є - короткочасними мікросервісами.

Функції виходять надзвичайно деталізованими. Вони повинні виконувати лише одну річ - і недовговічні, оскільки вся основна та інфраструктура як повністю руйнується після завершення.

Порівняння

Наведена нижче таблиця (Таб 2.1) ілюструє різницю в аспекті зусиль, необхідних для масштабування, і загального очікуваного терміну експлуатації базового часу виконання та спеціальної інфраструктури різних підходів до реалізації програмних рішень.

Таб. 2.1

Порівняльна таблиця довговічності експлуатації різних платформ для розгортання програмних додатків

Тип платформи яка використовується	**Життєвий цикл**	**Час масштабування**
Функції	від мілісекунд до секунд	миттєвий(час виконання запиту)

Контейнери	від годин до днів	секунди
Віртуальні Машини	від тижня до року	мінути
Фізична інфраструктура	роки	від дня до тижня

Для функцій термін служби базової інфраструктури на мілісекунд довший, ніж фактичний час виконання функції. Запропоновані контейнери можуть працювати тривалий час, оскільки вони справді є лише крихітними машинами, і теоретично їх не потрібно зупиняти. Однак, як правило, кожну оновлення призводить до необхідності перезавантажити та перезапустити контейнер.

Виділені VM, як і EC2, як правило, не змінюється та не перезапускаються навіть після нових оновлень програмного коду і, оновлюються набагато рідше і викликано це глобальним оновленням системи.

В свою чергу фізичні сервери, з великими передовими інвестиціями та високими витратами на обслуговування, як правило, проектуються з метою їх зберігання протягом багатьох років, а часто й десятків років.

В той самий час, функції спрацьовують миттєво, оскільки основний час виконання передбачається на основі запиту, а для практичних цілей хмарні постачальники, такі як Google і AWS, мають необмежений масштаб. Контейнери, за умови, що ми використовуємо щось на кшталт Pivotal Cloud Foundry або Heroku, дозволяють нам проводити масштабування протягом декількох секунд.

Якщо ваш екземпляр EC2 підтримує Elastic Beanstalk або щось подібне, ви можете масштабувати від n до n + 1 протягом декількох хвилин.

Фізичні сервери, природно, вимагатимуть замовлення, доставки, монтажу, налаштування, загартування тощо, що може зажадати тижнів чи навіть місяців.

Вартість - ще один аспект порівняльного аналізу хмарних систем. FaaS є економічно вигідним для низьких навантажень, оскільки ви платите лише за фактичну обробку (і через кілька мілісекунд після).

Системи що базуються на контейнерах все ще досить дешеві, але вам потрібно тримати його запущений 24/7, навіть якщо є багато часу, коли не спостерігається завантаження.

Вартість FaaS збільшується лінійно з завантаженням. Але як тільки ми перейдемо межу за якою економічно вигідніше тримати сервіс завжди запущений, має більше сенсу масштабувати контейнери.

Питання полягає в тому, чи є навантаження раптове або розподіленим.

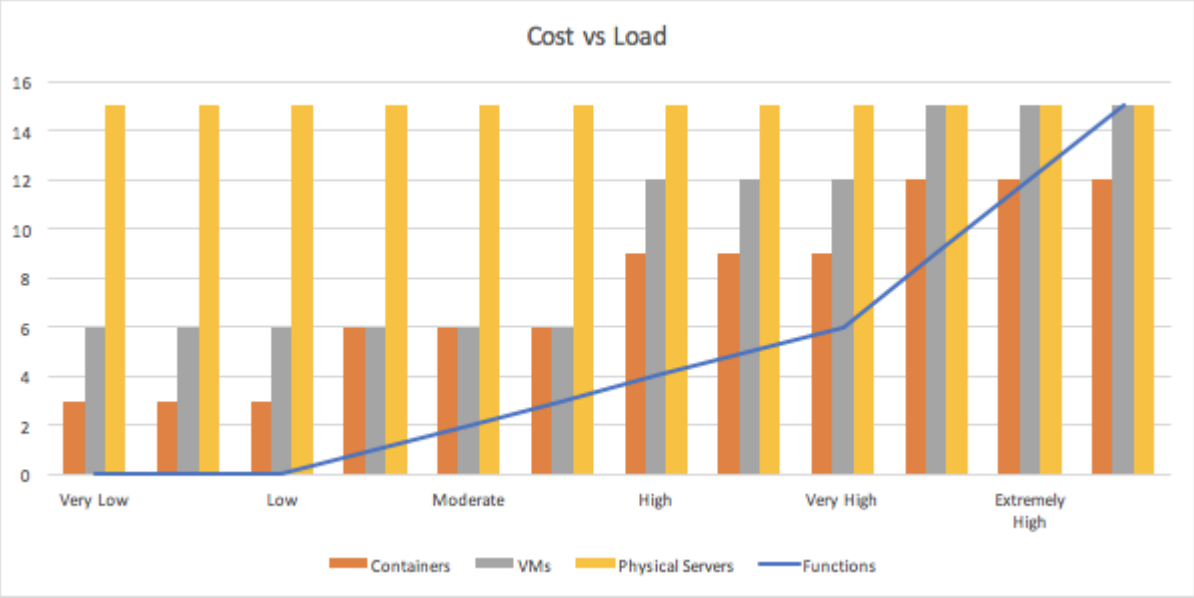


Рис 2.4 Порівняльна діаграма ціни до навантаження

Якщо велике навантаження розподіляється на більший часовий період і змінюється циклічно, послуга, яка завжди є доступною - виграє.

Якщо навантаження локалізується в часі і відбувається раптовими сплесками, навіть при великому обсязі, в підході який базуватиметься на FaaS може мати більше сенсу в фінансовому плані.

Використовуючи щось на зразок Opsworks, масштабування може бути налаштовано автоматично для машин які базуються на платформі EC2, але воно все ще є досить грубим, оскільки для завантаження на EC2 платформу потрібен значний час.

За допомогою фізичних серверів нам потрібно забезпечити і підтримувати працездатність серверів при максимальному навантаженні в перший день. Навіть гірше, ми маємо оцінити максимальне навантаження в нульовий день.

Компроміси та вигоди

FaaS надає можливість більш легкого обслуговувати програмні рішення на хмарних платформах та призведе до потенційно нижчих витрат

Продуктивність

Найбільший вплив на продуктивність має холодний час старту. Необхідно передбачити поведінку нового контейнеру та новий час виконання програмного коду.

З іншого боку, якщо у вас був традиційний контейнерний додаток (Spring Boot, Express JS), сервер завжди готовий обробити вхідний запит. Як правило, холодний час запуску становить 100 мс, тоді як функція працюватиме 1-2 секунди, внаслідок чого компонент часу холодного старту складає приблизно 5% -15% від загального часу виконання (і ви платите за це кожен раз, коли ви відправляєте запит до функції). Це те, що потрібно правильно оцінити, вибираючи FaaS як варіант.

Технічне обслуговування

Кожну функцію легше підтримувати, але в цілому набір незалежно розгорнутих функцій покладе на команду більший тягар, ніж одна служба, яка охоплює багато функцій.

Навіть якщо багато функцій як групи живуть в одному репо, їм знадобляться індивідуальні сценарії побудови та конвеєри розгортання; додатково нам потрібно буде стежити за кількома журналами.

Зниження витрат

Менші витрати - реальна вигода за умови того, що функції не використовуються постійно. Можливо легко розрахувати та оцінити необхідні ресурси. Якщо функції будуть називатися постійно, FaaS - це не правильний підхід, але якщо це трапляється періодично то використання Faas - виправдовує себе.

Безпека

Це великий виграш. Зловмисний хакер не може зламати ефемерний екземпляр - безпека зараз переймається насамперед проблемою для постачальника інфраструктури. Природно, є й інші вразливості - в деяких аспектах FaaS збільшує ризики. Уразливості на рівні коду все ще існують.

Анотація відсутня автентифікація

Аутентифікацію слід обробляти як окрему службу, а функцію слід піддавати лише ідентифікованим користувачам. Це можна легко досягти, використовуючи JWT. В ідеалі шлюз перевірив би маркер і передав би запит функції, якщо маркер дійсний.

2.2.2 Мікросервісна архітектура веб додатків

Архітектурний стиль мікросервісів - це підхід, при якому єдине додаток будується як набір невеликих сервісів, кожен з яких працює у власному процесі і коммуницирует з іншими використовуючи легковагі механізми, як правило HTTP.

Ці сервіси побудовані навколо бізнес-потреб і розгортаються незалежно з використанням повністю автоматизованої середовища. Існує абсолютний мінімум централізованого управління цими сервісами. Самі по собі ці сервіси можуть бути написані на різних мовах і використовувати різні технології зберігання даних.

Розбиття через сервіси

Архітектура мікро сервісів використовує бібліотеки, але їх основний спосіб розбиття додатку - шляхом ділення його на сервіси. Ми визначаємо бібліотеки як компоненти, які підключаються до програми і викликаються нею в тому ж процесі, в той час як сервіси - це компоненти, що виконуються в окремому процесі і спілкуються між собою за допомогою HTTP.

2.2.3 Монолітна архітектура

Монолітний сервер - досить очевидний спосіб побудови подібних систем. Вся логіка по обробці запитів виконується в єдиному процесі, при цьому ви можете скористатися наявними можливостями вашого мови програмування для поділу додатки на класи, функції і namespace-и. Ви можете запускати і тестувати додаток на машині розробника і використовувати стандартний процес розгортання для перевірки змін перед публікацією їх на робочі сервери. Ви можете масштабувати монолітні додатки горизонтальним шляхом за допомогою запуску декількох фізичних серверів та балансувальника навантаження.

Монолітні додатки можуть бути успішними, але все більше людей розчаровуються в них, особливо в світлі того, що все більше додатків розгортаються в хмарі. Будь-які зміни, навіть самі невеликі, вимагають перебудови і розгортання всього моноліту. З плином часу, стає важче зберігати хорошу модульну структуру, зміни логіки одного модуля мають тенденцію впливати на код інших модулів. Масштабувати доводиться все додаток цілком, навіть якщо це потрібно тільки для одного модуля цього додатка.

2.3 Порівняльний аналіз FaaS та мікросерервісних підходів

FaaS проти сервісних систем

Без сервісні системи та функції як сервісн (FaaS) часто поєднуються між собою, але правда полягає в тому, що FaaS насправді є підмножиною без серверних рішень. Без серверний підхід орієнтовано на будь-які категорії послуг, будь то обчислення, зберігання, бази даних, обмін повідомленнями, арі-шлюзи тощо, де конфігурація, управління та виставлення рахунків серверів непомітні для кінцевого користувача. FaaS, з іншого боку, хоча є, мабуть, найбільш центральною технологією в архітектурах без сервера, орієнтований на керовану подіями обчислювальну парадигму, в якій код програми або контейнери працюють лише у відповідь на події або запити.

Переваги FaaS

FaaS - це цінний інструмент, якщо ви хочете ефективно та економічно перенести програми в хмару. Нижче наведено деякі переваги:

1. Можливо більше зосередитися на коді, а не на інфраструктурі. За допомогою FaaS ви можете розділити сервер на функції, які можна масштабувати автоматично та незалежно, щоб вам не

довелося керувати інфраструктурою. Це дозволяє зосередитись на коді програми та може значно скоротити час виходу на ринок.

2. Сплачуйте лише за ті ресурси, якими ви користуєтесь, коли ви їх використовуєте: За допомогою FaaS ви платите лише тоді, коли відбувається дія. Коли запит виконано, все припиняється - не запускається код, не працює холостий сервер, не виникає витрат. Таким чином, FaaS є рентабельним, особливо для динамічних навантажень або запланованих завдань. FaaS також пропонує чудову загальну вартість власності для сценаріїв з високим навантаженням.
3. Автоматичне масштабування: За допомогою FaaS функції масштабуватися автоматично, незалежно та миттєво, за необхідності. Коли попит падає, FaaS автоматично зменшує масштаб назад.
4. FaaS пропонує притаманну високу доступність, оскільки вона розповсюджена в декількох зонах доступності в географічному регіоні і може бути розміщена в будь-якій кількості регіонів без додаткових витрат.

2.4 Проектування програмного продукту

Відповідно до описаних технологій, бібліотек та архітектурних підходів буде розроблена програмна база для порівняльного аналізу різних підходів в побудові веб додатків. Та для використання методу методу оптимізації на реальному прикладі.

Процес розробки поділяється на декілька етапів. По перше, буде розроблено програмний додаток, які базуються на монолітній архітектурі. Додаток буде побудована за допомогою програмного рішення Spring.

Наступний крок побудова системи яка базується на мікро-сервісній архітектурі. Яка теж має в основі програмне рішення Spring.

Далі буде побудовано систему яка буде виконувати оптимізацію програмного коду. Система складатиметься з двох частин — система для аналізу програмного коду та система для генерації програмного коду.

2.4.1 Проектування мікро сервісної системи

В даному підрозділі надана інформація про проектування та розробка програмного рішення яке реалізоване за допомогою микро сервісного підходу до побудови додатків.

Система складається з двох тестових проектів які базуються на основі фреймворку Spring.

Визначення контролера

DispatcherServlet відправляє запит контролерам для виконання певних функцій. Анотація @Controllek вказує, що конкретний клас є контролером. Анотація @RequestMapping використовується для мапінгу (зв'язування) з URL для всього класу або для конкретного методу обробника.

Анотація Controller визначає клас як Контролер Spring MVC. У першому випадку, @RequestMapping вказує, що всі методи в даному контролери відносяться до заданого URL-адресу. Наступна анотація @RequestMapping (method = RequestMethod.GET) використовується для оголошення методу.

Атрибут «value» вказує URL, з яким пов'язаний цей метод. Далі вказується, що метод буде обробляти GET-запити (method = RequestMethod.GET). Також, потрібно відзначити важливі моменти щодо наведеного вище контролера:

1. Бізнес-логіка визначається всередині пов'язаного таким чином службового методу.
2. З тіла цього службового методу можна викликати будь-які інші методи.
3. Ґрунтуючись на заданій бізнес-логікою, в рамках цього методу створюєте Модель (Model).
4. Модель може бути модифікована за допомогою атрибутів, які, в свою чергу, будуть додані до Виду (View).

Використання Spring анотацій

За замовчанням Spring використовую анотацію - @SpringBootApplication. Деякі анотацій додаються автоматично, якщо збережена рекомендована структура Spring проекту. Наприклад - якщо присутня бібліотека spring-webmvc, то анотація @EnableWebMvc додається автоматично з автоматичною конфігурацією за замовчуванням.

Ця конфігурація все ще може бути модифікованою та до неї може бути додана функціональність MVC паттерн за допомогою інтерфейсу WebMvcConfigurer добавленому до класу з а анотацією @Configuration.

Для поглибленого налаштування мережевих параметрів та веб функції необхідно використати екземпляри класу WebMvcRegistrationsAdapter. Реалізація цього інтерфейсу дозволить переоприділити такі методи як - RequestMappingHandlerMapping, RequestMappingHandlerAdapter або ExceptionHandlerExceptionResolver.

Також веб MVC функції можуть бути деактивовані за допомогою анотації @EnableWebMvc, що дозволить провести власну конфігурацію.

Репозиторії

Репозиторії - це кілька інтерфейсів які використовують JPA Entity для взаємодії з нею. Так, наприклад, інтерфейс public interface CrudRepository <T, ID extends Serializable> extends Repository <T, ID>забезпечує основні операції з пошуку, збереження, видалення даних (CRUD операції) та інші.

Також є абстракція PagingAndSortingRepository. Якщо того переліку, що надає інтерфейс достатньо для взаємодії з сутністю, то можно розширити базовий інтерфейс для своєї сутності, доповнити його своїми методами запитів та виконувати операції.

Для використання репозитаріїв потрібно створити класс сутності. Класс для сутності описує таблицю з бази даних. Поля об'єкту мають співпадати з полями таблиці. Кожна сутність має анотацію - @Entity, та анотацію @Table, якщо в базі даних присутня таблиця яку описує цю сутність.

Запити до сутності можуть будуватися двома способами:

1. За допомогою імені поля в сутності прямо в методі запита. Для побудови використовуються допоміжні слов, такі як - By, And, Or, find, query, count та інші.
2. За допомогою анотації Query. В тілі анотацій можна задати параметр SQL query

Бізнес логіка веб додатку

Бізнес-логіка - сукупність правил, принципів, залежностей поведінки об'єктів предметної області (сфери людської діяльності, яку система підтримує). Інакше можна сказати, що бізнес-логіка - це реалізація правил і обмежень операцій які автоматизуються. Є синонімом терміна «логіка предметної області». Бізнес-логіка задає правила, яким підкоряються дані предметної області.

Бізнес-логіка Spring додатку, знаходиться між репозиторієм та контролером. Клас з бізнес логікою, зазвичай, має анотацію - @Service.

@Service - (Сервіс-шар додатки) Анотація, що оголошує, що цей клас є сервіс - компонент сервіс-шару. Сервіс є підтипом класу @Component. Використання цієї інструкції дозволить шукати біни-сервіси автоматично.

Налаштування веб додатку

Налаштування та зміна функціональності додатку, через файл application.properties.

Файл application.properties дозволяє налаштувати велику кількість параметрів Spring додатку.

Для налаштування додатку Spring, необхідно створити конфігураційний файл в директорії ресурсів maven проекту.

Інтеграція конфігураційного файлу виконується за допомогою анотації @PropertSource

Тестування програмного додатку який базується на бібліотеці Spring

1. **Тестування контексту Spring додатків**

Починаючи з версії 3.1, бібліотеки Spring список можливостей для тестування збільшився. Додана підтримка тестування для класів з анотацією @Configuration: за допомогою використання анотації @RunWith (SpringJUnit4ClassRunner.class).

Визначення конфігураційних класів виконується Java за допомогою анотації @ContextConfiguration. Новий AnnotationConfigContextLoader завантажує визначення Bean об'єктів з класів @Configuration.

Тестування классу WebConfig не передбачено в програмному пакеті доданому до Spring бібліотеки.

2. **Тестування Spring додатку**

Spring Boot надає кілька приміток, для налаштувати Spring ApplicationContext при написанні тестових методів.

Для тестування використовується лише певний фрагмент конфігурації програми або моделюється весь процес запуску контексту.

Наприклад, для тестування створення всього контектсу, без запуску додатку, використовується анотація @SpringBootTest. Після конфігурації та моделювання повного контексту додатку необхідно додати @AutoConfigureMockMvc, який створить екземпляр классу MockMvc для тестування HTTP-запити.

Для уникнення створення всього контексту та перевірки лише обраних контролерів необхідно використовувати анотацію @WebMvcTest

Реалізація контролера не є загальнодоступною - це викликане відсутністю необхідності. Зазвичай, контролер є останнім у ланцюзі залежностей. Він отримує HTTP-запити від головного контролера Spring (DispatcherServlet) і делегує їх до рівня обслуговування.

Додаткові Maven залежності

На додаток до залежності Spring web mvc, необхідної для стандартного веб-додатка, необхідно встановити маршовий обмін змістом та видалити його для REST

Це бібліотеки, які використовуються для перетворення представлення ресурсу REST у JSON або XML.

2.4.2 Проектування без сервісного додатку

Налаштування функції Spring Cloud

Для побудови веб додатку з підтримкою технології Faas необхідно згенерувати проект з включенням бібліотеки spring cloud function web (spring-cloud-starter-function-web). Ця бібліотека включає всі необхідні залежності для коректного функціонування додатку.

Для локальної розробки цієї залежності достатньо, але її необхідно модифікувати перед етапи встановлення додатку до хмарних сервісів. (AWS, Google Cloud)

Створення головного классу додатку

Головний класс додатку має анотацію @SpringBootApplication, як і всі додатки побудовані за допомогою фреймворку Spring.

В головному класі створюється новий Bean об'єкт. Це тестова функція. За своєю побудовою нагадую звичайний метод. Має анотацію @Bean. Та повертає об'єкт Function<>.

Текстова функція виконує підрахунок значення факторіалу. Як параметр функція приймає значення для якого має бути підраховано факторіал.

Сканування хмарної функції Spring

Якщо використання та ініціалізація функції як @Bean не є прийнятною може бути створено клас, які буде імплементувати інтерфейс Function, в якому є можливість реалізувати метод для підрахунку факторіалу.

Місцезнаходження класу з функцією має бути визначено в конфігураційному файлі application.properties за допомогою встановлення параметру spring.cloud.function.scan.package

ВИСНОВКИ ДО РОЗДІЛУ 2

У даному розділі було розглянуто бібліотеки, технології та програмні рішення, що будуть використовуватися р цій магістерській дисертації та будуть використані в наступному розділі.

Для взаємодії між компонентами системи буде використовуватися підхід REST API та WebSocket. Це два найрозповсюдженіших способи для передачі даних між додатками. Ці два підходи підтримують увесь необхідний функціонал для реалізації запланованої системи.

Дані для аналізу і генерації додаток буде прати з вихідного коду Java. Для зчитування та збереження окремих рішень будуть використані окремі бібліотеки.

Буде згенеровано ряд додатків для порівняльного аналізу швидкості роботи системи. Усі системи будуть виконувати одні і ті ж самі обчислення для достовірності порівняння.

Проблема продуктивності роботи системи для аналізу, оптимізації та генерації є досить великою. Генерація великої кількості тексту складна задача. І тема поліпшення процесу оптимізації, аналізу та генерації може бути використана в якості подальшої теми дисертації для продовження наукової діяльності автора.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

Відповідно до засобів та моделей, спроектованих та описаних у другому розділі магістерської дисертації, розробити програмну реалізацію системи програмної оптимізації веб додатків та веб систем.

Розробка буде виконана за допомогою мови Java.

3.1 Розробка мікро сервісної системи

Ця система відповідає за тестування швидкості роботи системи яка розроблена за допомогою мікро-сервісної архітектури.

Система складається з двох додатків, які були написані на мові програмування Java з використанням бібліотеки Spring. Додатки відрізняються лише функціями функціями які вони обчислюють.

Перший додаток розраховує факторіал вказаного значення. Для побудови додатку використовувалася REST підхід.

Для отримання факторіалу необхідно відправити GET запит на адресу - http://адреса за якою розміщенно додаток/calculations/factorial?value={значення для якого буде підраховано факторіал}.

Другий додаток розраховує логарифм вказаного значення. Для побудови додатку використовувалася REST підхід.

Для отримання значення логарифму необхідно відправити GET запит на адресу - http://адреса за якою розміщено додаток/calculations/log?value={значення для якого буде знайдено логарифм}.

3.2 Розробка монолітної системи

Ця система відповідає за тестування швидкості роботи системи яка реалізован за допомогою монолітної архітектури.

Додаток побудований на основі монолітної архітектури обраховує значення логарифму та знаходить факторіал.

Були використані ті ж шляхи та імена параметрів, що і для мікро сервісної системи.

3.3. Розробка системи яка базується на технології FaaS

Ця система відповідає за тестування швидкості роботи системи яка реалізує підхід функція як сервіс.

3.4 Розробка системи для аналізу коду програми

Система відповідає за зчитування вихідного коду програми.

Система базується на зчитуванні вихідного коду які представляється як звичайний текст.

Операції, які виконує додаток:

1. Завантаження файлів проекту та аналіз файлів java
2. Аналіз наповнення класів
3. Побудова залежностей між різними рівнями додатку
4. Аналіз конфігураційних файлів
5. Аналіз файлів Docker

Алгоритм роботи функцій додатку

Алгоритм аналіз класу

За допомогою тексту

Алгоритм:

1. Вхідний параметр - шлях до файлу
2. Зчитування файлу
3. Зчитування кожної строки файлу
4. Збереження зчитаної строки в об'єкт StringBuilder
5. Закінчення циклу з читанням та записом
6. Повернення файлу у вигляді строки
7. Виклик класу Launcher

8. Виклик методу parseClass() з класу Launcher, використовуючи отриманий об'єкт типу StringBuilder
9. Створення об'єкта типу CtModel та назвою model
10. Присвоєння об'єкту model значення отриманого методом parseClass()

За допомогою об'єкту

1. Вхідні дані - шлях до файлу
2. Створити об'єкт launcher типу Launcher
3. Присвоєння об'єкту launcher значення addInputResource() зі шляхом до файлу як параметром

4. Викликати метод buildModel() у об'єкту launcher
5. Створення об'єкта типу CtModel та назвоб model
6. Присвоєння об'єкту model значення отриманого методом getModel()

3.5 Розробка системи для генерації програмного коду оптимізованої програми

Система відповідає за генерацією програмного коду на мові програмування Java.

Алгоритм роботи функцій додатку

Побудова контролеру веб додатку

Контролер відповідає за спілкування з іншими веб додатками та користувачем. Кожна кінцева точка відповідає за конкретну операцію та виконує закріплену в ній бізнес логіку.

Етапи побудови контролеру

1. В кореневому каталозі проекту створюється пакет під назвою web.controller.
2. В каталозі web.controller створюється Java клас під назвою MainController
3. До класу додається анотація @Controller та @RequestMapping
4. В тілі класу створюється методи для отримання даних з заданого проекту.

Нижче наведено покроковий приклад побудови метода, який буде виконувати функцію кінцевої точки та поєднувати веб додаток та зовнішні інтерфейси:

1. Створюються метод з назвою яка описує операцію, котру виконуватиме даний метод
2. Додається анотація яка уточнює операцію, котру виконуватиме ця кінцева точка (@GetMapping, @PutMapping, @PostMapping, @DeleteMapping)
3. В параметрах анотації вказується шлях, за яким кінцева точка буде доступна
4. В тілі методу викликається екземпляр сервісу в якому реалізована бізнес логіка яка має виконуватися для цієї кінцевої точки.

Побудова сервісу веб додатку:

1. В кореневому каталозі створюється пакет - service
2. В пакеті service створюється під директора impl
3. В пакеті service створюються інтерфейси майбутніх сервісів.
4. Інтерфейси з пакету service реалізуються в директорії impl
5. Сервіси з директорії impl мають анотації @Service та містять усю бізнес логіку додатку

Побудова репозиторію веб додатку

1. В кореневому каталозі створюється пакет - repository
2. В пакеті repository створюється класс Repository, котрий наслідує CrudRepostory<?,?> в якому вже реалізовані CRUD операції

Опис роботи кінцевих точок веб додатку

1. GET /template

   Побудова основи для веб додатку виконується за допомогою Spring Initialize API. Усі параметри повністю мігрують з офіційного сайту та офіційної документації.

2. POST/function

   Кінцева точка яка починає процес побудови окремого веб додатку базуючись на фреймворку Spring з використанням бібліотек, що підтримують хмарні обчислення.

Алгоритм роботи бізнес логіки яка активізується при звернені до кінцевої точки:

1. Перевірка існування проекту за вказаним шляхом
2. Перевірка існування контролеру зі вказаною назвою
3. Генерація основи веб додатку з додаванням бібліотек для ініціалізації хмарних обчислень
4. Аналіз вказаної кінцевої точки
5. Деконструкція кінцевої точки
6. Аналіз бізнес логіки необхідної для виконання завдання кінцевої точки
7. Створення об'єкта типу FunctionProjectTemplate
8. Визначення всіх полів об'єкта типу FunctionProjectTemplate

9. Імплементація усіх елементів FunctionProjectTemplate в згенеровано основу.
10. Інтеграція нового проекту в систему.
11. Визначення параметрів проекту

3. POST /project/endpoints

Кінцева точка починає процес побудови проекта який відокремлюється від головного.

Алгоритм роботи бізнес логіки яка активізується при звернені до кінцевої точки:

1. Перевірка існування проекту за вказаним шляхом
2. Перевірка існування контролеру зі вказаною назвою
3. Генерація основи веб додатку з додаванням бібліотек для побудови веб додатків Spring
4. Перевірка існування кожної вказаної кінцевої точки
5. Створення об'єкта типу RestProjectTemplate
6. Визначення всіх полів об'єкта типу RestProjectTemplate
7. Імплементація усіх елементів RestProjectTemplate в згенеровано основу.
8. Інтеграція нового проекту в систему
9. Визначення параметрів проекту

3.6 Аналіз ефективності оптимізованої системи

Для отримання найкращих результатів від оптимізації та максимально збільшити продуктивність необхідно оптимізовувати великі програмні рішення. З аналізу часу на побудову контексту можно зробити висновок, що FaaS, є найлегшим рішенням з усіх представлених.

```
Started ServerlessApplication in 2.016 seconds (JVM running for 3.007)
```

Рис 3.1 час на запуск без серверного додатку

Навіть враховуючи основні переваги безсерверних обчислень (користувачі платять лише за ресурси, які використовуються лише тоді, коли функція виконується), провайдери хмарних платформ вводять обмеження на доступні обчислювальні ресурси для без серверних систем - обмежують максимальну кількість одночасних запитів клієнта та максимальний ресурс пам'яті та центрального процессора, доступні для кожного виклику функції.

```
Started MonolitApplication in 4.635 seconds (JVM running for 7.544)
```

Рис 3.2 час на запуск без серверного додатку

Без серверний підхід є найпростішим в побудуванні та найлегшим з усіх рішень. Але дозволяє отримати збільшення продуктивності тільки для додатків, які мають велику кількість функцій які виконуються періодично і не потребують постійного запуску.

Обчислення, які виконуються за допомогою одних і тих же самих бібліотек не дають значного виграшу в процесі передачі і розрахунку.

Головна задача - оптимізація використання обчислювальних ресурсів, може бути виконана шляхом переносу частини функціоналу на інші платформи (в тому числі хмарні), але це не вирішує проблему цілком.

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі було виконано розробку програмної частини системи для оптимізації програмних рішень та було наведено приклад використання методу для оптимізації обчислювальних ресурсів за допомогою хмарних технологій.

За результатами розробки було отримано підсистеми для аналізу, оптимізації та генерації програмних додатків.

Підсистема аналізу вихідного коду виконує процес зчитування програмного коду та його подальше розбиття на складові частини. Складові системи, отримані після аналізу, подається в додаток для подальшого використання в процесі оптимізації.

Підсистема оптимізації працює по одному з обраних сценарії. Генерує додаток базуючись на одному з обраних архітектурних підходах.

Підсистема генерації реалізує процес створення програмного коду. Генерує основу веб додатку для використання бібліотеки Spring.

ВИСНОВКИ

Зміст магістерської дисертації присвячений розробці методу для оптимізації обчислювальних ресурсів.

У першому розділі було досліджено базові методи та алгоритми для оптимізації обчислювальних ресурсів. Проаналізовані різні підходи до оптимізації. Оптимізація програмного коду, налаштування залежностей, видалення дублікатів коду, обновлення програмної бази, міграція на новіщі рішення. Розглянута оптимізація контейнерів, які використовуються як середа для запуску додатків. Проаналізовані методи з оптимізації фізичних ресурсів, налаштування кластерних систем.

У другому розділі було виконано огляд засобів для реалізації системи для оптимізації. Бібліотека Java Poet виконує процеси пов'язані з генерацією коду на мові Java, бібліотека Spring надає необхідну програмну базу для моделювання різних систем, які використовуються для тестування ефективності методу оптимізації.

Третій розділ присвячено розробці програмного продукту та аналізу ефективності розробленого методу оптимізації. За результатами цього розділу біло отримано підсистеми для аналізу, оптимізації та генерації програмного коду.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.Koschke E. Survey of research on software clones. In Duplication, Redundancy, and Similarity in Software / Koschke E., Merlo, A. Walenstein. - Internationales Begegnungs-und Forschungszentrum f¨ur Informatik (IBFI), Schloss Dagstuhl, Germany - 2008.

2. Ducasse, S., Rieger, M., and Demeyer, S. A language independent approach for detecting duplicated code. In Proceedings IEEE International Conference on Software Maintenance -'Software Maintenance for Business Change - 1999, C. 109–118.

3. Baker, B. S. On finding duplication and near-duplication in large software systems. InProceedings of 2nd Working Conference on Reverse Engineering - 1995, C. 86–95.

4. Umberto Azadi, Dott. Bartosz Walter Automation of duplicate code detection and refactoring, Anno Accademico, Università degli Studi di Milano Bicocca - 2019

5. Fowler, M. Refactoring: Improving the Design of Existing Code. Addison-Wesley - 1999.

6. Stelios Sidiroglou-Douskos, Eric Lahtinen, Anthony Eden, Fan Long, Martin Rinard CodeCarbonCopy, MIT EECS and MIT CSAIL - 2017.

7. Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. Compilers, Principles, Techniques. Addison Wesley - 1986 C. 12-13

8. M. Hall, D. Padua, and K. Pingali. Compiler research: The next 50 years. Commun. ACM / M. Hall, D. Padua, and K. Pingali - 2009. - [Електронний ресурс] - Режим доступу: http://dl.acm.org/citation.cfm?id=1461946.

9. Болодурина И. П., Парфенов Д. И. "Алгоритмы комплексной оптимизации потребления вычислительных ресурсов в облачной системе дистанционного обучения" / Болодурина И. П., Парфенов Д. И// Вестник ОГУ №9 - С. 158-165.

10. В. А. Богатырев, А.В. Богатырев, И.Ю. Голубев, С.В. Богатырев "Оптимизация распределения запросов между кластерами отказоустойчивой вычислительной системы" В. А. Богатырев, А.В. Богатырев, И.Ю. Голубев, С.В. Богатырев (Научно-технический вестник информационных технологий, механики и оптики, №3 - 2019. - С. - 89-84

11. Bo Liu, Pengfei Li, Weiwei Lin, Na Shu, Yin Li, Victor Chang - A new container scheduling algorithm based on multi-objective optimization [Електронний ресурс] - Режим доступу - https://www.researchgate.net/publication/326574604_A_new_container_scheduling_algorithm_based_on_multi-objective_optimization - 2017

12. Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. IEEE computational intelligence magazine. - 2006. С. - 28–39

13. Shu-qiang Huang, Huan-ming Zhang - Research on Improved MVC Design Pattern Based on Struts and XSL International Symposium on Information Science and Engineering - 2008

14. Mathias Kuhneman, Thomas Rauber, Gudula Runger A Source Code Analyzer for Performance Prediction. 18th International Parallel and Distributed Processing Symposium. - 2004.

15. Michel Brassard, COMPONENT-BASED SOURCE CODEGENERATOR [Електронний ресурс] - Режим доступу: https://patentimages.storage.googleapis.com/24/c8/32/8154c638e1cd21/US6742175.pdf) - 2000.

16. R. Fielding, T. Berners-Lee Hypertext Transfer Protocol [Електронний ресурс] - Режим доступу: [https://tools.ietf.org/html/rfc1945] - 2018

17. Уоллс К. С Spring в действии М:ДМК Пресс, 2013 Уоллс К. С 32

18. Haines, Steven. Serverless computing with AWS Lambda, Part 1 (англ.), JavaWorld. - 2018. С.- 34