УДК 004.75:004.42

МІКРОСЕРВІСНА КОНТЕЙНЕРНА АРХІТЕКТУРА СИСТЕМИ КЕРУВАННЯ ПОТОКАМИ ДАНИХ

DOI 10.36994/2788-5518-2021-02-02-17

Булах Б.В., *к.т.н. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна, bogdan.bulakh@gmail.com*

Харченко К.В., *к.т.н. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна, konst1970@gmail.com*

Анотація. *Розглядається архітектура та тестова реалізація мікросервісної архітектури з використанням контейнерів та REST протоколу взаємодії між компонентами системи керування потоками даних. Таке рішення дозволяє будувати прості та ефективні системи керування потоками даних на базі веб-протоколу з REST та JSON передачею параметрів. Тестова реалізація на мові Python показує можливість швидкої та надійної побудови практичної системи з керуванням потоками даних. Контейнерна архітектура дозволяє запускати мікросервіси системи керування потоками даних як на одному сервері, так і у різних, включно з хмарними сервісами.*

Ключові слова: *парадигма керування потоками даних, мікросервіс, контейнер, REST, JSON, Python.*

MICROSERVICE CONTAINER ARCHITECTURE OF DATA FLOW COMPUTATION SYSTE

Bogdan Bulakh, *Ph.D. National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, bogdan.bulakh@gmail.com*

Kostyantyn Kharchenko, *Ph.D. National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, konst1970@gmail.com*

Abstract. *The architecture and test implementation of the microservice architecture using containers and the REST protocol of interaction between the components of the data flow computation system is considered. This solution makes it possible to build simple and efficient web-based flow control systems with REST and JSON parameter transfer. The Python test implementation demonstrates the ability to quickly and reliably build a practical data flow computation system. Container architecture allows running microservices of the data flow computation system on one server as well as on different servers, including cloud services.*

The data flow computation paradigm is briefly mentioned, some basic software tools and technologies needed to build demo data flow application, including Flask Python micro framework, REST protocol, Requests library, Docker container engine are briefly described. Generic architecture of the multi-node data flow system based on containerized microservices is presented. The novelty of this approach lies in the using of both microservices and containers to deploy each single data consumer or producer as a service, that should be easy to deploy and flexible to configure without using specialized heavyweight data dispatching software or specific data flow description languages. The basic setup and configuration steps, as well as code listings are included. Basic execution steps are commented, including command-line interface commands to be executed. The demo data flow system described is simple enough to fit in the single article's size and show the main benefits of the proposed idea, but it also has some drawbacks, including: hardcoded ports dependencies, no special input for each microservice entity, insecure connection between microservices. These issues can be easily fixed, and will be addressed as a part of the future work. Other directions of the further work include: research of the complex data flow scenarios implemented on top of the proposed architecture, study of the microservices orchestration (at both service level and container level), complex multi-request interaction cases.

Keywords: *data flow computation paradigm, microservice, container, REST, JSON, Python.*

Вступ

Сучасні вимоги до побудови обчислювальних систем потребують застосування різноманітних підходів до архітектури. Це включає також застосування як парадигми керування потоками даних, так і парадигми керування потоками команд. Розвиток мікросервісної архітектури та контейнеризації дає широкі можливості для побудови практичних інформаційних систем з парадигмою керування потоками даних. Узагальнений та універсальний протокол обміну даними на базі REST API з форматом JSON дає можливість легко організовувати обмін даними між складовими частинами системи керування потоками даних. Розглянемо один із варіантів побудови архітектури такої системи та її реалізацію за допомогою сучасних засобів розробки веб-орієнтованого програмного забезпечення.

Парадигма керування потоками даних це парадигма програмування, яка моделює програми, як спрямовані графіки даних, що протікають між операціями (на відміну від парадигми керування потоками команд). Такий підхід відомий з 1960-х років і набув поширення у певних колах для опису процедури обчислень. Практичного застосування парадигма керування потоками даних набула у системах обробки цифрових даних, опису бізнес процесів, побудові керування системами інтернету речей, тощо [1,2,5-7].

Новизна даної роботи полягає у застосуванні мікросервісної архітектури та конейнеризації для побудови окремих складових системи керування потоками даних на базі веб-технологій з використанням HTTP протоколу і формату обміну даними JSON.

Мікросервіси - це техніка розробки програмного забезпечення, або варіант структурного стилю, орієнтований на сервісну архітектуру (SOA), який організовує додаток як сукупність слабко пов'язаних служб. В архітектурі мікросервісів послуги дрібнозернисті, а протоколи вза'ємодії - легкі.

Мікросервіс не є шаром у монолітній програмі (наприклад, веб-контролер або резервний інтерфейс). Скоріше це самодостатня частина функціоналу бізнесу з чіткими інтерфейсами, і може, завдяки власним внутрішнім компонентам, реалізувати шарувату архітектуру. З точки зору стратегії архітектура мікропослуг по суті відповідає філософії Unix "Роби щось одне і роби це добре".

Representational state transfer (REST) - це архітектурний стиль програмного забезпечення, який визначає набір обмежень, які будуть використані для створення веб-служб. Веб-сервіси, що відповідають архітектурному стилю REST, називаються послугами RESTful Web, забезпечують сумісність між комп'ютерними системами в Інтернеті. Веб-сервіси RESTful дозволяють запитуючим системам отримувати доступ та маніпулювати текстовими поданнями веб-ресурсів за допомогою рівномірного та заздалегідь визначеного набору операцій без стану.

Об'єктна нотація JavaScript (JSON) - це формат файлів відкритого стандарту або формат обміну даними, який використовує читаний людиною текст для передачі об'єктів даних, що складаються з пар атрибутів і значень та типів даних масиву (або будь-яке інше значення, яке може бути серіалізується). Це дуже поширений формат даних з різноманітним діапазоном застосувань, таких як заміна XML в системах AJAX.

Flask - це мікрофреймворк, написаний на Python. Він названий мікрофреймворком, оскільки не потребує конкретних інструментів чи бібліотек. У нього немає шару абстрагування бази даних, засобів перевірки форм або будь-яких інших компонентів, де раніше існуючі сторонні бібліотеки забезпечують загальні функції. Однак Flask підтримує розширення, які можуть додавати функції програми так, ніби вони були реалізовані в самій Flask. Розширення існують для об'єктно-реляційних відображень, перевірки форм, обробки завантажень, різних технологій відкритої автентифікації та декількох загальних інструментів. Розширення оновлюються набагато частіше, ніж основна програма Flask.

Бібліотека requests - це бібліотека HTTP для Python, випущена за ліцензією Apache2. Мета цього проекту - зробити запити HTTP простішими та зручнішими для людини. Поточна версія, яку використано у тестовому проекті є 2.22.0.

Контейнер - це стандартний блок програмного забезпечення, який пакує код та всі його залежності, тому програма швидко та надійно працює з одного обчислювального середовища в інше. Зображення контейнера Docker - це легкий, автономний, виконуваний пакет програмного забезпечення, який включає все необхідне для запуску програми: код, виконувані файли, системні інструменти, системні бібліотеки та налаштування.

Óбрази контейнерів стають контейнерами під час виконання, а у випадку контейнерів Docker - образи стають контейнерами, коли вони запускаються в Docker Engine. Доступне як для ОС Linux (так і для MacOS), Windows, контейнерне програмне забезпечення завжди працюватиме однаково, незалежно від інфраструктури. Контейнери ізолюють програмне забезпечення від свого оточення та забезпечують сталість його роботи, незважаючи на те, чи це етап розробки, чи випуску готової версії програмного забезпечення .

Подібний підхід використовується наприклад у Spring Cloud Data Flow [3], яка об'єднує обробку потоків та пакетних даних для мікросервісів даних через хмару чи на власних серверах. Це дозволяє розробникам створювати, упорядковувати та проводити рефакторинг потоків даних за допомогою єдиної моделі програмування для випадків загального використання, таких як введення даних, аналітика в режимі реального часу та імпорт/експорт даних. Реалізація відповідного рішення виконана на мові програмування Java.

Apache Beam SDK [4] також надає можливості працювати з потоками даних. Apache Beam - це відкрита, уніфікована модель для визначення як пакетної, так і потокової передачі даних паралельних потоків даних. Використовуючи один з SDK-дистрибутивів з відкритим кодом, ви створюєте програму, яка визначає конвеєр. Потім конвеєр виконується одним із підтримуваних Beam-процесорів, що підтримуються, включаючи Apache Apex, Apache Flink, Apache Spark та Google Cloud Dataflow.

Beam є особливо корисним для завдань із незручною паралельною обробкою даних, в яких задача може бути розкладена на безліч менших пакетів даних, які можна обробляти незалежно та паралельно. Ви також можете використовувати Beam для витягування, перетворення та завантаження завдань та чистої інтеграції даних (ETL). Ці завдання корисні для переміщення даних між різними носіями даних та джерелами даних, перетворення даних у більш бажаний формат або для завантаження даних у нову систему.

Постановка задачі в даній роботі є створення архітектури та тестової реалізації мікросервісної архітектури системи керування потоками даних з REST/JSON протоколом передачі з використанням контейнерів. Метою створення тестового коду на мові програмування Python є ілюстрація практичного підтвердження можливості реалізації такої системи і її тестування на реальних контейнерах. Досліджується конфігурація, програмні засоби та бібліотеки розробки, необхідні для реалізації простої системи керування потоками даних.

Виконання досліджень

Архітектуру системи керування потоками даних з використанням мікросервісів можна описати як набір веб-серверів з API, що викликають один одного відповідно до конфігурації графа обчислень і передають дані за допомогою REST інтерфейсу у вигляді JSON файлів (рис. 1, рис.2).

Відповідно кожен мікросервіс можливо розмістити в окремому контейнері, а контейнери відповідно у різноманітних хмарних сервісах.

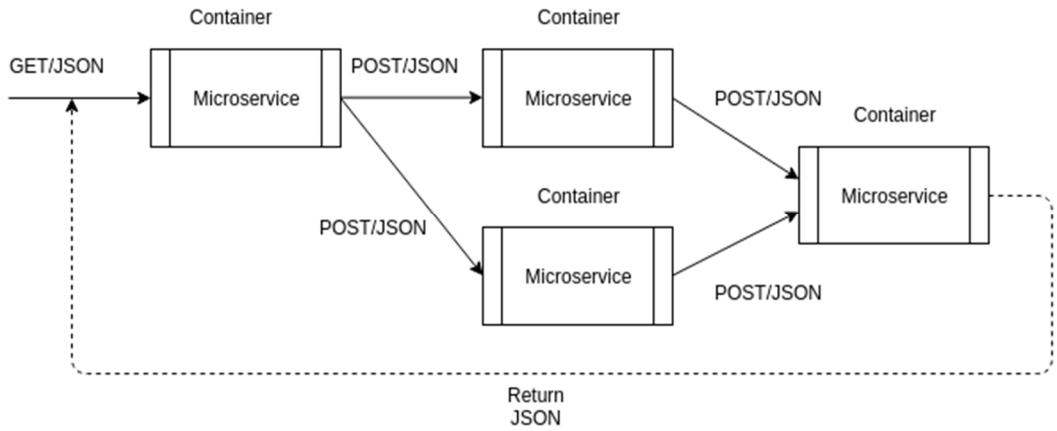

Рис. 1. Архітектура системи керування потоками даних з використанням мікросервісів та контейнерів.

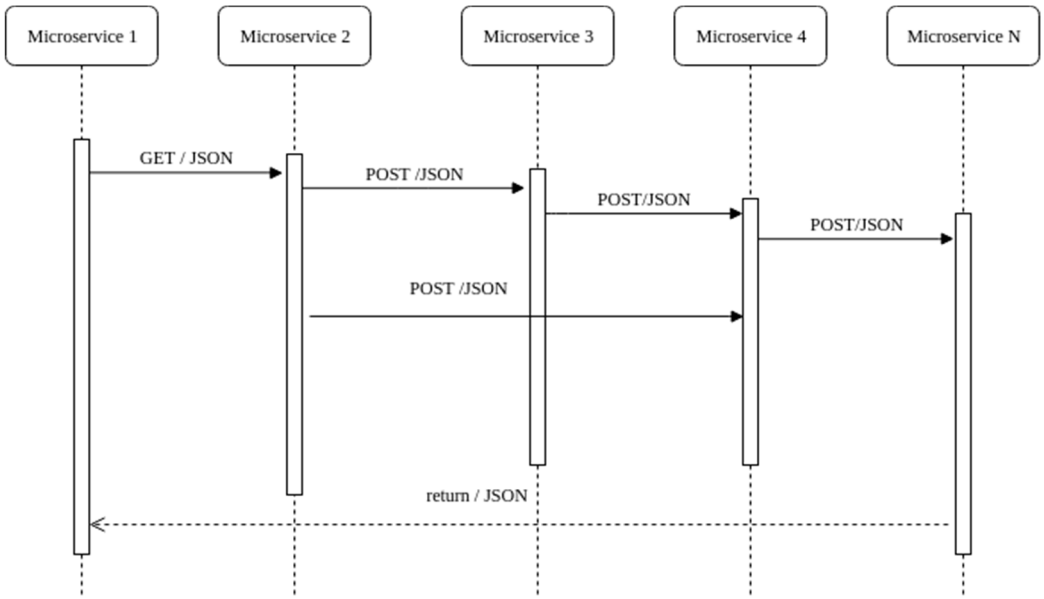

Рис. 2. Діаграма послідовностей керування потоками даних з використанням мікросервісів та контейнерів.

Для реалізації тестового прикладу системи керування потоками даних було обрано наступні безкоштовні засоби розробки та бібліотеки:

- Мова програмування Python версія 3
- Мікрофреймворк для веб-додатків Flask

- Бібліотека requests для формування запитів від мікросервіса до мікросервіса
- Система контейнеризації Docker

Гнучкість налаштування мікросервісів та забезпечується за допомогою параметрів налаштувань контейнерів у команді зовнішньої змінних оточення, в нашому випадку є адреса тестового комп'ютеру, яка представлена змінною DATAFLOW="http://192.168.1.85:5002/calculate".

Лістінг файлу dfserver.py

```
1.  from flask import Flask, escape, url_for, jsonify, request
2.  import requests
3.  import os
4.  app = Flask(__name__)
5.  @app.route('/')
6.  def index():
7.  return jsonify(name="microservice is ready")
8.  @app.route('/run')
9.  def run():
10. address = os.environ.get('DATAFLOW')
11. res = requests.post(address, json={"text1": "Hello ", "text2": "World"})
12. return jsonify(name=res.text)
13. @app.route('/calculate/', methods=['GET', 'POST'])
14. def calculate():
15. req_data = request.get_json()
16. w1 = req_data['text1']
17. w2 = req_data['text2']
18. return w1+w2
```

Контейнер для мікросервісу побудовано на базі збірки Python:3. Для конфігурації фреймворку Flask використовується команда pip3 install Flask. Для конфігурації бібліотеки requests використовується файл requirements.txt, який обробляється також командою pip3. Експерименти з іншими варіантами побудови контейнеру були неуспішними, або займали багато місця і вимагали багато часу для побудови. Приклад готової конфігурації контейнеру показано на рис.3.

Лістінг файлу Dockerfile

```
1.  FROM python:3
2.  COPY . /dfrest
3.  WORKDIR /dfrest
4.  RUN pip3 install Flask
5.  RUN pip install -r requirements.txt
6.  CMD flask --version
7.  ENV FLASK_APP "/dfrest/dfserver.py"
8.  CMD flask run --host=0.0.0.0 --port=5000
```

9. Лістінг файлу requirements.txt
10. requests==2.21.0

Для побудови docker container маємо таку команду:

```
> docker build -t dfrest:v0.1 dfrest/
```

Для запуску контейнерів маємо таку команду:

```
> docker run -d -p 5001:5000 -e DATAFLOW="http://192.168.1.85:5002/calculate" dfrest:v0.1
> docker run -d -p 5002:5000 -e DATAFLOW="http://192.168.1.85:5002/calculate" dfrest:v0.1
```

Але поточна реалізація системи керування потоками даних також має і свої певні недоліки, які необхідно буде виправити у наступних версіях, а саме:
відв’язати залежність від портів на клієнтському рівні, для цього знадобиться файл конфігурації системи з описом графу обчислень
передавти окремий текст для виконання у конкретний мікросервіс, це зробить систему універсальною
підвищити безпеку використання системи, застосувавши шифрований протокол обміну.

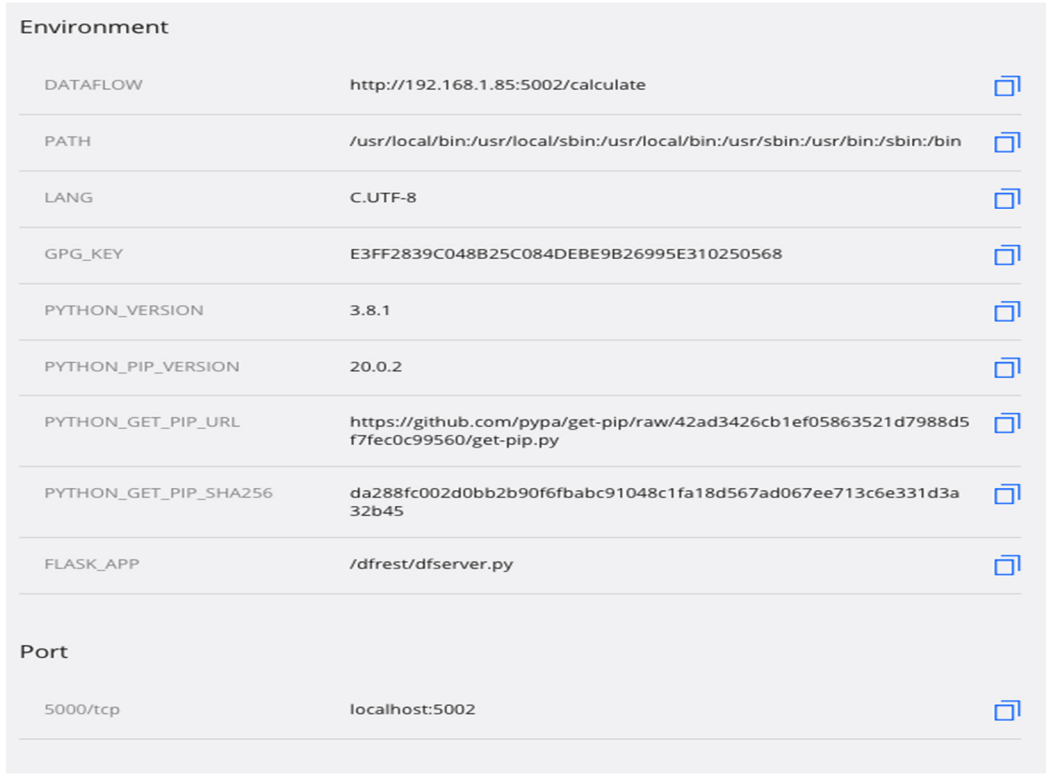

Рис. 3. Приклад конфігурації контейнеру для роботи мікросервісу у системі керування потоками даних.

Тестовий приклад для демонстрації роботи системи керування потоками даних

Що саме відбувається у тестовому прикладі системи керування потоками даних (рис.4):

1. Запускається веб-сервер Flask
2. Налаштовується метод запиту обробки /run для запуску системи керування потоками даних
3. За допомогою методу HTTP GET запускається метод /run для початку роботи системи керування потоками даних
4. Метод запиту обробки /run запускає інший інстанс мікросервісу за допомогою методу HTTP POST з методом обробки /calculate
5. Метод запиту обробки /calculate приймає параметри у вигляді JSON файлу, у нашому прикладі виконує об'єднання двох строкових даних в одну строку та повертає розраховане значення попередньому мікросервісу.

Ланцюжок обчислень задається відповідними налаштуваннями методів обробки запитів мікросервсів.

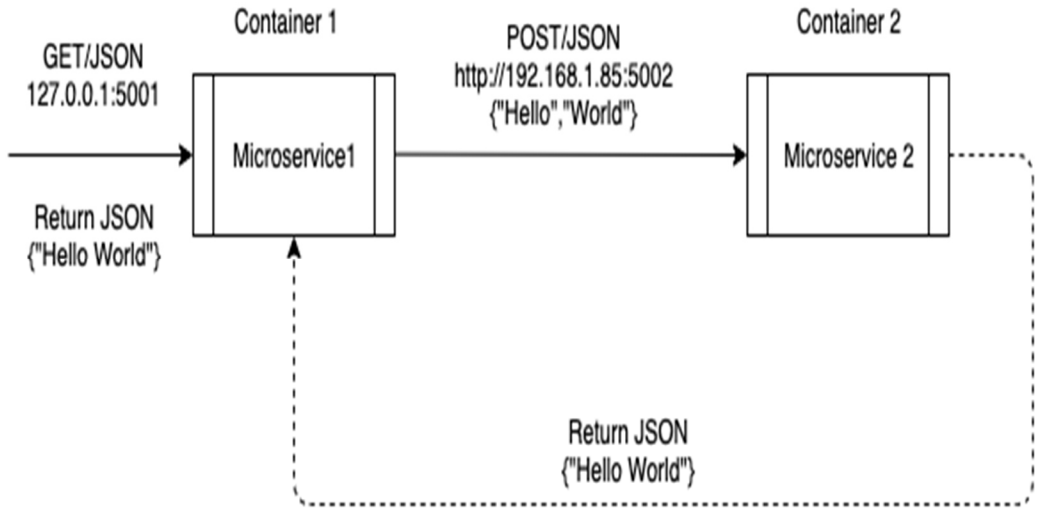

Рис. 4. Тестовий приклад застосування двох контейнерів для тестування системи керування потоками даних.

Лістінг файлу test.py

```
1. import requests
2. r1 = requests.get('http://127.0.0.1:5001/', data={'key':'value'})
3. print(r1)
4. print(r1.text)
5. r2 = requests.get('http://127.0.0.1:5002/', data={'key':'value'})
6. print(r2)
7. print(r2.text)
8. r3 = requests.get('http://127.0.0.1:5001/run', data={'key':'value'})
```

```
print(r3)
print(r3.text)
Результат роботи мікросервісів з потоками даних у контейнері:
python3 test.py
<Response [200]>
{"name":"microservice is ready"}
<Response [200]>
{"name":"microservice is ready"}
<Response [200]>
{"name":"Hello World"}
```

Висновки

У результаті проведених досліджень було доведено, що побудова системи керування потоками даних з використанням REST API можлива у невеликому обсязі коду на мові програмування Python. Розроблено відповідну архітектуру що до системи керування потоками даних з використанням мікросервісів та контейнерів. Система контейнеризації дала можливість запускати відповідні мікросервіси на одному комп'ютері, що було неможливо без застосування контейнерів, або вимагало значно складнішого коду серверної частини. Розроблено відповідний код на мові прогармування Python де перевіряється робота тестового прикладу системи керуваннння потоками даних. Наступними кроками розвитку даного напрямку може бути написання більш складного прикладу обчислень з використанням системи потоків даних, застосування оркестрації мікросервісів та дослідження передачі вхідних параметрів до мікросервісу з декількох запитів мікросервісів.

Література

1. Petrenko, A.I. Intelligent service orchestration as a service / A.I. Petrenko, B.V. Bulakh // Proc. of 2018 IEEE First International Conference on System Analysis and Intelligent Computing (SAIC), 8-12 October, 2018, Kyiv. - 2018. - 201-205 pp.

2. Petrenko A. Automatic Service Orchestration for e-Health Application / A. Petrenko, B. Bulakh // Advances in Science, Technologies and Engineering Systems Journal. - 2019. - 4(4) - 244-250 pp.

3. "Spring Cloud Data Flow Reference Guide", Docs.spring.io: [Електронний ресурс]. - Режим доступу: https://docs.spring.io/spring-cloud-dataflow/docs/current/reference/htmlsingle/. - Дата доступу: 28.01.2020.

4. "Beam Overview", Beam.apache.org: [Електронний ресурс]. - Режим доступу: https://beam.apache.org/get-started/beam-overview/. - Дата доступу: 28.01.2020.

5. Kharchenko, K. Implementation of Neural Networks with Help of a Data Flow Virtual Machine / K. Kharchenko, O. Beznosyk, V. Romanov // Proceedings of the 2018 IEEE 2nd International Conference on Data Stream Mining and Processing, DSMP 2018. - 2018. - 407 – 409 pp.

6. Kharchenko, K. A Set of Instructions for Data Flow Virtual Machine / K. Kharchenko, O. Beznosyk, V. Romanov // IEEE First Ukraine Conference on ELECTRICAL AND COMPUTER ENGINEERING (UKRCON 2017). - Kyiv, Ukraine, 2017. - 931 – 934 pp.

7. Харченко, К.В. Парадигма керування потоками даних та їх графічне представлення в SOA // Східно-Європейський журнал передових технологій. - Х: 2014. - №3/9 (69) - С. 22 - 29.

References

1. Petrenko, A.I. Intelligent service orchestration as a service / A.I. Petrenko, B.V. Bulakh // Proc. of 2018 IEEE First International Conference on System Analysis and Intelligent Computing (SAIC), 8-12 October, 2018, Kyiv. - 2018. - 201-205 pp.

2. Petrenko A. Automatic Service Orchestration for e-Health Application / A. Petrenko, B. Bulakh // Advances in Science, Technologies and Engineering Systems Journal. - 2019. - 4(4) - 244-250 pp.

3. "Spring Cloud Data Flow Reference Guide", Docs.spring.io: [Online]. - Available: https://docs.spring.io/spring-cloud-dataflow/docs/current/reference/htmlsingle/. - Accessed: 28.01.2020.

4. "Beam Overview", Beam.apache.org: [Online]. - Available: https://beam.apache.org/get-started/beam-overview/. - Accessed: 28.01.2020.

5. Kharchenko, K. Implementation of Neural Networks with Help of a Data Flow Virtual Machine / K. Kharchenko, O. Beznosyk, V. Romanov // Proceedings of the 2018 IEEE 2nd International Conference on Data Stream Mining and Processing, DSMP 2018. - 2018. - 407 – 409 pp.

6. Kharchenko,K. A Set of Instructions for Data Flow Virtual Machine / K. Kharchenko, O. Beznosyk, V. Romanov // IEEE First Ukraine Conference on ELECTRICAL AND COMPUTER ENGINEERING (UKRCON 2017). - Kyiv, Ukraine, 2017. - 931 – 934 pp.

7. Harchenko, K.V. Paradigma keruvannya potokami danih ta їh grafichne predstavlennya v SOA // Shidno-Єvropejs'kij zhurnal peredovih tehnologij. - X: 2014. - №3/9 (69) - S. 22 - 29.