УДК 004.75
DOI: 10.20535/SRIT.2308-8893.2021.3.04

ПОРІВНЯННЯ ВИКОРИСТАННЯ APACHE OPENWHISK ТА GOOGLE CLOUD FUNCTIONS ДЛЯ РОЗРОБЛЕННЯ БЕЗСЕРВЕРНИХ ЗАСТОСУНКІВ НА БАЗІ GOOGLE CLOUD PLATFORM

Т.Є. КОНДРАТЮК, Т.О. НАУМЕНКО

Анотація. Порівняно використання пропрієтарної хмарної платформи Google Cloud Functions з платформою з відкритим вихідним кодом Apache OpenWhisk для написання безсерверних застосунків на базі Google Cloud Platform. Для порівняння обрано такі критерії: підтримувані мови програмування, зручність та швидкість розроблення і розгортання функцій, можливості передавання параметрів до функцій, опції моніторингу стану функцій, швидкість холодного старту. У результаті визначено основні переваги й недоліки кожної платформи. На основі особливостей платформ наведено рекомендації щодо використання. Результати дослідження можуть бути використані для подальшого вивчення FaaS технологій, оскільки в цій царині існує величезна кількість рішень, котрі також необхідно порівнювати.

Ключові слова: Google Cloud Platform, Google Cloud Functions, Apache OpenWhisk, відкритий вихідний код, функція як сервіс, мікросервіси.

ВСТУП

Функція як сервіс (FaaS) є однією із форм безсерверної парадигми хмарних обчислень і визначається за допомогою платформ FaaS, котрі виконують фрагменти коду, які ініціюються подіями (тобто функції).

Після появи Amazon Lambda в 2014 р. безсерверні обчислення і, зокрема, функції як сервіси підкорили галузь розроблення програмного забезпечення [1]. Дійсно, їх можливість обчислень, що зумовлені подіями (event-drivencomputing) і масштабуються до тисяч одночасних функцій, спонукають багатьох користувачів хмарних технологій застосовувати безсерверні обчислення в найрізноманітніших випадках. У FaaS розробники надають невеликі фрагменти вихідного коду у вигляді функцій мови програмування, що дотримуються чітко визначеного інтерфейсу. Ці функції активуються подіями, такими як вхідні HTTP-запити або додавання даних до сховища. Хмарний провайдер виконує функцію та автоматично масштабує ресурси для обслуговування робочих навантажень. FaaS використовується у різних випадках, у тому числі як «клей», що поєднує більший безсерверний застосунок, як бекенд-технологія для реалізації REST сервісу, а також для аналітичних даних та завдань машинного навчання.

Упродовж останніх років усі основні постачальники хмарних послуг подали свої FaaS рішення, включаючи учасників «великої трійки», а саме: AWS Lambda, Azure Functions, Google Cloud Functions. Водночас з ними з'явилося багато FaaS платформ з відкритим вихідним кодом, таких як

OpenFaaS, Kubeless, OpenWhisk, Knative, Fission, що підтримують розгортання функцій на базі Kubernetes кластера. У зв'язку з таким розмаїттям стає дедалі важче обирати постачальника для власних потреб.

Аналіз останніх досліджень. Сфера безсерверних обчислень стрімко розвивається, постійно з'являються нові завдання та їх вирішення. Велика кількість конкурентів у цій галузі спонукає порівнювати подані рішення. Через постійні вдосконалення сервісів та обчислювальної техніки, а також різні методи вимірювання залишаються актуальними наукові праці, у яких порівнюються FaaS платформи за функціональністю, швидкістю виконання, ціною тощо [9, 10, 11, 12].

Формулювання цілей роботи (постановка завдання). Визначити переваги і недоліки використання Apache OpenWhisk як FaaS рішення для Google Cloud порівняно з Cloud Functions.

ПІДТРИМУВАНІ МОВИ ПРОГРАМУВАННЯ

Одним з важливих критеріїв порівняння FaaS платформ є підтримання мов програмування (див. таблицю). На даному етапі розвитку інформаційних технологій наукове суспільство підтримує ідею обрання мови програмування під необхідний функціонал, тож необхідно підтримувати якомога більше мов для залучення більшої аудиторії користувачів.

Підтримувані платформи [2, 3]

Платформа	Cloud Functions	OpenWhisk
Node.js	10, 12, 14	10, 12, 14
Python	3.7, 3.8, 3.9	2.7, 3.7
Go	1.13	1.13, 1.14, 1.15
Java	11	8
Ruby	2.6, 2.7	2.5
PHP	7.4 (beta)	7.3, 7.4, 8.0
.NET	3.1 (beta)	2.2
Swift	-	3.1.1, 4.1, 4.2
Ballerina	-	0.990.2 (beta)
Rust	-	1.34
Власні Docker образи	-	+

Як бачимо, OpenWhisk виділяється більшою кількістю підтримуваних платформ. Перевагою також є можливість використовувати власні Docker образи як середовище для функцій. Їх можна писати з нуля або на базі вже існуючих образів від OpenWhisk. Проте є кілька обмежень [4]:

- Власні Docker образи повинні реалізувати інтерфейс Action. Це протокол, що використовується платформою для передавання запитів на виклик до контейнерів. Очікується, що в контейнерах працюватиме HTTP сервер на порту 8080 з кінцевими точками /init та /run.
- Власні Docker образи повинні бути доступні на DockerHub. DockerHub — єдиний реєстр контейнерів, який зараз підтримується. Це означає, що всі образи повинні бути загальнодоступними.
- Власні Docker образи будуть стягуватися з DockerHub до локального реєстру платформи після першого виклику. Це може призвести до збіль-

шення часу холодного старту з першим викликом для нового або оновленого образу. Після витягування зображень їх кешують локально.

Як недолік OpenWhisk можна виокремити відсутність середовища виконання Java 11. На час написання цієї роботи рішення з підтриманням Java 11 перебуває на етапі розгляду вже протягом двох років [5].

РОЗРОБЛЕННЯ ТА РОЗГОРТАННЯ ФУНКЦІЙ

Щоб розробити функцію для Cloud Functions, необхідно встановити Function Framework для обраної мови програмування. Він дозволяє запустити функцію локально. Після цього її можна викликати за допомогою утиліти curl.

Способи розгортання Google Cloud Functions:

- Утиліта gcloud;
- Cloud Functions API;
- Графічний редактор у GCP Console.

Розроблення для OpenWhisk потребує налаштування Kubernetes кластера. Для локального розроблення рекомендовано використання Kubernetes, що постачається з Docker (тобто Docker для Mac або Windows, Minikube для Linux). Необхідно завантажити репозиторій з github, що включає конфігурації у форматі yaml для Helm. Далі в конфігурації вказати IP кластера та разгорнути на ньому сервіси OpenWhisk за допомогою helm. Для розгортки функцій використовується утиліта wsk.

Отже, лише для установлення середовища OpenWhisk необхідно володіти хоча б мінімальними навичками роботи з Kubernetes та Helm. Ці знання також необхідні для моніторингу стану функцій, тоді як функцію для Google Cloud Functions можна написати та запустити безпосередньо в браузері без будь-яких інструкцій.

ПЕРЕДАВАННЯ ПАРАМЕТРІВ ДО ФУНКЦІЙ

HTTP Cloud Functions приймає запит як параметр функції, тому для передавання даних можна використовувати параметри запиту, JSON запит або форму (рис. 1–6).

```
#google cloud functions
def helloFunction(request):
    name = request.args.get('name')
    return f"Hi {name}"
```

Рис. 1. Приклад функції мовою Python, що приймає дані через параметри запиту

```
$ curl -X GET https://<url>/helloFuntion?name=Taras
Hi Taras
```

Рис. 2. Приклад виклику функції з рис. 1

```
#google cloud functions
def helloFunction(request):
    name = request.form.get('name')
    return f"Hi {name}"
```

Рис. 3. Приклад функції на мові Python, що приймає дані через форму

```
$ curl -X GET https://<url>/helloFuntion -d "name=Taras"
Hi Taras
```

Рис. 4. Приклад виклику функції з рис. 3

Для функцій можна задавати змінні середовища (environment variables). Це можна зробити через gcloud або користувацький інтерфейс Cloud Console.

```
#google cloud functions
def helloFunction(request):
    name = request.get_json().get('name')
    return f"Hi {name}"
```

Рис. 5. Приклад функції мовою Python, що приймає дані через JSON тіло запиту

```
$ curl -X POST https://<url>/helloFuntion -d '{"name": "Taras"}'
Hi Taras
```

Рис. 6. Приклад виклику функції з рис. 5

Для Cloud Functions автоматично передається змінна середовища Google_Application_Credentials, яка використовується бібліотеками авторизації Google для отримання Google Credentials, котрі необхідні для доступу до сторонніх сервісів у хмарі (наприклад, баз даних).

Apache OpenWhisk підтримує єдиний спосіб передавання змінних до функцій — параметри. Фактично параметр — це звичайний JSON об'єкт. Також є можливість задавати параметри за замовчуванням під час розгортання функції (рис. 7–9).

```
#apache openwhisk
def main(args):
    name = args.get('name')
    place = args.get('place')
    greeting = 'Hi ' + name + ' from ' + place
    return {"greeting": greeting}
```

Рис. 7. Приклад функції мовою Python

```
$ wsk action update helloFunction --param place Vinogradar
```

Рис. 8. Приклад задання значення за замовчуванням для параметра 'place'

```
$ wsk action invoke --result helloFunction --param name Taras
{
    "greeting": "Hi Taras from Vinogradar"
}
```

Рис. 9. Приклад виклику функції з рис. 7

Для функції OpenWhisk неможливо задати змінні середовища, тому отримати Google Credentials стандартним способом не вдастя.

Для отримання Google Credentials можна під час створення функції передавати вміст файлу з ключами як параметр за замовчуванням, а під час виконання зчитувати його.

Іншим можливим розв'язанням цієї проблеми є створення власного Docker образу, котрий зчитуватиме файл з ключами під час розгортання функції і розміщення його в контейнер. Проте таке рішення доволі складно реалізувати і може спричинити сповільнення запуску функції. Отже, підхід, коли дані передаються у функцію лише через параметри, виявляється недієздатним для використання бібліотек, що зчитують вразливі дані з диска.

МОНІТОРИНГ

За станом виконання Cloud Functions можна слідкувати за допомогою графічного інтерфейсу Cloud Console. Тут можна відслідковувати кількість викликів, час виконання, використання пам'яті, активність, помилки для кожної окремої функції.

Для моніторингу в OpenWhisk використовується Grafana. Окрім можливостей, що надає інтерфейс CloudFunctions, у Grafana також відображається інформація про холодні і теплі запуски функцій і наявні фільтри по регіону, просторі імен, імені функції. Також меню можна налаштувати під себе, видаляючи та/або переміщуючи метрики, або додаючи власні вікна та/або дошки.

ШВИДКІСТЬ ХОЛОДНИХ СТАРТІВ

Одним з недоліків динамічного виділення ресурсів для FaaS є явище, яке називається холодним стартом. По суті, програмам, які не використовувались деякий час, потрібно більше часу для запуску та оброблення першого запиту. Хмарні провайдери тримають багато загальних працівників. Щоразу, коли безсерверному застосунку потрібно масштабуватись, або від 0 до 1 екземпляра, або від N до $N+1$ екземплярів аналогічним чином, середовище виконання вибирає одного з доступних працівників і налаштовує його для обслуговування зазначеної програми. Ця процедура потребує часу, тому затримка оброблення подій збільшується. Щоб уникнути цього для кожної події, вже налаштований для виконання певної функції працівник деякий час залишається піднятим. Коли знову знадобиться його викликати, цей працівник залишатиметься доступним для якнайшвидшого оброблення події. Така ситуація називається теплим стартом [6]. Результати холодного старту для різних мов програмування на протестованих платформах зображено на рис. 10–12.

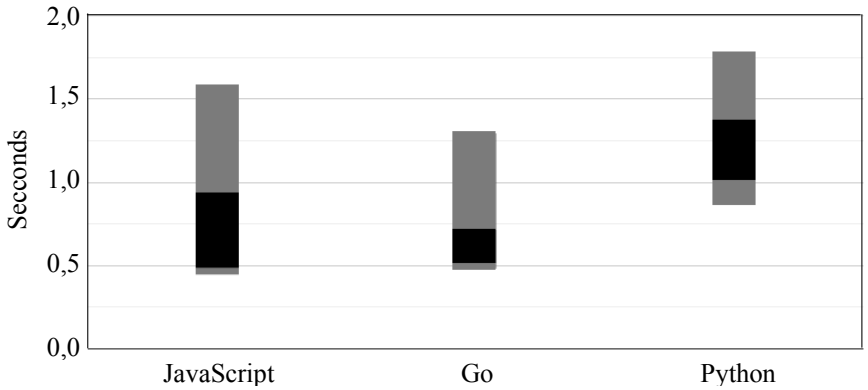

Рис 10. Тривалість холодного старту Google Cloud Functions для різних мов. Темніші ділянки — 67% випадків, світліші — 95% [7]

Із графіків, зображених на рис. 10–12, можна зробити висновок, що тривалість холодного старту функцій в OpenWhisk зі стандартними налаштуваннями можна порівняти з тривалістю холодного запуску для Google Cloud Functions. Проте для OpenWhisk можливе використання оптимізованих параметрів запуску, котрі пришвидшують виконання холодної функції більш ніж удвічі.

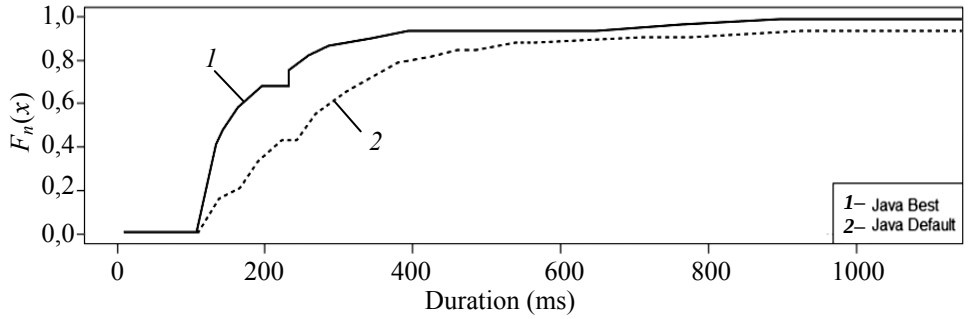

Рис 11. Тривалість холодного старту Apache OpenWhisk для мови Java для стандартних та оптимізованих параметрів функції [8]

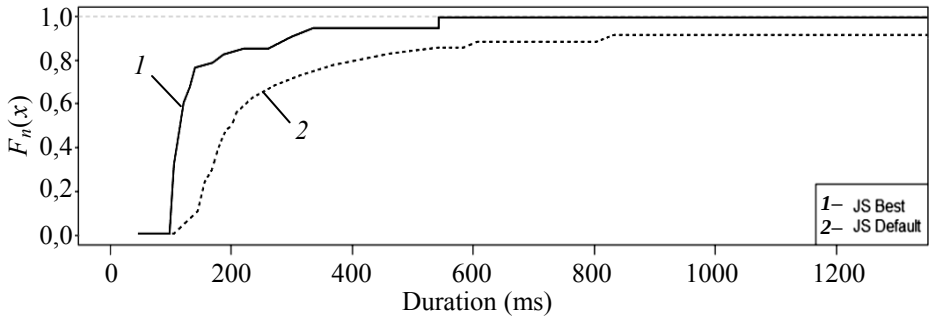

Рис 12. Тривалість холодного старту Apache OpenWhisk для мови JavaScript для стандартних та оптимізованих параметрів функції [8]

ВИСНОВКИ

Результати дослідження демонструють, що платформа Apache OpenWhisk є цікавою FaaS альтернативою для Google Cloud Functions. Попри вищий вхідний поріг, ця технологія випереджає сервіс Google за гнучкістю через більшу кількість підтримуваних платформ, можливість тонкого налаштування кластера Kubernetes та використання спеціалізованих Docker образів. Також важливою перевагою є незалежність від вендора — кластер з OpenWhisk можна підняти як у будь-якого постачальника хмарних послуг, так і на власному обладнанні. Цю технологію зручно використовувати і як доповнення до вже наявних сервісів, що використовують інфраструктуру Kubernetes. У цілому платформа Apache OpenWhisk з відкритим вихідним кодом є більш гнучкою і не менш функціональною за пропрієтарну платформу CloudFunctions.

Apache OpenWhisk успішно бере участь в перегонах з іншими FaaS платформами і продовжує вдосконалюватись зусиллями спільноти програмістів, що вносять зміни до її вихідного коду. Це спонукає постачальників

хмарних послуг до вдосконалення власних рішень, а отже, сприяє прогресу в цій галузі.

Проведене дослідження концентрується на порівнянні конкретних двох платформ за конкретними показниками, що дає змогу виявити кращі характеристики кожної з них на даному етапі розвитку FaaS технології. А також є підтвердженням важливості подальших досліджень, що дозволять порівнювати інші платформи та розширювати кількість показників для підтримання наукової спільноти з розроблення кращих рішень.

ЛІТЕРАТУРА

1. Nane Kratzke, *A Brief History of Cloud Application Architectures*. Lübeck University of Applied Sciences, Department of Electrical Engineering and Computer Science, 2018. doi:10.3390/app8081368
2. *Google Cloud Functions*. Available at: https://cloud.google.com/functions/docs/writing
3. *Apache OpenWhisk.* Available at: https://openwhisk.apache.org/documentation.html
4. *Apache OpenWhisk Docker actions*. Available at: https://github.com/apache/openwhisk/blob/master/docs/actions-docker.md
5. *Apache OpenWhisk java 11 runtime pull request.* Available at: https://github.com/apache/openwhisk-runtime-java/pull/82
6. Mikhail Shilkov, *What is a cold start?* Available at: https://mikhail.io/ serverless/coldstarts/define/
7. Mikhail Shilkov, *Cold Starts in Google Cloud Functions*. Available at: https://mikhail.io/serverless/coldstarts/gcp/
8. Sebastián Quevedo, Freddy Merchán, Rafael Rivadeneira, and Federico Dominguez, "EvaluatingApacheOpenWhisk – FaaS", *Easy Chair Preprint,* 2019. doi: http://doi.org/10.1109/ETCM48019.2019.9014867
9. Joel Scheuner and Philip Leintner, "Function-as-a-Service performance evaluation: A multivocalliterature review", *Journal of Systems and Software*, 2020.doi: http://doi.org/10.1016/j.jss.2020.110708
10. Brecht De Rooms, *A Comparison of Serverless Function (FaaS) Providers*. Available at: https://fauna.com/blog/comparison-faas-providers
11. T. Naumenko and A. Petrenko, "Analysis of problems of storage and processing of data inserverless technologies", *Technology Audit and Production Reserves*, 2 (2 (58)), pp. 20–25, 2021. doi: http://doi.org/10.15587/2706-5448.2021.230174http://doi.org/10.15587/2706-5448.2021.230174
12. Bernard Brode, *Why the Serverless Revolution Has Stalled*. Available at: https://www.infoq.com/articles/serverless-stalled/

Надійшла 01.07.2021

INFORMATION ON THE ARTICLE

Taras Y. Kondratiuk, ORCID: 0000-0001-8161-2751, Educational and Scientific Complex "Institute for Applied System Analysis" of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine, e-mail: kondratiuk.taras123@gmail.com

Tetiana O. Naumenko, ORCID: 0000-0002-8660-597X, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine, e-mail: tnaumenko13@gmail.com

СРАВНЕНИЕ ИСПОЛЬЗОВАНИЯ APACHE OPENWHISK И GOOGLE CLOUD FUNCTIONS ДЛЯ РАЗРАБОТКИ БЕССЕРВЕРНЫХ ПРИЛОЖЕНИЙ НА БАЗЕ GOOGLE CLOUD PLATFORM / Т.Е. Кондратюк, Т.А. Науменко

Аннотация. Произведено сравнение использования проприетарной облачной платформы Google Cloud Functions с платформой с открытым исходным кодом Apache OpenWhisk для написания бессерверных приложений на базе Google Cloud Platform. Для сравнения выбраны такие критерии: поддерживаемые языки программирования, удобство и скорость разработки и развертывания функции, возможности передачи параметров в функции, опции мониторинга состояния функций, скорость холодного старта. В результате были определены преимущества и недостатки каждой платформы. На базе особенностей платформ, приведены рекомендации к использованию. Результаты исследования могут быть использованы для дальнейшего изучения FaaS технологий, поскольку в этой области существует огромное количество решений, которые также необходимо сравнивать.

Ключевые слова: Google Cloud Platform, Google Cloud Functions, Apache OpenWhisk, открытый исходный код, функция как сервис, микросервисы.

COMPARISON OF USING APACHE OPENWHISK AND GOOGLE CLOUD FUNCTIONS FOR DEVELOPMENT OF SERVERLESS APPLICATIONS ON GOOGLE CLOUD PLATFORM / T.Y. Kondratiuk, T.O. Naumenko

Abstract. A comparison of using proprietary cloud platform Google Cloud Functions and open source platform Apache OpenWhisk for writing serverless applications based on the Google Cloud Platform was made. The following criteria were chosen for comparison: supported programming languages, convenience and speed of development and deployment of functions, possible ways to pass parameters to functions, options for monitoring the status of functions, speed of the cold start. As a result, main advantages and disadvantages of each platform were identified. Recommendations for use cases are given based on the features of the platforms. The results of the study can be used for further studies in the field of FaaS technologies, as there are a lot of solutions in this area that also need to be compared.

Keywords: Google Cloud Platform, Google Cloud Functions, Apache OpenWhisk, open source code, function as a service, microservices.