Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

Огляд сучасних платформ для побудови Serverless архітектури

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
доцент Франчук О.В
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2020 р.
Виконав студент ІПЗ-1
Гавришко Ярослав
(прізвище та ініціали)
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Декан факультету інформатики,
доцент., д.т.н.
_____А. М. Глибовець
(підпис)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту 1-го курсу магістратури, факультету інформатики__
Гавришку Ярославу Олеговичу_____
ТЕМА: Побудова системи для оцінки схожості зображень

Зміст ТЧ до магістерської роботи:

Зміст
Анотація
Вступ
1 РОЗДІЛ 1: Розвиток технології FaaS
2 РОЗДІЛ 2: Serverless рішення
3 РОЗДІЛ 3: Переваги та недоліки
4 РОЗДІЛ 4: Майбутнє Serverless технології і можливі рішення для сучасних проблем
Висновки
Список літератури

Дата видачі „___” _____ 2020 р. Керівник _____
(підпис)

Завдання отримав _____
(підпис)

ЗМІСТ

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	лютий 2020 р.	
2.	Оволодіння інформацією про хмарні обчислення	15.02.2020	
3.	Дослідження різних підходів для побудови serverless архітектури	22.02.2020	
4.	Реалізація простих застосунків за допомогою провідних провайдерів	15.03.2020	
5.	Побудова serverless системи	20.04.2020	
6.	Тестування	25.04.2020	
7.	Порівняльний аналіз провайдерів	02.05.2020	
8.	Остаточне оформлення пояснювальної роботи та слайдів	03.05.2020	
9.	Захист курсової роботи	29.05.2020	

Студент _____**Гавришко Я.О**_____

Керівник _____**Франчук О.В**_____

“_____”_____

Анотація

Останнім часом, популярність serverless обчислень різко зростає, ажде сьогодні це один з найкращих способів для управління коштами, надійністю, доступністю та масштабованістю. У даній курсовій роботі представлено деталі serverless пропозицій від провідних провайдерів таких, як: AWS, Azure, Google Cloud Platform. Було зроблено порівняння між ними за відповідними категоріями: сховище, обчислення, бази даних, обмін повідомленнями, управління API’s та інші інструментарії. Також було представлено порівняльний аналіз serverless архітектур для найбільш поширених випадків з акцентом на перевагах, відкритих проблемах та можливих рішень.

Перелік умовних позначень, символів, скорочень і термінів

Serverless - безсерверні обчислення, модель хмарних обчислень для яких платформа динамічно керує виділенням машинних ресурсів.

FaaS (Function as a Service) - у даній роботі синонім Serverless.

BaaS (Backend as a Service) - модель для надання розробникам веб-додатків і мобільних додатків способу пов’язати свої додатки з резервним хмарним сховищем та API, а також надає такі функції, як управління користувачами, push-сповіщення та інтеграція із службами соціальних мереж.

PaaS (Platform as a Service) - модель надання хмарних обчислень, при якій споживач отримує доступ до використання інформаційно-технологічних платформ: операційних систем, систем управління базами даних, зв'язного програмного забезпечення, засобів розробки і тестування розміщених у хмарних провайдерах.[7]

API (Application Programming Interface) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

ПЗ - програмне забезпечення.

Вступ

Serverless обчислення - це еволюційна технологія, що дозволяє створювати і запускати застосунки і сервіси не турбуючись про сервери. При такому типі реалізації немає необхідності володіти якоюсь інфраструктурою. У serverless архітектурі основна частина програми працює у ефемерних контейнерах, наданих головними хмарними провайдерами. Ці контейнери запускаються на певні події та припиняють свою роботу після виконання. Такий тип хмарних обчислень інакше називається, як Function-as-a-Service(Faas), оскільки одиницею коду є функція, яка виконується платформою.

Метою даної роботи є огляд serverless пропозицій від головних провайдерів хмарних рішень, порівняльний аналіз способів реалізації serverless архітектур для найбільш поширених випадків.

РОЗДІЛ 1: Розвиток технології FaaS

Історія

Раніше системні адміністратори готували фізичні сервери для розгортання програмного забезпечення. Це був відносно довгий процес: потрібно було встановити операційну систему, відповідні драйвери для пристроїв, переконатись що є достатньо ресурсів таких, як: пам’ять, диск, процесор. Також було потрібно дбати про підтримку та оновлення обладнання. Такий підхід відомий за назвою як “bare metal”, адже тут одиницею розгортання є фактичний сервер.

Потім з’явився інший підхід до розгортання ПЗ - віртуальна машина. Тепер, замість того, щоб розгортати застосунок на окремому апараті, розробники могли зробити це за допомогою імітації сервера. Це призвело до великої гнучкості при модернізації та міграції ПЗ, а також зробило розгортання більш повторюваним та гнучким, і дозволило системним адміністраторам відв’язати програмне забезпечення від апаратного. Тепер, якщо стався апаратний збій, то системний адміністратор міг перенести віртуальну машину на інше обладнання та уникнути проблем. Однак віртуальні машини все ще мали деякі обмеження та накладні витрати. Вони видавали себе справжніми серверами і це було не завжди потрібно. Тут одиницею розгортання є віртуальна машина.

Продовженням роботи з віртуальними машинами було розгортання контейнерів. З’явилися різні технології контейнерної обробки, такі як: Docker, OpenVZ, LXC, FreeBSD-зони. Ці технології дозволили системному адміністратору “розділити” операційну систему і мати різні програми, що

працюють в одній і тій же системі, не заважаючи один одному. Крім того, розроблено багато інструментів для полегшення створення та обслуговування контейнерів. Багато компаній використовують це для покращення процесів розробки та розгортання ПЗ. Тут одиницею розгортання є контейнер.

У кожній з цих парадигм важливо де саме виконується програмне забезпечення, чи то на фізичному сервері, чи у віртуальній машині, чи у хмарі. Serverless архітектура дає нам ще один рівень абстракції: сам код. З цим новим рівнем абстрагування нам не потрібно дбати про місце, де буде виконуватись наш код.[1]

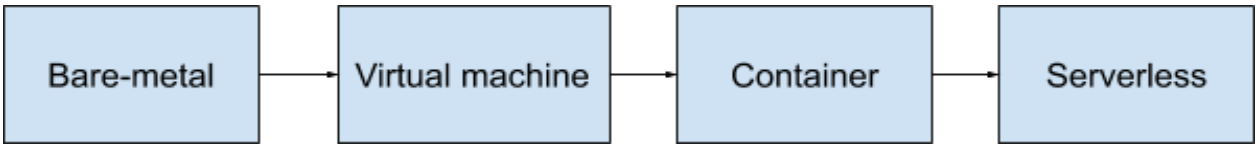

Рисунок 1

Основні характеристики

FaaS дозволяє запускати код без підготовки та керування серверами. Ось його основні характеристики:

1. Будь який тип backend коду, написаний на будь якій популярній мові програмування.
2. Не потребує жодного адміністрування, розробник хвилюється лише про завантаження коду.
3. Автоматичне збільшення обчислювальної потужності
4. Висока доступність

5. Код може бути викликаний більшістю хмарних сервісів, а також API.

У serverless додатках усі компоненти архітектури не потребують забезпечення, обслуговування та адміністрування серверів. Залежно від використання ці компоненти можуть вміщати в себе обчислення, різні сховища, бази даних, машине навчання, обробку потоків, чергу повідомлень, стрімінг тощо.

РОЗДІЛ 2: Serverless рішення

Serverless - це комбінація FaaS і BaaS(Backend-as-a-Service)[1]. На високому рівні PaaS(Platform-as-a-Service) схожий на serverless підхід, але це не так. Адріан Коккрофт сказав: “Якщо ваш PaaS може ефективно запускати інстанси кожні 20 мілісекунд, які виконуються пів секунди, тоді ми можемо назвати це serverless.” . FaaS повинен бути набагато більш scalable ніж PaaS[1]. Поведінка serverless дуже схожа на ситуацію у публічному транспорті - користуйтесь ним лише коли потрібно, платіть лише коли користуєтесь.

Serverless рішення від провідних провайдерів

Serverless технологія забезпечує абстракцію серверів, інфраструктури і операційних систем. В останній роки усі основні хмарні провайдери пропонують велику кількість serverless пропозицій.
У Таблиці 1 наведено список усіх serverless пропозицій від провідних провайдерів.

AWS

Amazon Web Services (AWS) - лідер на ринку у cloud просторі[2]. Вони мають найдосконаліший набір serverless продуктів. Ці повністю керовані сервіси дозволяють розробникам білдити та запускати serverless додатки. Нещодавно AWS запустила Serverless Application Repository, що є

по суті початковою точкою для serverless проектів. У цьому репозиторії є кілька загальнодоступних serverless додатків, які можна знайти, задеплоїти та запаблішити всього у пару кліків, як показано на малюнку нижче.

Рисунок 2

Serverless Application Use cases

Веб додатки і бекенд. Наприклад: розглянемо serverless додаток з новинами. У цьому випадку ми можемо хостити код вебсайту у S3, для обробки даних можемо використати Lambda, що буде отримувати дані з DynamoDB та експоузити їх через API gateway. Показано малюнок нижче.

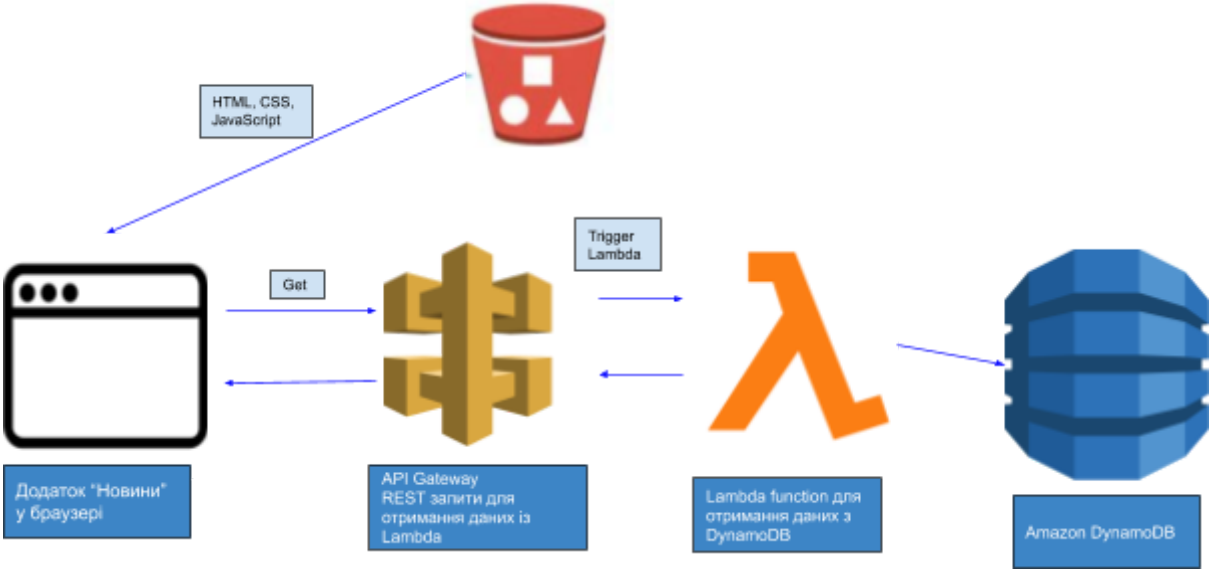

Рисунок 3. [5]

Data processing

Data processing систему у реальному часі може бути зроблена за допомогою AWS Lambda, Amazon Kinesis, S3 і DynamoDB. На Рисунку 4 зображено просту serverless архітектуру для додатку редагування фотографій.

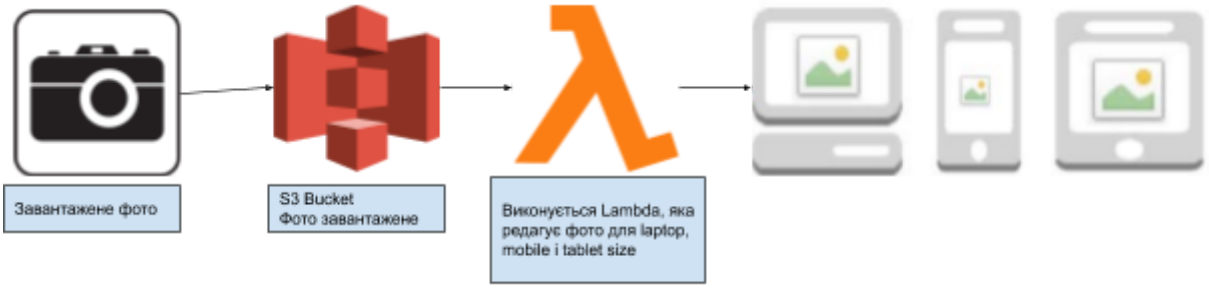

Рисунок 4. [2]

Заплановані задачі і аналіз логів

Lambda - прекрасний приклад для планування задач, логів та побудови безпосередньо самого планування.

На рисунку нижче показано процеси збору та аналітики логів використовуючи Kinesis і ElasticSearch.

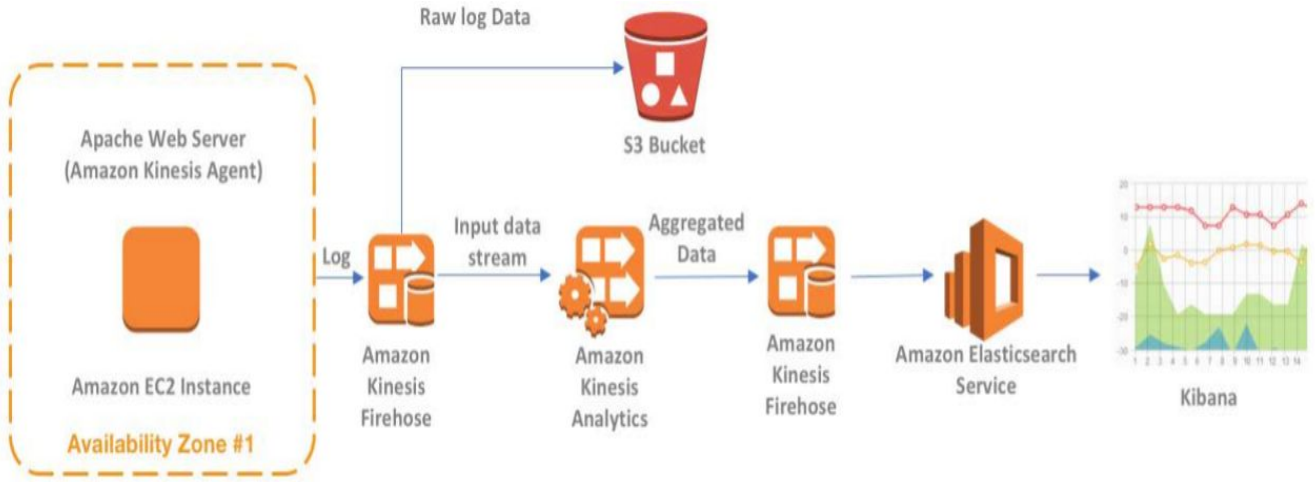

Рисунок 5. Аналіз логів за допомогою AWS. [6]

Azure

Microsoft Azure є другим на ринку хмарних рішень. Вони надають широкий сет інтрументів для побудови serviceless додатків. Azure може бути найкращим вибором, якщо організація чи компанія вже використовує значну кількість програмного забезпечення від Microsoft. [3]

Нижче наведено деякі use-cases і архітектури, які рекомендує сам Azure.

Архітектура веб застосунків

Azure Functions є FaaS рішенням від Azure, яке може бути exposed використовуючи WebHook URL, щоб працювати як мікросервіси. Це хороший підхід для , наприклад, виконання CRUD операцій для single web page. На рисунку нижче зображено додаток для цільової реклами відповідно до налаштувань користувача.

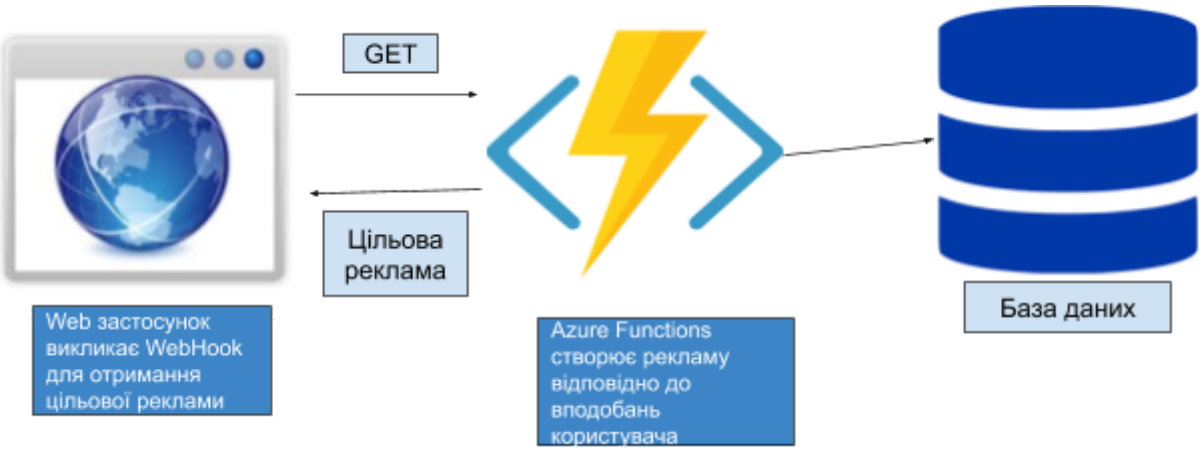

Рисунок 6. [3]

IoT

Azure Stream Analytics отримує повідомлення від Internet of Things пристроїв, потім викликає Azure function для обробки, трансформування даних і записує їх у Azure CosmosDB як показано на Рисунку 7.

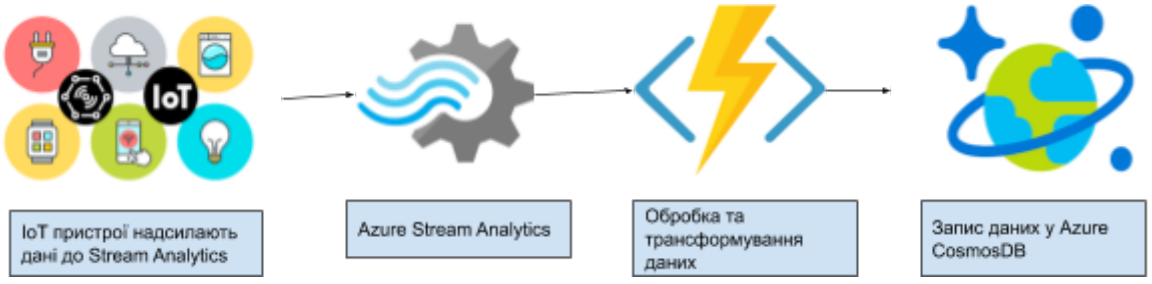

Рисунок 7. [3]

Інтеграція з SaaS

Azure Functions можуть бути викликані за допомогою певної активності у SaaS-based застосунках. Наступний Рисунок 8 демонструє приклад зберігання файлу у OneDrive, який викликає Azure function. У цій функції, ми можемо модифікувати excel файл, також можна створювати графіки для аналізу за допомогою Microsoft Graph API.

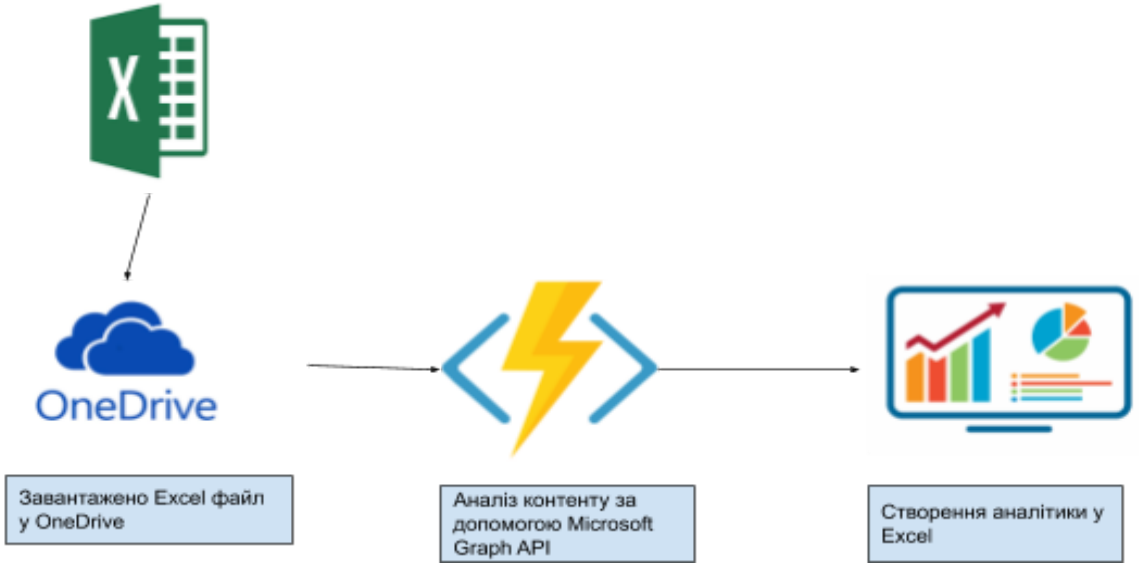

Рисунок 8. [3]

Back-end for mobile

Azure Functions підтримує HTTP triggers і “output bindings”. Такі тригери можна налаштувати під реакцію на WebHooks і працювати з цим як з API. Такі API часто використовують як бекенд для мобільних застосунків. Рисунок 9 демонструє додаток з наступним флоу: мобільний

додаток робить знімок та викликає функцію в Azure, для того щоб отримати “authorization token” і зберегти зображення у blob-сховищі. Інша Azure функція змінює розмір зображення та завантажує його у blob-сховище.

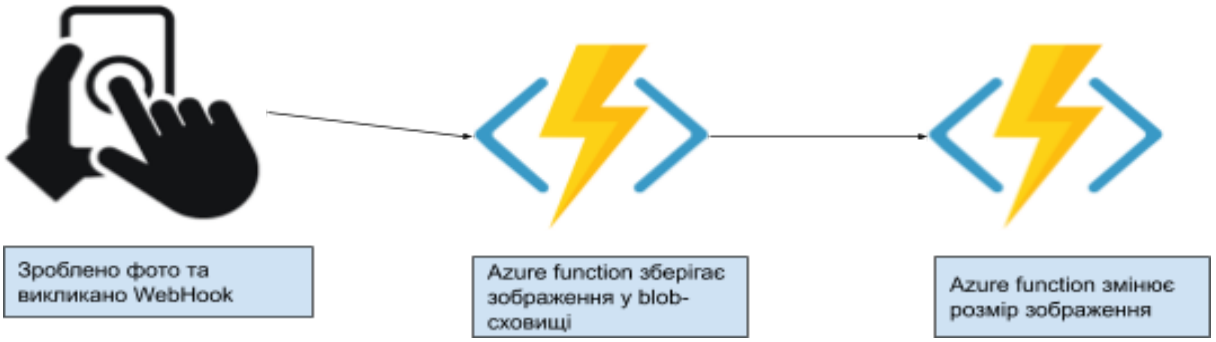

Рисунок 9. [3]

Google

Google третій головний гравець на ринку хмарних рішень. Він також має велику кількість пропозицій для побудови serverless застосунків. Ми можемо визначити три три найбільш популярні випадки використання.[4] Google serverless applications:

- Web app з компонентами: Browser -> App Engine -> Datastore
- Мікросервіс з компонентами: Мікросервіс -> Хмарна функція -> Datastore
- ETL з компонентами: File -> Cloud Dataflow -> BigQuery

У Таблиці 1 нижче наведено приклади use-case-ів за допомогою Google serverless технології.

Таблиця 1. Serverless у Google [4]

Випадок	Google serverless product
Мобільний додаток	Firebase/Firestore
Web клієнт	Firebase/Firestore
Web backend	App Engine -> Datastore
Мікросервіси	Cloud Functions -> Datastore
Data Processing	Cloud Functions/Cloud Functions for Firebase
Боти	Cloud Functions
IoT device messages	Cloud Pub/Sub -> Dataflow
NoSql бази даних	Cloud Datastore
ETL	Cloud Dataflow -> BigQuery
Blob-сховище	Cloud Storage
Analytics warehouse	BigQuery
Personalization	Cloud Machine LearningEngine

Нижче на Рисунку 10 зображено serverless архітектура побудована за допомогою Firebase і Google Cloud Function.

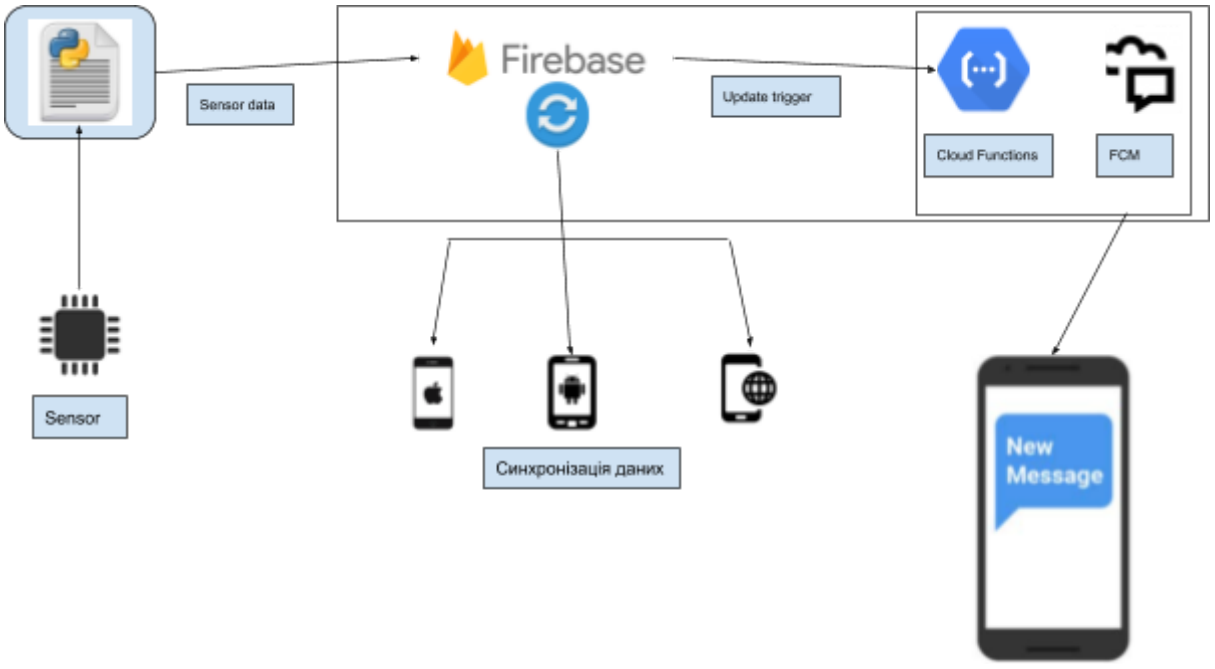

Рисунок 10

У наведеному прикладі програма написана на python зчитує дані з сенсору і записує їх у Firebase базу даних, використовуючи HTTP запити. Firebase обновлює усі девайси у реальному часі та є трігером для клауд функції яка займається розсилкою повідомлень.

РОЗДІЛ 3: Переваги та недоліки, відкриті проблеми

Serverless технологія розвивається так швидко, що багато недоліків які є на даний момент через короткий проміжок часу будуть не актуальними. Пропоную розглянути основні переваги та недоліки serverless архітектури.

Переваги:

- Автоматичне збільшення обчислювальної потужності: згадайте гру Pokemon Go, чи можливо вона без хмарних технологій? Serverless масштабування немає межі. Забезпечувати інфраструктуру - високий ризик. Програми повинні маштабуватись або зменшуватись відповідно до зовнішніх запитів.
- Низька ціна: користувач оплачує лише те, що він використав. Не потрібно переплачувати за простій системи.
- Немає потреби у підтримці інфраструктури: користувачу не потрібно мати власної архітектури.
- Легке розвертання застосунків: більшість провайдерів пропонують command-line interface(CLI) для легкої та швидкісної викладки застосунку.
- Немає потреби у налаштуванні безпеки застосунку: інфраструктура повністю підтримується провайдером, тобто уся відповідальність за безпеку на стороні провайдера.
- Можливість виконувати код у фізичній близькості до кінцевого користувача.
- Економія часу: велика кількість сервісів, яка допомагає у розробці, вже є реалізована тими ж провайдерами.

Недоліки:

- Затримка: для високонавантажених застосунків старт FaaS системи може бути досить повільним.
- Дебагінг і моніторинг є складним у більшості serverless застосунків.
- Новизна технології: дана технологія є досить новою і не є ще достатньо випробуваною у великих ентерпрайз застосунках.
- Проблеми з security: моніторинг, підтримка і тестування serverless застосунків є складним, що у свою чергу залишає багато відкритих до уражень місць.
- Велика зв’язність з провайдером: при збільшенні розмірів застосунку досить складно мігрувати компоненти/цілий застосунок до іншого провайдера.
- Відсутність локальних data storage: FaaS це stateless підхід, який не зберігає жодних локальних даних, тому розробникам необхідно зберігати дані лише у базі даних.
- Ліміт виконання: присутній ліміт виконання для функцій. Для прикладу у AWS Lambda є ліміт виконання однієї функції - 5 хвилин.

РОЗДІЛ 4: Майбутнє Serverless технології і можливі рішення для сучасних проблем

Serverless системи ще досі знаходяться у періоді їхнього становлення, тому поширені стандарти, такі як: декларація констант, дозволів, управління секретними ключами, робота з параметрами для подій, визначення кофігурацій - все це ще є відсутнім у serverless системах. З таким стрімким розвитком технології, у найближчі роки можемо очікувати вирішення загально відомих “больових” точок. Давайте більш детально поговоримо про ці “больові” точки та їх можливі рішення.

Конфігурування, дебагінг, логування, моніторинг і деплоймент

Конфігурування, дебагінг, білдинг, логування, моніторинг і деплоймент - це найбільш критичні проблеми у serverless технології. У найближчий час стандарти хмарних платформ будуть значно збільшуватись через високий попит на них. Усі постачальники даних рішень будуть старатись стати кращим ти виграти конкуренцію у інших. Усі провайдери повинні підтримувати IDE з відкритим кодом для дебагінгу, деплойменту і конфігурування. Конфігурування API Gateway повинно бути добре налагодженим та налаштовуватись через простий файл конфігурації. Якщо розробник для деплою використовує IDE, в результаті Lambda і API Gateway повинні бути створені автоматично.

Час виконання, затримка часу на старті і керування сесіями

Першим рішенням для цієї проблеми може бути дозвіл створення екземпляру Lambda при першому виклику програми з орієнтовним активним терміном експлуатації. Щоразу, будь яка програма, яка запускається і в очікуванні надіслати запит до Lambda, повинна активувати Lambda заздалегідь на орієнтовний проміжок часу.

Тривалість виконання зазвичай представлена хмарними провайдерами. Завдання, які потребують довший час виконання, можуть бути розбиті на логічні частини та виконатись на декількох Lambda паралельно.

Зв'язність з провайдером

Загальна платформа з відкритим кодом для розробки та розгортання застосунків в декількох хмарних середовищах може усунити проблему високої зв’язності з провайдером.

Новизна технології та безпека

З часом serverless технологія стане “зрілою” і люди отримають більше обізнаності. Розробник дотримуватиметься нових вказівок щодо безпеки, щоб уникнути вразливостей.

Локальні data storages і доступ до платформи

Це не є так критично. Ці вимоги можуть бути вирішені використовуючи базу даних і в деяких особливих випадках, де потрібен доступ до платформи, можна використати контейнери.

Висновки

У даній роботі було досліджено найкращі підходи до реалізації хмарних serverless застосунків для різних видів використання, а також розглянуто serverless пропозиції від провідних провайдерів.

Як могли побачити, всі хмарні провайдери пропонують майже всі види serverless компонентів, але все таки AWS є лідером на даному сегменті ринку. Google дуже добре себе проявив у галузях штучного інтелекту та мобільних додатків. Azure найкраще підходить для компаній, які вже мають велику кількість ПЗ, створеного за допомогою технологій Microsoft.

Після детального ознайомлення з усіма пропозиціями, видно, що AWS більш зрілі та мають більш ширший вибір продуктів порівняно з іншими. AWS Lambda має декілька triggering points, які надають більшу гнучкість архітектурі. Lambda Edge дуже зручна річ, якщо є потреба запуску коду в залежності від локації користувача. У Google немає такої можливості, а Azure дозволяє зробити це лише для IoT пристроїв. Для сховищ і баз даних усі три провайдери надають однакові можливості, так само як і для обміну повідомленнями.

Також варто зазначити , що у serverless computing існує велика кількість технічно складних так інтелектуально глибоких проблем, починаючи від інфраструктурних проблем таких, як оптимізація першого запуску (cold start), до компонентних програмних моделей.

Список посилань

1. M. Roberts, "Serverless Architectures," MartinFowler.com, May 22, 2018.
2. Amazon Web Services, "Serverless Computing and Applications," AWS, 2018.
3. Microsoft, "Serverless Computing," Azure, 2018.
4. Google, "Serverless," GCP, 2018.
5. https://aws.amazon.com/serverless/build-a-web-app.
6. Amazon Web Services, "Build a Log Analytics Solution," AWS, 2018.
7. https://uk.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0_%D1%8F%D0%BA_%D0%BF%D0%BE%D1%81%D0%BB%D1%83%D0%B3%D0%B0