2. Understanding IoT Security – Part 2 of 3: IoT Cyber Security [Електронний ресурс] – Режим доступу до ресурсу: https://iot-analytics.com/understanding-iot-cyber-security-part-

Голіков В.В.
Криворізький економічний інститут Київського національного університету ім. Вадима Гетьмана
Вдовиченко І.Н.
К.т.н.,доцент, Криворізький національний університет

ВИКОРИСТАННЯ БЕЗСЕРВЕРНИХ ОБЧИСЛЕНЬ У WEB-ДОДАТКАХ

Проаналізовано перспективи безсерверних обчислень та serverless архітектури додатку. Огляд сильних та слабких сторін використання безсерверних технологій при розробці web-додатків.

Останнім часом хмарні технології набирають все більшої популярності. Це відбувається з простої причини - легкої доступності, відносної дешевизни і відсутності початкового капіталу - як знань для підтримки і розгортання інфраструктури, так і фінансового характеру. Serverless - безсерверна архітектура додатків. Насправді, не така вже вона й безсерверна. Основу архітектури складають мікросервіси, або функції (lambda), що виконують певне завдання і запускаються на логічних контейнерах, захованих від сторонніх очей. Тобто кінцевому користувачеві дано тільки інтерфейс завантаження коду функції (сервісу) і можливість підключення до цієї функції джерел подій (events). Іноді безсерверні обчислення також іменують «Функція як послуга», тому що одиницею коду є функція, яка виконується платформою. По суті для виконання одного запиту створюється окремий контейнер, який знищується після виконання.

Основними платформами, що надають послуги безсерверного середовища виконання є такі як AWS Lambda від Amazon, Google Cloud Functions від Google, OpenWhisk від IBM, Azure Functions від Microsoft Azure та інші [1].

Ідея безсерверних обчислень полягає в чотирьох рисах: абстракція, еластичність, ефективна вартість та обмежений життєвий цикл. За допомогою абстракції користувач не керуєте сервером, на

якому запускається програма. Він взагалі нічого не знаєте про сервер, усі нюанси операційної системи, оновлення, мережеві налаштування та інше заховане від користувача. Це зроблено для того, щоб розробники могли сконцентруватися на розробці корисного функціоналу, а не адмініструванні серверу. Еластичність виражається в тому, що провайдер безсерверних обчислень автоматично виділяє більше чи менше обчислювальних ресурсів в залежності від того, на скільки велике навантаження приходить на додаток. З цього також випливає і ефективна вартість, якщо додаток простоює — розробник нічого не платить, тому що в цей момент додаток не використовує обчислювальних ресурсів. Додаток запускається у контейнері і через короткий час, від десятка хвилин до декількох годин, сервіс автоматично його зупиняє. Звісно, якщо додаток знову повинен бути викликаний — новий контейнер буде запущено.

При деяких допущеннях, Serverless архітектура може бути використана для будь яких проектів. Однак є випадки в яких її використання легше та безпечніше — відкладені та фонові задачі. Приклади таких задач: створення резервної копії за розкладом; асинхронна відправка повідомлень користувачам (push, email, sms); різноманітні експорти та імпорти і т.д. Усі ці приклади виконуються по розкладу або не мають на увазі моментальної відповіді користувачу. Це пов'язано з тим, що додатки (функції) в безсерверній моделі не працюють постійно, а запускаються при необхідності, а у разі невикористання автоматично відключаються. Це призводить до того, що для запуску функції потребується час, іноді до декількох секунд [2].

Однак це не означає, що безсерверні обчислення неможливо використовувати в частинах додатку, з якими взаємодіють користувачі і для яких важливий час відповіді. Навпаки, Serverless функції широко використовуються у чат-ботах, бекенд частині для «інтернету речей», маніпуляція запитами до основного бекенду (наприклад, для ідентифікації користувача чи отримання інформації про його геопозицію через IP-адресу).

Перевагами використання Serverless архітектури можна виділити: максимальну еластичність, швидке масштабування з нуля до тисяч паралельно працюючих функцій; повна абстракція від операційної системи чи будь-якого програмного забезпечення, що вико-

ристовується для виконання додатку; при правильному проектуванні функцій легше побудувати слабо-зв'язану архітектуру, при якій помилка в одній функції не вплинуть на працездатність всього додатку.

Недоліків у такої архітектури багато, основними є необхідність турбуватись про збереження зворотної сумісності; схема взаємодії в класичному монолітному додатку і в розподіленій системі сильно відрізняється, необхідно думати про асинхронну взаємодію, можливих затримках, моніторингу окремих частин додатку; неправильна архітектура може призвести до того, що помилка в одній функції призведе до непрацездатності великої кількості інших; запуск функції може потребувати до декількох секунд, що може бути критично; коли запит від користувача проходить через десяток функцій важко відловити можливу причину помилки, якщо така стається; прив'язка до провайдера хмарних обчислень, функції розроблені для AWS буде важко перенести на, наприклад, Google Cloud Platform, через те що функції рідко використовуються в ізоляції, окрім них використовуються бази даних, черги повідомлень, системи логування та ін., що відрізняються від провайдера до провайдера [3].

Підсумовуючи можна сказати: хоч мінусів вийшло більше ніж плюсів, це не означає, що «Function as a Service» поганий напрям, зовсім навпаки, багато ризиків можуть бути мінімізовані, або сприйняті як факт.

ЛІТЕРАТУРА

1. AWS Serverless Application Model (AWS SAM) [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/serverless-application-model/
2. AWS Lambda и никаких серверов. [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/245949/
3. Learn to build full-stack Serverless apps. [Електронний ресурс] – Режим доступу: https://serverless-stack.com/