DOI 10.34185/1562-9945-2-163-2026-04
УДК 004.75

Ю.О. Гунченко, К.І. Камєнєв, А.В. Камєнєва, М.О. Єпік

ОЦІНКА ПРОДУКТИВНОСТІ AWS LAMBDA ДЛЯ МАСШТАБОВАНОЇ ТА БЕЗПЕЧНОЇ АВТОРИЗАЦІЇ КОРИСТУВАЧІВ У КРОСПЛАТФОРМЕНИХ ХМАРНИХ ЗАСТОСУНКАХ

Анотація. У статті проаналізовано ефективність використання безсерверної архітектури AWS Lambda для реалізації масштабованої системи авторизації користувачів у хмарному середовищі. Авторизаційна функція розгортається як кросплатформений компонент, що взаємодіє з керованою реляційною базою даних Amazon Aurora PostgreSQL і використовує сервіс AWS Systems Manager для безпечного зберігання конфігурацій. Досліджено продуктивність функції залежно від параметрів середовища (обсяг пам'яті, холодний/теплий старт), а також реалізацію генерації JWT-токенів для ідентифікації користувачів. Результати можуть бути застосовані при розробці хмарних додатків, орієнтованих на високу масштабованість, портативність і безпечну роботу з базами даних.
Ключові слова: AWS Lambda, авторизація користувачів, безсерверні обчислення, Amazon Aurora PostgreSQL, jwt токени, продуктивність хмарних функцій, кросплатформені застосунки.

Постановка проблеми. У сучасних умовах стрімкого розвитку цифрових технологій особливого значення набуває забезпечення безпечної та масштабованої авторизації користувачів у хмарних додатках. Із зростанням кількості веб- і мобільних клієнтів традиційні серверні рішення перестають задовольняти вимоги до гнучкості, продуктивності та економічної ефективності. У зв'язку з цим актуальним є впровадження архітектурних підходів, що дозволяють масштабувати обчислювальні ресурси відповідно до навантаження, забезпечуючи при цьому високу доступність і захищеність даних. Одним із таких рішень є безсерверна (serverless) архітектура, яку пропонує платформа Amazon Web Services (AWS) через сервіс AWS Lambda.

Безсерверні обчислення дозволяють виконувати програмні функції без необхідності керування інфраструктурою, що особливо важливо для розробників кросплатформених застосунків. У контексті авторизації це дає можливість створювати модульні, незалежні та легко масштабовані компоненти, які можуть бути інтегровані в мобільні або вебдодатки незалежно від операційної системи чи клієнтського середовища. При цьому забезпечується надійний доступ до централізованої бази даних, наприклад, Amazon Aurora PostgreSQL, що гарантує збереження даних, транзакційну цілісність і підтримку стандарту SQL.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Застосування AWS Lambda у поєднанні з керованою базою даних і системою керування конфігураціями (AWS Systems Manager Parameter Store) відкриває нові можливості для реалізації сучасних підходів до управління обліковими записами, зберігання секретів і генерації токенів доступу, зокрема JWT. Така система дозволяє гнучко обробляти вхідні запити, здійснювати валідацію облікових даних, створювати криптографічно захищені маркери доступу та записувати журнали подій — усе це без постійно працюючого серверного оточення.

На відміну від класичних серверних архітектур, де обслуговування запитів залежить від попередньо налаштованих інстансів, AWS Lambda виконує функції на вимогу. Це дозволяє не лише зменшити витрати, а й підвищити еластичність системи, оскільки вона автоматично адаптується до змін навантаження. Проте така гнучкість потребує ретельного аналізу продуктивності в різних умовах, зокрема при холодному та теплому старті, а також при різних обсягах виділеної пам’яті.

Аналіз останніх досліджень і публікацій. Останні роки відзначаються стрімким зростанням інтересу до безсерверних обчислень (serverless computing), що спричинило появу численних досліджень у галузі архітектур типу Function-as-a-Service (FaaS). У працях [1], [2] розглядаються ключові переваги безсерверних моделей, серед яких: автоматичне масштабування, зниження витрат на інфраструктуру, адаптивність до нерівномірного навантаження та зручність для розробників. Особливо важливою є підтримка подієвої архітектури, яка дозволяє створювати реактивні сервіси, зокрема функції авторизації на вимогу.

Розгортання авторизаційних механізмів у хмарному середовищі потребує інтеграції з керованими базами даних. У дослідженнях [3], [4] аналізується ефективність використання Amazon Aurora як масштабованого і сумісного з PostgreSQL рішення для зберігання облікових даних. Автори підкреслюють її високу доступність, автоматичне резервне копіювання, підтримку транзакцій та інтеграцію з іншими сервісами AWS. Особливий акцент робиться на важливості використання керованих конфігурацій і секретів через AWS Systems Manager Parameter Store [5], що дозволяє безпечно зберігати паролі, ключі й інші конфіденційні параметри.

Сучасні методи авторизації користувачів, описані в [6], [7], базуються на використанні токенів доступу, зокрема JWT (JSON Web Token), які підтримують кросплатформену взаємодію між клієнтом і сервером. Це дозволяє реалізувати безстанні API, що особливо важливо для безсерверних функцій, які не зберігають сесію користувача між викликами. У роботах також відзначається необхідність дотримання принципів безпечної аутентифікації та обмеження прав доступу до бази даних.

Проблематика продуктивності AWS Lambda розглядається в працях [8], [9], де автори досліджують вплив "холодного старту", обсягу виділеної пам’яті та обсягу викликів функцій на затримку відповіді. Окрему увагу приділено оптимізації часу виконання для задач, пов’язаних із доступом до баз даних і генерацією токенів. У статті [10] проводиться порівняльний аналіз AWS Lambda з іншими FaaS-рішеннями

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

(Azure Functions, Google Cloud Functions), що також підтверджує конкурентоспроможність архітектури AWS у задачах авторизації.

У контексті кросплатформеності варто згадати дослідження [11], [12], у яких наголошується на перевагах побудови універсальних хмарних бекендів, здатних обслуговувати як мобільні, так і вебклієнти. Завдяки використанню RESTful API, функцій авторизації та уніфікованих форматів даних (JSON), можливим стає створення єдиного мікросервісу, що адаптується до потреб різних платформ.

Таким чином, огляд наявних досліджень демонструє, що поєднання AWS Lambda, Amazon Aurora та JWT-технологій є перспективним підходом для побудови масштабованих, безпечних і кросплатформених систем авторизації. Проте питання точного аналізу продуктивності та визначення оптимальних конфігурацій для подібних систем залишається відкритим, що й обумовлює актуальність даної статті.

Мета дослідження. Метою даної роботи є оцінка ефективності використання AWS Lambda для реалізації масштабованої системи авторизації користувачів з точки зору інтеграції з базою даних та кросплатформених клієнтів. У роботі також аналізується роль керованих сервісів у підвищенні безпеки, стабільності та продуктивності таких рішень. Результати цього дослідження можуть бути використані як основа для проєктування авторизаційних модулів у хмарних додатках, орієнтованих на мобільні платформи, вебінтерфейси та багатокомпонентні системи.

Виклад основного матеріалу дослідження.

Хмарні компоненти масштабованої авторизації користувачів

1. Архітектура системи

Масштабована авторизація у хмарному середовищі потребує узгодженої взаємодії між обчислювальними функціями, базою даних та засобами безпечного зберігання конфігурацій. У цьому дослідженні використано три основні компоненти платформи AWS: **AWS Lambda, Amazon Aurora PostgreSQL та AWS Systems Manager Parameter Store (SSM).** У цьому розділі описано їхню функціональну роль та технічну інтеграцію в межах авторизаційного модуля.

Архітектура реалізованої системи авторизації базується на поєднанні безсерверних обчислень, керованої бази даних та централізованого зберігання конфігурацій. Користувач надсилає облікові дані через HTTP-запит, який обробляє API Gateway. Запит викликає функцію AWS Lambda, яка отримує параметри з AWS Systems Manager Parameter Store, підключається до бази даних Amazon Aurora PostgreSQL, виконує перевірку облікових даних і генерує JWT-токен. Цей токен повертається клієнту у відповіді. Уся система побудована всередині AWS, що забезпечує високу доступність, масштабованість і безпеку.

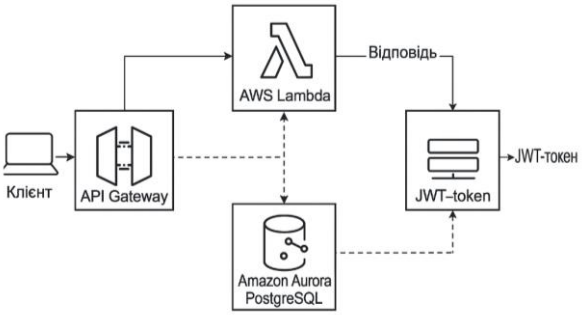

Рисунок 1 - Архітектура системи авторизації користувачів у AWS

На схемі зображено: клієнт (веб чи мобільний застосунок) надсилає запит через API Gateway до AWS Lambda. Lambda-функція взаємодіє з Amazon Aurora PostgreSQL для перевірки облікових даних та зберігає конфігурацію в SSM Parameter Store. Після успішної перевірки повертається JWT-токен.

2. AWS Lambda: виконання коду на вимогу

AWS Lambda є безсерверним сервісом, що дозволяє запускати функції у відповідь на події (наприклад, HTTP-запит від клієнта) без потреби керування інфраструктурою. У даній системі Lambda-функція виконує обробку запитів авторизації: зчитує облікові дані, перевіряє їх у базі даних та, у разі успіху, формує JWT-токен.

Оригінальний фрагмент функції авторизації має вигляд:

```
const { Client } = require('pg');
const jwt = require('jsonwebtoken');
const { getParameter } = require('./ssm');

exports.handler = async (event) => {
    const { username, password } = JSON.parse(event.body);

    const client = new Client({
        host: process.env.DB_HOST,
        user: process.env.DB_USER,
        password: await getParameter('dbPassword'),
        database: 'authdb',
    });
    await client.connect();

    const res = await client.query(
        'SELECT id, role FROM users WHERE username = $1 AND
password = crypt($2, password)',
        [username, password]
    );
    if (res.rows.length === 0) {
        return {
            statusCode: 401,
            body: 'Unauthorized',
```


ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

```
        };
    }

    const token = jwt.sign(
        { userId: res.rows[0].id, role: res.rows[0].role },
        process.env.JWT_SECRET,
        { expiresIn: '15m' }
    );

    return {
        statusCode: 200,
        body: JSON.stringify({ token }),
    };
};
```

Ця функція працює як незалежний обчислювальний мікросервіс, що не вимагає постійної активності й автоматично масштабується відповідно до навантаження.

3. Для зберігання облікових записів використовується **керована реляційна база даних Amazon Aurora PostgreSQL** — сумісна з PostgreSQL високопродуктивна база даних, що забезпечує автоматичне масштабування, резервне копіювання та низьку затримку. У контексті авторизації вона виконує перевірку імені користувача та пароля.

Для безпечного зберігання паролів використовується функція `crypt()` із модуля `pgcrypto`, що дозволяє зберігати хешовані паролі й перевіряти їх без розшифрування.

4. AWS Systems Manager Parameter Store — безпечне зберігання конфігурацій

Конфіденційна інформація, така як пароль до бази даних або секрет для підпису токенів, зберігається в **AWS SSM Parameter Store**. Це дозволяє уникнути прямого запису секретів у коді та централізовано керувати доступом до них.

Функція для доступу до параметрів виглядає так:

```
const AWS = require('aws-sdk');
const ssm = new AWS.SSM();
async function getParameter(name) {
    const result = await ssm.getParameter({
        Name: name,
        WithDecryption: true,
    }).promise();
    return result.Parameter.Value;
}
```

Таким чином, авторизаційна система спирається на безсерверне обчислення, централізоване зберігання секретів і керовану базу даних, що разом формують надійну, масштабовану та безпечну архітектуру.

Представлено огляд провідної хмарної платформи AWS, з акцентом на її безсерверні обчислювальні сервіси та керовані бази даних. AWS Lambda продемонструвала високу гнучкість і масштабованість для запуску подієорієнтованого коду без необхідності керування інфраструктурою. Особливо корисною є можливість автоматичного

масштабування, підтримка багатьох мов програмування, інтеграція з іншими сервісами AWS, підтримка приватних мереж і контейнеризації.

Amazon Aurora забезпечує надійне, продуктивне й масштабоване зберігання даних з автоматичною реплікацією, балансуванням навантаження на читання і високою доступністю. Її сумісність із MySQL/PostgreSQL, підтримка безсерверної конфігурації (Aurora Serverless v2) та розділення сховища й обчислювальних потужностей дозволяють легко інтегрувати її у складні масштабовані системи.

Таким чином, комбінація AWS Lambda та Aurora надає сучасну, ефективну архітектуру для реалізації продуктивних і витривалих додатків, зокрема в задачах авторизації та управління даними.

Експериментальна оцінка продуктивності AWS Lambda-функцій для авторизації користувачів

1. Вплив обсягу пам’яті на час і вартість виконання функції з алгоритмом Argon2

У цьому підрозділі проаналізовано залежність продуктивності AWS Lambda-функції, яка виконує хешування пароля за допомогою алгоритму Argon2, від обсягу виділеної пам’яті (ОП — обсяг оперативної пам’яті функції). Argon2 — сучасний криптографічний алгоритм, рекомендований для безпечного зберігання паролів, який використовується у функції авторизації для перевірки правильності облікових даних.

Для оцінки проведено серію експериментів з різними обсягами пам’яті — від 128 МБ до 2048 МБ.

Для кожної конфігурації виміряно:

– час виконання функції у випадку холодного старту (перше завантаження середовища),

– час виконання при теплому старті (повторний виклик без повторної ініціалізації),

– вартість виконання в обох випадках, розраховану за офіційною моделлю тарифікації AWS Lambda.

Аналіз часу виконання: холодний старт.

На рисунку 2 показано експоненційне зменшення часу виконання функції при збільшенні ОП. Для Argon2, який є ресурсоємним алгоритмом, низький обсяг пам’яті (128 МБ) призводить до неприйнятно високої затримки — понад 4 секунди. Збільшення обсягу до 2048 МБ скорочує час до 260 мс, що відповідає прискоренню в 15 разів.

Таким чином, використання малих обсягів пам’яті (<512 МБ) для криптографічних функцій значно погіршує продуктивність і не рекомендоване для авторизаційних сервісів реального часу.

Аналіз часу виконання: теплий старт.

Рисунок 3 демонструє аналогічну тенденцію: при повторному виклику функції (теплий старт) час виконання зменшується з 3185 мс до 183 мс. Хоча приріст менш відчутний через повторне використання ініціалізованого середовища, ефект залишається значним — до 17-кратного прискорення.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Висновок: для стабільно низької затримки при warm start рекомендовано використовувати обсяг пам’яті не менше 1024 МБ, оптимально — 2048 МБ.

Цікаво, що вартість виконання функції залишається майже незмінною для всіх рівнів ОП. Для холодного старту значення коливається в межах $8.4–8.7 \times 10^{-6}$, попри те, що пам’ять зростає у 16 разів. Це пояснюється тим, що при збільшенні ОП функція виконується значно швидше, тому зменшується час, а загальний тариф балансується.

Висновок: підвищення обсягу пам’яті майже не впливає на вартість, але суттєво покращує швидкодію, тому є доцільним навіть у бюджетних сценаріях.

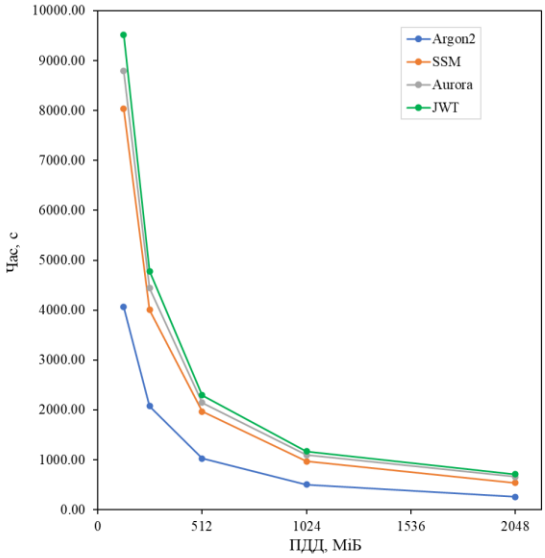

Рисунок 2 – Графік залежності холодного старту функцій від об’єму ОП

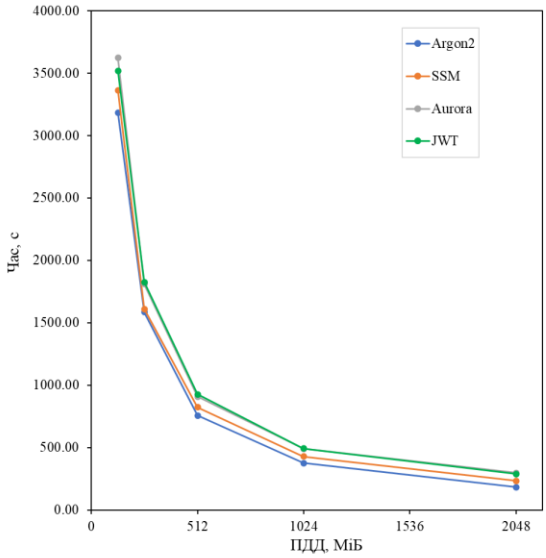

Рисунок 3 - Графік залежності теплого старту функцій від об’єму ОП

2. Продуктивність AWS Lambda при зверненні до SSM

У цьому підрозділі оцінюється вплив обсягу виділеної пам’яті на швидкодію та вартість функції AWS Lambda під час звернення до сервісу AWS Systems Manager Parameter Store (SSM). Цей компонент використовується у функції авторизації для безпечного отримання конфігураційних параметрів (наприклад, паролів до БД або секретних ключів). На відміну від обчислень у межах самої функції, звернення до SSM вимагає зовнішнього запиту, валідації доступу через IAM, дешифрування значення через KMS і передачі результату в Lambda, що значно збільшує загальну затримку.

Аналіз часу виконання (холодний старт).

Графік 4 демонструє типову експоненційну залежність: при 128 МБ функція виконується понад 8 секунд, при 2048 МБ — лише 540 мс. Це майже 15-кратне зниження часу відповіді. Однак абсолютні значення затримки вищі, ніж при хешуванні Argon2, оскільки запит до SSM включає мережеву взаємодію, авторизацію, розшифрування KMS і передачу відповіді.

Висновок: SSM — один із найбільш “важких” етапів авторизації. Зниження затримки можливе лише через збільшення обсягу пам’яті або попереднє кешування значень у коді функції.

Аналіз часу виконання (теплий старт).

Графік 4 також показує суттєве зменшення часу при теплому старті — з 3363 мс (128 МБ) до 233 мс (2048 МБ). Це свідчить, що навіть при повторному запуску функції продуктивність при зверненні до SSM суттєво залежить від виділеної пам'яті, хоч і меншою мірою, ніж при ініціалізації середовища.

Висновок: навіть при warm start рекомендовано використовувати обсяг пам'яті не менше 1024 МБ, щоб забезпечити прийнятну затримку в авторизаційних API.

Оцінка вартості виконання.

Вартість для викликів до SSM є найвищою серед усіх досліджуваних етапів. У середньому вона становить від 16 до 18 $\times 10^{-6}$ USD при холодному старті (рисунок 5). Це пов'язано не лише з тривалістю виконання, а й з тим, що звернення до SSM активує допоміжні сервіси, зокрема KMS і IAM. Навіть при теплому старті вартість зменшується незначно: наприклад, при 2048 МБ — лише з 18.0 до 7.7 $\times 10^{-6}$ USD.

Висновок: через високу вартість і затримку використання SSM варто обмежити лише критично важливими параметрами, інші - кешувати в середині функції або зберігати у вигляді зашифрованих шарів (Lambda layers).

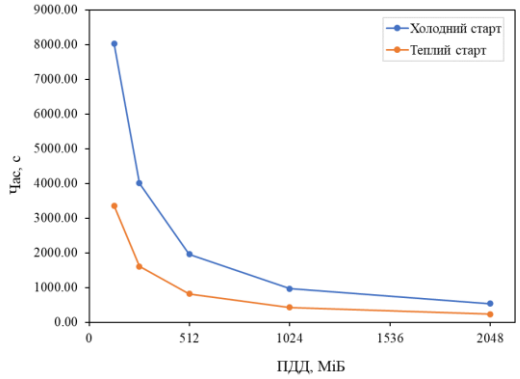

Рисунок 4 – Залежність швидкості виконання функції від об'єму ОП

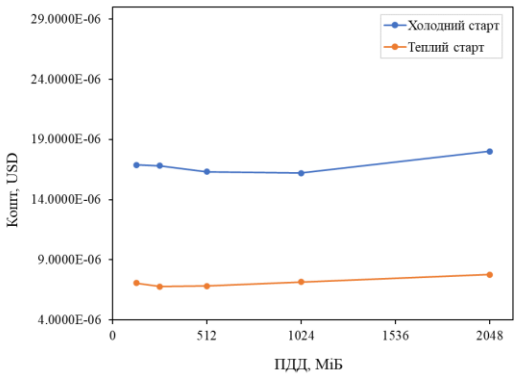

Рисунок 5 – Графік залежності кошту виконання функції від об'єму ОП

3. Продуктивність запитів до бази даних Aurora PostgreSQL

У цьому експерименті AWS Lambda-функція виконує підключення до бази даних Aurora PostgreSQL і виконує запит SELECT для перевірки облікових даних користувача. Aurora працює в режимі Serverless v2 в одній зоні доступності. Мета експерименту - оцінити, як зміна обсягу пам'яті впливає на час підключення, виконання запиту та загальну затримку, а також вартість функції.

Таблиця 2

Залежність швидкості виконання функції від об'єму ОП

ОП	Холодний старт		Теплий старт	
	Час, мс	Кошт, USD	Час, мс	Кошт, USD
128	8790.83	18.4607E-06	3623.36	7.6090E-06
256	4439.54	18.6461E-06	1810.59	7.6045E-06
512	2146.18	17.8133E-06	908.69	7.5422E-06
1024	1095.66	18.2976E-06	489.77	8.1791E-06
2048	658.69	21.9344E-06	297.44	9.9049E-06

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Аналіз результатів показує, що зі збільшенням об’єму ОП час виконання функції значно скорочується в обох сценаріях. У випадку холодного старту функція працює повільніше, але навіть тоді приріст ОП забезпечує зменшення затримки у 13 разів (від ~8790 мс до ~659 мс). У теплому старті зменшення часу виконання ще більш ефективне - з ~3623 мс до ~297 мс.

Водночас вартість виконання функції залишається практично стабільною, коливаючись у межах одного порядку (близько 8–9 $\times 10^{-6}$ USD). Це свідчить про те, що збільшення об’єму пам’яті є доцільним, оскільки забезпечує значне покращення продуктивності без пропорційного зростання витрат.

4. Генерація JWT-токена: продуктивність і витрати

Суть експерименту: JSON Web Token (JWT) використовується для передачі авторизаційної інформації між клієнтом і сервером. Lambda-функція після успішної перевірки облікових даних формує токен з підписом, який зберігає ID користувача, роль, час створення та строк дії. Генерація токена включає криптографічне шифрування (HMAC або RSA), серіалізацію JSON і повернення відповіді через API Gateway. Мета експерименту - визначити, як обсяг пам’яті, виділений функції (ОП), впливає на час генерації JWT та її вартість.

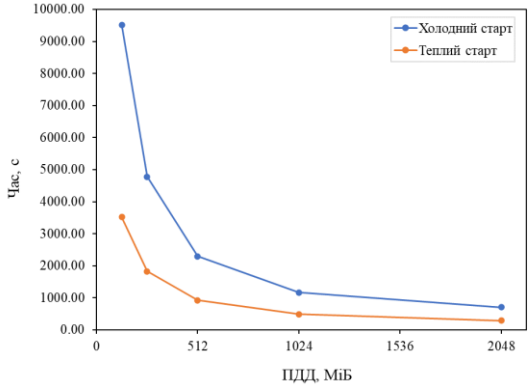

Рисунок 6 - Залежність швидкості виконання функції від об’єму ОП

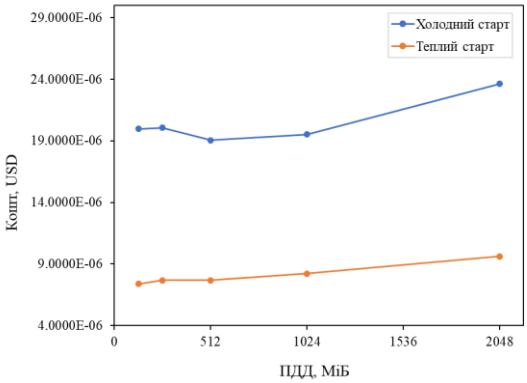

Рисунок 7 - Графік залежності кошту виконання функції від об’єму ОП

Графік на рисунку 6 демонструє, що зі збільшенням ОП час генерації токена суттєво зменшується як у випадку холодного старту, так і теплого. При холодному старті спостерігається зниження часу з ~9518 мс (128 МБ) до ~710 мс (2048 МБ), тобто у понад 13 разів. При теплому старті - з ~3520 мс до ~289 мс, що є майже 12-кратним покращенням.

Вартість виконання функції, яка представлена на рисунку 7, залишається стабільною - вона не зростає лінійно з обсягом пам’яті, а коливається в межах одного порядку: приблизно від 7 до 10 $\times 10^{-6}$ USD. Це свідчить про ефективність масштабування: більший ОП дозволяє досягти значно меншого часу виконання без істотного збільшення витрат.

Таким чином, для генерації JWT найбільш доцільним є використання ОП у діапазоні 1024–2048 МБ, що забезпечує мінімальні затримки з незначним впливом на загальну вартість функції. Як результат, усі 4 компоненти (argon2, SSM, БД, JWT) підтвер-

джують, що збільшення пам’яті значно знижує час виконання, майже не впливаючи на вартість при теплому старті.

Нижче наведена зведена таблиця, яка порівнює **усі 4 етапи** виконання Lambda-функції авторизації за ключовими метриками: час (мс) і вартість ($), як для холодного, так і для теплого старту, при максимальній конфігурації (2048 МБ).

Таблиця 3

Продуктивність етапів Lambda-функції (2048 МБ)

Етап виконання	Холодний старт (мс)	Теплий старт (мс)	Вартість холодного старту ($)	Вартість теплого старту ($)
Argon2	260.35	183.46	8.6697E-06	6.1093E-06
SSM	540.55	233.25	1.8003E-05	7.7688E-06
База даних	658.18	312.14	2.1921E-05	1.0392E-05
JWT	709.47	289.62	2.3625E-05	9.6452E-06

Висновки зі зведеної таблиці:

– Найшвидший компонент: Argon2 при теплому старті — лише 183 мс.

– Найповільніший при холодному старті: JWT — понад 700 мс, через ініціалізацію криптографії.

– Найдорожчий компонент: JWT при холодному старті — $\$23.6 \times 10^{-6}$.

– Найменш чутливий до ПДД: SSM, бо затримки більше залежать від зовнішніх викликів.

– Оптимальна стратегія: для реального продакшн-навантаження варто використовувати provisioned concurrency або контейнери Lambda з попередньо ініціалізованими залежностями, і при цьому встановити пам’ять не менше 1024–2048 МБ.

Висновки. У результаті експериментальної оцінки продуктивності Lambda-функції авторизації було досліджено чотири ключові етапи: хешування паролю (argon2), звернення до SSM, взаємодію з базою даних PostgreSQL та генерацію JWT. Проведено порівняння часу виконання та вартості для сценаріїв холодного й теплого стартів при обсязі пам’яті 2048 МБ.

Найдовшим етапом при холодному старті виявився генератор токенів JWT (понад 700 мс), що пов’язано з ініціалізацією криптографічних залежностей. Найшвидше виконання спостерігалося для Argon2 при теплому старті. Вартісний аналіз підтвердив, що функція з обраною конфігурацією залишається економічно ефективною, однак значно виграє від використання "теплих" екземплярів.

Загалом, результати свідчать про доцільність застосування AWS Lambda для авторизаційних задач, особливо в поєднанні з керованими сервісами AWS. Найкраща продуктивність досягається при використанні високої конфігурації пам’яті (1024–2048 МБ) та попередньо виділених екземплярів (provisioned concurrency). Це дозволяє зменшити затримки, уникнути непередбачуваних холодних стартів і забезпечити стабільну масштабовану роботу сервісу авторизації.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

У статті було досліджено можливості використання безсерверної архітектури AWS Lambda у поєднанні з керованою базою даних Amazon Aurora для реалізації масштабованої системи авторизації користувачів. Проведений аналіз підтверджує, що таке поєднання дозволяє досягти високої продуктивності, автоматичного масштабування, гнучкого управління ресурсами та зниження витрат на інфраструктуру. Завдяки підтримці різних мов програмування та архітектур, AWS Lambda добре інтегрується з кросплатформеними застосунками, зокрема мобільними, веб- і IoT-рішеннями. Aurora, у свою чергу, забезпечує стабільну та масштабовану реляційну базу даних з високим рівнем доступності, що критично важливо для обробки користувацьких запитів і зберігання облікових даних.

Проведене дослідження підтвердило ефективність використання безсерверної архітектури AWS Lambda для реалізації сучасної авторизації користувачів у хмарному середовищі. Результати експериментів продемонстрували, що правильно підібрані параметри середовища, зокрема обсяг оперативної пам’яті, суттєво впливають на швидкодію функцій без значного збільшення вартості їх виконання. Інтеграція з керованою базою даних Amazon Aurora та безпечним зберіганням конфігурацій у SSM забезпечує надійність, гнучкість і масштабованість системи.

Отримані висновки можуть бути використані як основа для проектування продуктивних, економічно ефективних та кросплатформених авторизаційних сервісів у реальних виробничих середовищах. Це відкриває перспективи подальшого дослідження в напрямі автоматичного масштабування, оптимізації витрат і підвищення стійкості хмарних архітектур до навантажень нового покоління.

ЛІТЕРАТУРА

1. Baldini, I., Castro, P., Chang, K. et al. Serverless computing: Current trends and open problems. *Research Advances in Cloud Computing*, 2017, pp. 1–20.

2. Spillner, J., Mateos, C., Monge, D. A. FAASCAPES: towards a taxonomy for serverless computing in cloud environments. *IEEE Cloud Computing*, 2019, 6(5), pp. 48–57.

3. Wu, H., Shah, A., Gupta, R. Aurora: Design and Performance of the Next-Generation MySQL-Compatible Cloud Database. *AWS re:Invent Conference*, 2017.

4. Baeza-Yates, R. et al. Scalable Databases in the Cloud: Data Models and Performance. *Proceedings of the VLDB Endowment*, 2018, 11(12), pp. 2106–2109.

5. AWS Documentation. AWS Systems Manager Parameter Store – Best Practices. [Online]. Available: https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-parameter-store.html

6. Jones, M., Bradley, J., and Sakimura, N. JSON Web Token (JWT). *IETF RFC 7519*, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519

7. Choudhury, O. et al. Enabling secure and scalable authentication with JWT in distributed systems. *Journal of Cloud Computing*, 2020, 9(1), p. 24.

8. Wang, L., Zhang, M., Ristenpart, T., Swift, M. D. Peeking Behind the Curtains of Serverless Platforms. *USENIX ATC*, 2018, pp. 133–146.

9. McGrath, G., Brenner, P. Serverless computing: Design, implementation, and performance. *IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2017.
10. Spillner, J. Comparing FaaS Performance: AWS Lambda, Azure Functions, and Google Cloud Functions. *arXiv preprint*, arXiv:2003.04867, 2020.
11. Richards, M. *Software Architecture Patterns*. O'Reilly Media, 2015. (Chapter on Microservices and Backend for Frontend).
12. Taivalsaari, A., Mikkonen, T. A Roadmap to Web Applications: From Web 1.0 to Web 4.0. *Web Information Systems and Technologies*, 2017, pp. 318–336.

REFERENCES

1. Baldini, I., Castro, P., Chang, K. et al. Serverless computing: Current trends and open problems. *Research Advances in Cloud Computing*, 2017, pp. 1–20.
2. Spillner, J., Mateos, C., Monge, D. A. FAASCAPES: towards a taxonomy for serverless computing in cloud environments. *IEEE Cloud Computing*, 2019, 6(5), pp. 48–57.
3. Wu, H., Shah, A., Gupta, R. Aurora: Design and Performance of the Next-Generation MySQL-Compatible Cloud Database. *AWS re:Invent Conference*, 2017.
4. Baeza-Yates, R. et al. Scalable Databases in the Cloud: Data Models and Performance. *Proceedings of the VLDB Endowment*, 2018, 11(12), pp. 2106–2109.
5. AWS Documentation. AWS Systems Manager Parameter Store – Best Practices. [Online]. Available: https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-parameter-store.html
6. Jones, M., Bradley, J., and Sakimura, N. JSON Web Token (JWT). *IETF RFC 7519*, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519
7. Choudhury, O. et al. Enabling secure and scalable authentication with JWT in distributed systems. *Journal of Cloud Computing*, 2020, 9(1), p. 24.
8. Wang, L., Zhang, M., Ristenpart, T., Swift, M. D. Peeking Behind the Curtains of Serverless Platforms. *USENIX ATC*, 2018, pp. 133–146.
9. McGrath, G., Brenner, P. Serverless computing: Design, implementation, and performance. *IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2017.
10. Spillner, J. Comparing FaaS Performance: AWS Lambda, Azure Functions, and Google Cloud Functions. *arXiv preprint*, arXiv:2003.04867, 2020.
11. Richards, M. *Software Architecture Patterns*. O'Reilly Media, 2015. (Chapter on Microservices and Backend for Frontend).
12. Taivalsaari, A., Mikkonen, T. A Roadmap to Web Applications: From Web 1.0 to Web 4.0. *Web Information Systems and Technologies*, 2017, pp. 318–336.

Received 20.02.2026
Accepted 27.02.2026
Published 31.03.2026

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Performance evaluation of AWS Lambda for scalable and secure user authorization in cross-platform cloud applications

The paper examines the problem of designing scalable and secure user authorization systems in cloud environments using serverless computing. The relevance of this research is driven by the increasing demand for high-performance services that remain cost-effective and reliable under varying workloads. Traditional server-based solutions no longer fully address these requirements, which has led to the growing adoption of Function-as-a-Service (FaaS) models. AWS Lambda was selected as the core platform, offering automatic scaling, event-driven execution, and a pay-per-use model. For credential validation, the architecture integrates with Amazon Aurora PostgreSQL, a high-availability managed database compatible with PostgreSQL, while AWS Systems Manager Parameter Store ensures secure storage and centralized management of configuration parameters and secrets.

The proposed architecture processes user credentials through an API Gateway that triggers Lambda functions, validates the input against Aurora, and generates JSON Web Tokens (JWT) to authorize user access. Experimental evaluation focused on the effects of allocated memory, cold versus warm starts, and calls to external services on execution time and overall cost. The results revealed an exponential decrease in response latency when memory was increased from 128 MB to 2048 MB: execution time dropped from several seconds to under one second, while costs remained nearly constant. Aurora queries demonstrated stable low-latency performance, JWT generation reached optimal efficiency at 1024–2048 MB, and the Parameter Store introduced the highest delays, indicating the importance of caching and optimized secret management.

The findings confirm the feasibility of AWS Lambda as the foundation for cloud-based authorization services. The combination of Lambda, Aurora, and Parameter Store creates a robust, secure, and cost-efficient architecture that adapts dynamically to workload changes. The proposed approach is particularly relevant for cross-platform mobile and web applications, where scalability, data security, and low response time are critical. Future research should focus on optimizing cold start mitigation strategies, improving cost-performance trade-offs, and enhancing the resilience of serverless architectures under next-generation workloads.

Гунченко Юрій Олександрович – зав. кафедри комп'ютерних систем та технологій, д.т.н., професор, Одеський національний університет імені І.І.Мечникова.
ORCID: https://orcid.org/0000-0003-4423-8267.

Камєнєв Кирило Ігорович – магістр кафедри математичного забезпечення комп'ютерних систем, доктор філософії в галузі транспорту, Одеський національний університет імені І.І.Мечникова.
ORCID: https://orcid.org/0000-0002-9200-9496

Камєнєва Алла Вікторівна – доцент кафедри кафедри комп'ютерних систем та технологій, к.т.н., Одеський національний університет імені І.І.Мечникова.
ORCID: https://orcid.org/0000-0002-9970-9081

Єпік Марина Олександрівна – доцент кафедри кафедри комп'ютерних систем та технологій, к.т.н., Одеський національний університет імені І.І.Мечникова.
ORCID: https://orcid.org/0000-0001-9021-3680

Gunchenko Yurii – head of the department of computer systems and technologies, doctor of technical sciences, professor, Odesa I.I. Mechnikov National University.
ORCID: https://orcid.org/0000-0003-4423-8267.
Kamieniev Kyrylo – master of the department of mathematical support of computer systems, phd in transport, Odesa I.I. Mechnikov National University.
ORCID: https://orcid.org/0000-0002-9200-9496
Kamienieva Alla – associate professor of the department of computer systems and technologies, phd in technical sciences, Odesa I.I. Mechnikov National University.
ORCID: https://orcid.org/0000-0002-9970-9081
Iepik Maryna – associate professor of the department of computer systems and technologies, phd in technical sciences, Odesa I.I. Mechnikov National University;
ORCID: .https://orcid.org/0000-0001-9021-3680

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)