DOI 10.34185/1562-9945-3-152-2024-13
УДК 004.021

Б.В. Молодець, Т.М. Булана

АНАЛІЗ ІСНУЮЧИХ АРХІТЕКТУР ДЛЯ РОЗРОБКИ СИСТЕМИ ОЦІНКИ ЯКОСТІ ПОВІТРЯ

Анотація: Робота присвячена існуючих архітектур для розробки системи оцінки якості повітря: serverless архітектура, мікросервісна архітектура, монолітна архітектура та сервісно орієнтова архітектура. В рамках роботи описані переваги та недоліки цих архітектур, наведені можливі схеми реалізації. В якості провайдера хмарних послуг було обрано Amazon Web Service. У результаті аналізу було виявлено, що найбільш підходящою є сервісно орієнтовна архітектура. Хоча мікросервісна архітектура набула широкого визнання в останні роки, сервісна орієнтована архітектупа має свої переваги, особливо для поточної інформаційної системи: більша взаємодія сервісів, більша єдність даних та стандартизація, більш просте управління сервісами, менша складність інфраструктури.
Ключові слова: інформаційна система, моніторинг якості повітря, архітектура програм ного забезпечення, SOA, MSA, serverless architecture, монолітна архітектура, AWS

Інформаційні системи моніторингу є важливим інструментом у боротьбі з забрудненням повітря. Сучасні технології дозволяють в режимі реального часу збирати та аналізувати дані про якість повітря, що надає можливість оперативно реагувати на забруднення та вживати заходів для його усунення.

При проектуванні будь-якої інформаційної системи або програмного продукту, одним з ключових аспектів є вибір відповідної архітектури. Архітектура системи визначає спосіб організації компонентів, взаємодії між ними та загальну структуру системи. Правильний вибір архітектури може значно вплинути на продуктивність, масштабованість, безпеку та інші характеристики системи.

Вибір архітектури програмного забезпечення є ключовим етапом у процесі розробки будь-якого проекту. Коректна обрана архітектура визначає ефективність, гнучкість та масштабованість системи.

Перш за все, варто визначити основні вимоги до системи, такі як швидкість розробки, масштабованість, надійність, зручність у використанні тощо. На основі цих вимог ми можемо зрозуміти, яка архітектура буде

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

найбільш відповідати нашим потребам. Для інформаційної системи оцінки якості повітря було сформульовані наступні вимоги:

– Архітектура має забезпечити модульність, гнучкість та швидкість розгортання та розвитку

– Забезпечення інтегрованості та взаємодії різних компонентів системи за допомогою сервісів, що сприяє легкості розширення.

– Можливість розгортання системи у хмарних сервісах для забезпечення масштабованості, доступності та безпеки даних.

– Розділення системи на логічні шари (наприклад, шари доступу до даних, бізнес-логіки та презентації) для забезпечення зручності розробки та підтримки.

Важливо також пам'ятати, що архітектура може еволюціонувати протягом часу, тому варто мати можливість адаптувати її до змін у вимогах та умовах проекту.

Serverless архітектура - це підхід до розробки програмного забезпечення, де розробник не має безпосереднього контролю над серверами, на яких запускається програмне забезпечення. Замість того, щоб керувати інфраструктурою, розробник працює з окремими функціями (функціями як сервісом, Function as a Service - FaaS), які викликаються відповідно до потреб системи [1].

Розробник пише і розгортає окремі функції, які викликаються лише при необхідності, без необхідності управління інфраструктурою. Можна розрізнити кілька парадигм безсерверних обчислень:

– Функція як послуга (FaaS), реалізована, наприклад, за допомогою AWS Lambda (Hendrickson, et al., 2016),

– Програмне забезпечення як послуга (SaaS) через Google Cloud

– База даних як послуга (DBaaS), як у випадку з Microsoft Azure для PostgreSQL.

FaaS можна розглядати як гібрид платформи як послуги (PaaS) та програмного забезпечення як послуги (SaaS): інфраструктура, а також інформація повністю контролюється хмарним постачальником, навіть якщо програмне забезпечення управляється клієнтами або розробниками (Korhonen, 2017) (Kumari, Sahoo, Behera, Misra, Sharma, 2021). Системи FaaS або "Функція як послуга" дозволяють розробникам створювати та розміщувати додатки (що складаються з однієї функції або набору функцій), які динамічно масштабуються. Ці додатки призначені для виконання однієї функції на вимогу, а потім

відкладають запуск до наступного разу, коли вона знадобиться. Такі системи забезпечують величезну економію коштів для розробників додатків, дозволяючи їм платити лише за ті екземпляри, на яких виконуються їхні безсерверні застосунки [2].

Так як для розробки системи було обрано платформу Amazon Web Services то для реалізації цієї архітектури можна було б скористатися наступним: AWS Lambda - це сервіс для виконання коду без необхідності управління інфраструктурою. Ви можете завантажувати свою функцію до Lambda та викликати її відповідно до потреб вашого додатку, Amazon API Gateway - сервіс для створення, публікації, підтримки, моніторингу та захисту API (можна використовувати API Gateway для створення HTTP або WebSocket API для виклику ваших Lambda функцій через інтернет), Amazon DynamoDB - повністю керована NoSQL база даних, яка забезпечує швидкий та надійний доступ до даних для вашого додатку (можна використовувати DynamoDB для зберігання та керування даними, які обробляються вашими Lambda функціями) та інші (рисунок 1).

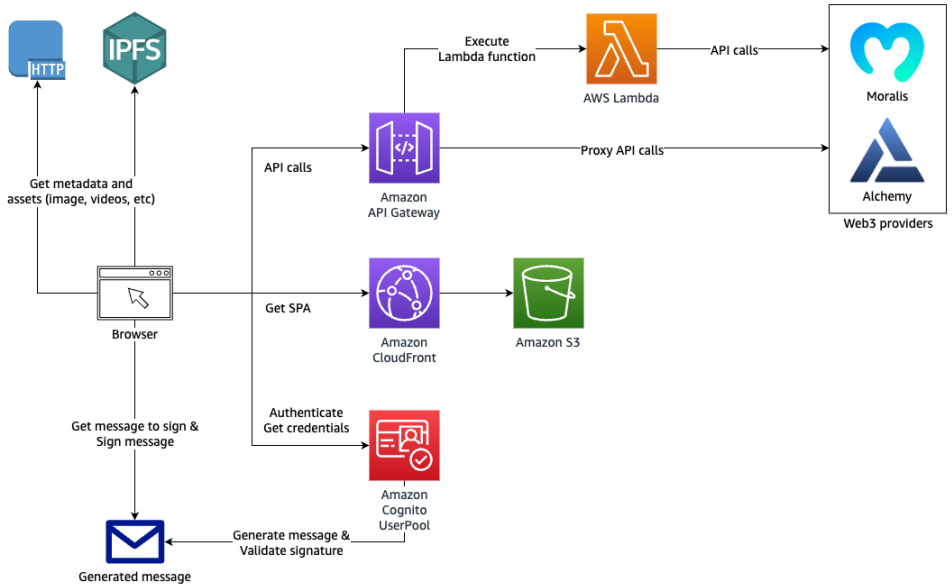

Рисунок 1 - Схема Serverless архітектури для розгортання на Amazon Web Services [3]

Використання цієї архітектури надає наступні переваги:

– Масштабованість - Serverless дозволяє автоматично масштабувати ресурси в залежності від навантаження, що робить його ідеальним для проектів зі змінним навантаженням;

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

– Швидкість розгортання - Serverless дозволяє швидко розгортати функції без необхідності управління інфраструктурою.

Але у Serverless архітектури є свої недоліки:

– Обмеження функцій - Існують обмеження на тривалість виконання функцій, розмір пам'яті та інші обмеження, які можуть бути несумісними з деякими типами задач (в тому числі задач по розрахунку якості повітря);

– Складність відлагодження: Відлагодження serverless додатків може бути складнішим через їх розподілений характер.

Так як в нашої системи регулярно працюватиме збір даних з різних ресурсів та є велика кількість складних розрахунків (таких як запуск моделі CALLPUFF) [4] всі переваги цієї архітектури нівелюються.

Альтернативою Serverless архітектури може стати мікросервісна архітектура. Мікросервісна архітектура - це підхід до розробки програмного забезпечення, де програмний додаток розбивається на невеликі, самостійні модулі, відомі як мікросервіси. Кожен мікросервіс виконує конкретну функцію та має власний набір інтерфейсів API для взаємодії з іншими мікросервісами. Мікросервіси взаємодіють між собою через API, що дозволяє їм співпрацювати та обмінюватися даними. Кожен мікросервіс може використовувати власний стек технологій, що дозволяє використовувати найкращі інструменти для конкретних завдань. Якщо один мікросервіс відмовляє, це не призводить до відмови всього додатку, а лише до проблем в окремій функціональності [5].

Для розгортання мікросервісної архітектури на платформі Amazon Web Services можна скористатися наступним: сервісами контеризації Amazon Elastic Compute Cloud (EC2) або Amazon Elastic Kubernetes Service (EKS) , які дозволяють запускати та керувати контейнерами для ваших мікросервісів, Amazon Simple Storage Service (S3) в якості сховища для зберігання об’єктів , такі як файли, зображення та інше, Amazon CloudFront в якості швидкому та ефективному доставці вмісту веб-застосунку, Amazon ALB для балансування навантаження (дозволяє розподілити трафік між різними екземплярами застосунку або мікросервісаи в залежності від правил марщрутизації), Amazon DynamoDB та Amazon Aurora як нереляційну та реляційну бази даних відповідно та Amazon ElastiCache для підвищеня продуктивність та масштабованість застосунку шляхом кешування даних в оперативній пам'яті та забезпечуючи швидкий доступ[6]. Схема зображена на рисунку 2.

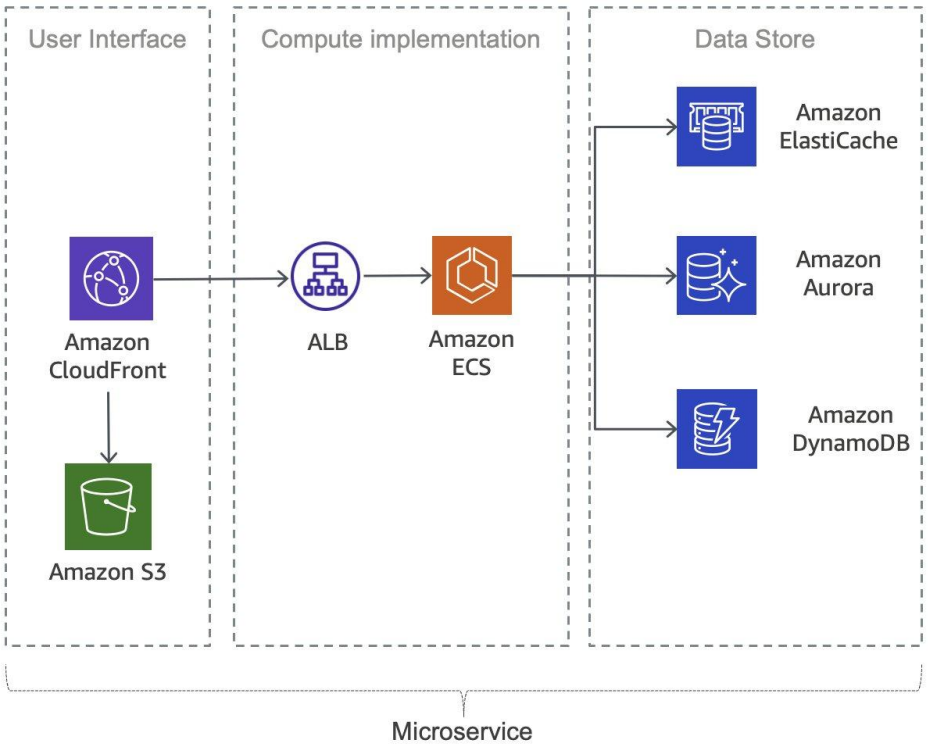

Рисунок 2 - Схема мультисервісної архітектури для розгортання на Amazon Web Services [7]

До переваг мікросервісної архітектури можна віднести наступне:

– Гнучкість: Мікросервісна архітектура дозволяє розділити додаток на невеликі, незалежні сервіси, що надає більшу гнучкість у розробці, тестуванні та впровадженні нових функцій;

– Швидкість розгортання: Можливість незалежного розгортання кожного сервісу дозволяє прискорити процес впровадження змін та оновлень.

– Масштабованість: Кожен сервіс може бути масштабований незалежно, що дозволяє оптимізувати ресурси та витрати.

До недоліків мікросервісної архітектури можна віднести наступні пункти:

– Складність управління конфігурацією: З більшою кількістю сервісів у системі стає складніше керувати конфігураціями та налаштуваннями. Доведеться вирішити питання версіонування, розгортання та управління налаштуваннями кожного сервісу окремо.

– Складність моніторингу та відлагодження: З більшою кількістю сервісів також ускладнюється моніторинг їх роботи та відлагодження випадків неполадок. Доведеться впроваджувати системи моніторингу та журналювання для кожного сервісу окремо.

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

– Системні проблеми із залежностями: Більше сервісів означає більше залежностей між ними. Якщо один сервіс стає недоступним або працює некоректно, це може вплинути на інші сервіси, що може призвести до каскадного ефекту збоїв.

– Проблема інтеграції та управління: Інтеграція між сервісами може вимагати значних зусиль у вирішенні питань сумісності, забезпеченні безпеки та забезпеченні спільного управління.

– Обмін даними. Зменшується швидкодія системи через накладні розходи на мережеві виклики між сервісами (в тому числі при обміні даними).

Ще одною не менш популярною з точки зору використання в проектах можна назвати монолітну архітектуру. Монолітна архітектура - це тип архітектури програмного забезпечення, де весь програмний продукт розробляється як єдине ціле, зазвичай у вигляді одного великого додатку. Основна ідея полягає в тому, що всі компоненти додатку взаємодіють один з одним без зовнішніх інтерфейсів або середовищ виконання. Основні характеристики монолітної архітектури включають:

– Цілісність - усі компоненти додатку розглядаються як єдине ціле, розробка, впровадження та масштабування ведуться разом.

– Внутрішня взаємодія - всі частини додатку спілкуються одна з одною напряму, без використання мережевих запитів або сервісних викликів.

– Монолітний стек технологій - у монолітній архітектурі використовується єдиний стек технологій для всіх компонентів.

– Простота розгортання і масштабування - так як застосунок складається з одного компонента, розгортання і масштабування може бути простішим порівняно з розподіленими системами.

До недоліків можна віднести складність масштабування. Монолітні додатки можуть бути важко масштабувати, оскільки всі їх компоненти взаємодіють тісно між собою. При збільшенні навантаження доводиться масштабувати всю архітектуру, навіть якщо лише деякі компоненти потребують додаткових ресурсів [8].

Альтернативним рішенням, яке було обрано для розробки інформаційної системи, стала сервісно-орієнтовна архітектура (SOA).

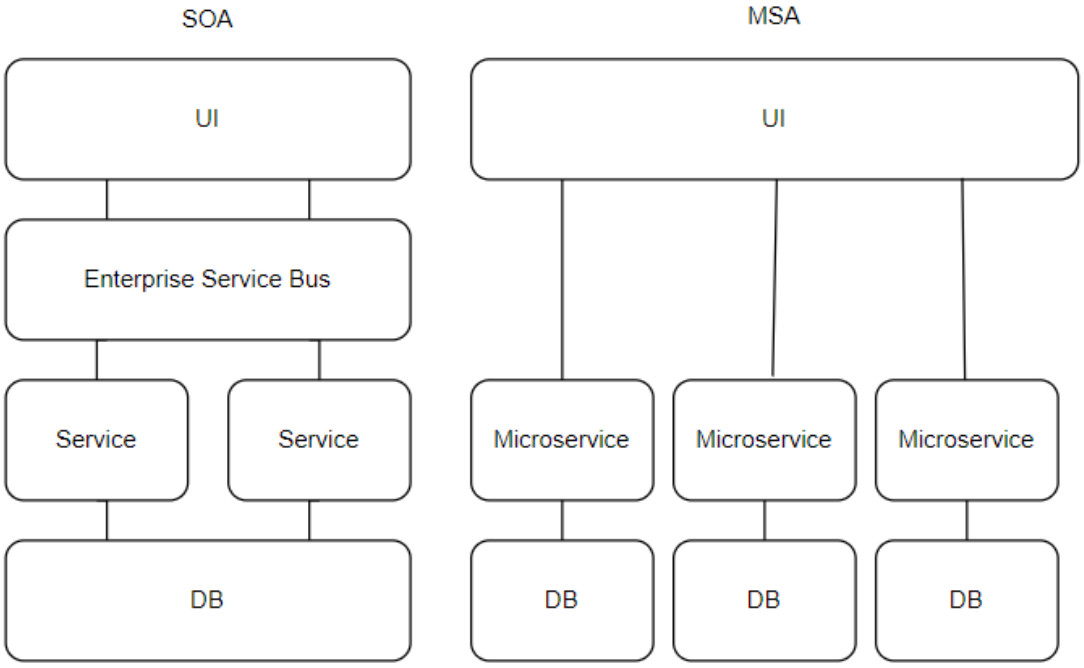

Рисунок 3 - Схеми Сервісно-орієнтованої та Мікросервісної архітектури

Це архітектурний підхід до розробки програмного забезпечення, в якому функціональність застосунків розглядається як набір послуг, які можуть бути використані та взаємодіяти між собою за допомогою стандартних протоколів. Основна ідея SOA полягає в тому, щоб розділити функціональність додатків на окремі сервіси, які можуть бути реалізовані та підтримувані незалежно один від одного [9].

У сервісно-орієнтовній архітектурі функціональність додатків розбивається на окремі сервіси, які можуть бути незалежними один від одного. Кожен сервіс виконує конкретну функцію і надає зовнішній інтерфейс для взаємодії з іншими сервісами. Кожен сервіс має ясно визначені межі та відокремлений від інших сервісів. Це дозволяє сервісам бути більш автономними та менш залежними від змін у інших частинах системи. Сервісно-орієнтовна архітектура дозволяє легко інтегрувати нові сервіси з існуючими системами, що дозволяє побудувати додатки з високим рівнем гнучкості та розширюваності. Завдяки розділенню функціональності на окремі сервіси, сервісно-орієнтовна архітектура дозволяє краще масштабувати систему, оскільки можна масштабувати тільки ті сервіси, які потребують додаткових ресурсів [10].

Основними відмінностями сервісно-орієнтовна архітектура від мікросервісної архітектури є наступні пункти:

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

– Сервісно-орієнтовна архітектура зазвичай створює більш великі та важкі сервіси, які можуть містити різноманітну функціональність; Мікросервісна архітектура, навпаки, розглядає функціональність як набір дрібних, відокремлених сервісів, які керуються принципом "один сервіс - одна функція".

– У сервісно-орієнтовній архітектурі, великі сервіси можуть мати складні залежності один від одного, що може робити їх важко масштабувати та підтримувати. У мікросервісній архітектурі, кожен сервіс незалежний і може бути масштабованим та підтримуваним окремо.

– У сервісно-орієнтовній архітектурі сервіси можуть використовувати різні протоколи та стандарти для взаємодії між собою. У мікросервісній архітектурі, зазвичай використовуються легкі та стандартні протоколи, такі як HTTP або REST, для взаємодії між сервісами.

– У сервісно-орієнтовній архітектурі керування багатьма великими сервісами може бути складним завданням. У мікросервісній архітектурі, хоча кількість сервісів може бути більшою, кожен сервіс менший і простіший у розгортанні та управлінні.

Таблиця 1

Таблиця відмінностей сервісно-орієнтовної архітектури від мікросервісної архітектури

	Сервісно-орієнтовна архітектура	Мікросервісна архітектура
Реалізації	Різні сервіси зі спільними ресурсами.	Самостійні та спеціалізовані невеликі сервіси.
Підключення	ESB використовує кілька протоколів обміну повідомленнями, таких як SOAP, AMQP і MSMQ.	API, Java Message Service, Pub/Sub
Сховище даних	Спільне зберігання даних.	Незалежне зберігання даних.
Розгортання	Впровадження змін вимагає повної перебудови.	Кожен мікросервіс можна зберігати в контейнерах.
Можливість багаторазового використання	Повторно використовує сервіси через спільні ресурси.	Кожен сервіс має свої незалежні ресурси. Мікросервіси можна повторно використовувати за допомогою їх API
Гнучке управління	Узгоджене управління даними у всіх сервісах.	Різні політики керування даними для кожного сховища.

Хоча мікросервісна архітектура набула широкого визнання в останні роки, сервісна орієнтована архітектупа має свої переваги, особливо в певних випадках та для поточної інформвційної системи:

– Більша взаємодія сервісів: У SOA, сервіси зазвичай взаємодіють між собою більш глибоко та в тіснішому співробітництві, оскільки вони можуть мати більшу кількість спільних елементів та функціональних можливостей.

– Більша єдність даних та стандартизація: SOA сприяє розробці спільних стандартів для обміну даними між сервісами, що дозволяє забезпечити більшу єдність даних та керування ними в різних частинах системи.

– Більш просте управління сервісами: У SOA, управління меншою кількістю великих сервісів може бути простішим у порівнянні з управлінням великою кількістю дрібних мікросервісів, які потребують більшої уваги до деталей.

– Менша складність інфраструктури: SOA може мати меншу загальну складність інфраструктури порівняно з мікросервісною архітектурою, оскільки кількість сервісів може бути меншою, але кожен сервіс може бути складним та функціонально багатшим.

У контексті розробки системи моніторингу якості повітря, сервісно-орієнтована архітектура відіграють значущу роль. Вона сприяє високій модульності та зручному взаємодії між компонентами системи. Враховуючи швидко зростаючі технології та вимоги до систем оцінки якості повітря, використання сервісно-орієнтованої архітектур дозволить створити динамічну та ефективну систему, здатну адаптуватися до змінних умов та вимог користувачів.

ЛІТЕРАТУРА

1. Lakhai V., Bachynskyy R. (2021). Investigation of Serverless Architecture. Advances in Cyber-Physical Systems. 6. 134-139. DOI: 10.23939/acps2021.02.134.

2. Kumari A., Sahoo B. (2022). Serverless Architecture for Healthcare Management Systems. DOI: 10.4018/978-1-6684-4580-8.ch011.

3. dApp authentication with Amazon Cognito and Web3 proxy with Amazon API Gateway [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/blogs/architecture/dapp-authentication-with-amazon-cognito-and-web3-proxy-with-amazon-api-gateway/

4. Molodets B., Hnatushenko V., Boldyriev D., Bulana T. Information System of Air Quality Assessment Using Data Interpolation from Ground Stations. MoMLeT+DS

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

2023: 5th International Workshop on Modern Machine Learning Technologies and Data Science, June 3, 2023, Lviv, Ukraine. P.233-245
5. Dragoni, N. et al. (2017). Microservices: Yesterday, Today, and Tomorrow. In: Mazzara, M., Meyer, B. (eds) Present and Ulterior Software Engineering. Springer, Cham. DOI: 10.1007/978-3-319-67425-4_12
6. Villamizar M., Garcés O., Ochoa L., Castro H., Salamanca L., Verano M. M., Casallas R., Gil S., Valencia C., Zambrano A., Lang M. (2016). Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. DOI: 10.1109/CCGrid.2016.37.
7. Simple microservices architecture on AWS - Implementing Microservices on AWS (amazon.com) [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/simple-microservices-architecture-on-aws.html
8. Abgaz Y., Mccarren A., Elger P., Solan D., Lapuz N., Bivol M., Jackson G., Yilmaz M., Buckley J., Clarke, P. (2023). Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. IEEE Transactions on Software Engineering. PP. 1-32. DOI: 10.1109/TSE.2023.3287297.
9. What is SOA (Service-Oriented Architecture)? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/service-oriented-architecture/
10. Lewis G., Smith D., Chapin N., Kontogiannis K. (2010). Proceedings of the Fourth International Workshop on a Research Agenda for Maintenance and Evolution of Service- Oriented Systems (MESOA 2010). 107.

REFERENCES

1. Lakhai V., Bachynskyy R. (2021). Investigation of Serverless Architecture. Advances in Cyber-Physical Systems. 6. 134-139. DOI: 10.23939/acps2021.02.134.
2. Kumari A., Sahoo B. (2022). Serverless Architecture for Healthcare Management Systems. DOI: 10.4018/978-1-6684-4580-8.ch011.
3. dApp authentication with Amazon Cognito and Web3 proxy with Amazon API Gateway [Electronic resource] – Resource access mode: https://aws.amazon.com/blogs/architecture/dapp-authentication-with-amazon-cognito-and-web3-proxy-with-amazon-api-gateway/
4. Molodets B., Hnatushenko V., Boldyriev D., Bulana T. Information System of Air Quality Assessment Using Data Interpolation from Ground Stations. MoMLeT+DS 2023: 5th International Workshop on Modern Machine Learning Technologies and Data Science, June 3, 2023, Lviv, Ukraine. P.233-245

5. Dragoni, N. et al. (2017). Microservices: Yesterday, Today, and Tomorrow. In: Mazzara, M., Meyer, B. (eds) Present and Ulterior Software Engineering. Springer, Cham. DOI: 10.1007/978-3-319-67425-4_12
6. Villamizar M., Garcés O., Ochoa L., Castro H., Salamanca L., Verano M. M., Casallas R., Gil S., Valencia C., Zambrano A., Lang M. (2016). Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures. DOI: 10.1109/CCGrid.2016.37.
7. Simple microservices architecture on AWS - Implementing Microservices on AWS (amazon.com) [Electronic resource] – Resource access mode: https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/simple-microservices-architecture-on-aws.html
8. Abgaz Y., Mccarren A., Elger P., Solan D., Lapuz N., Bivol M., Jackson G., Yilmaz M., Buckley J., Clarke, P. (2023). Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. IEEE Transactions on Software Engineering. PP. 1-32. DOI: 10.1109/TSE.2023.3287297.
9. What is SOA (Service-Oriented Architecture)? [Electronic resource] – Resource access mode: https://aws.amazon.com/what-is/service-oriented-architecture/
10. Lewis G., Smith D., Chapin N., Kontogiannis K. (2010). Proceedings of the Fourth International Workshop on a Research Agenda for Maintenance and Evolution of Service- Oriented Systems (MESOA 2010). 107.

Received 19.04.2024.
Accepted 22.04.2024.

Analysis of existing architectures for the development of an Information System of Air Quality Assessment

Choosing the appropriate architecture is one of the key aspects, when designing any information system or software product. The system architecture determines how components are organized, how they interact, and the overall structure of the system.

The work is devoted to software architectures: serverless architecture, monolithic architecture, microservice architecture, and service-oriented architecture. All of them were compared with each other.

The following requirements were defined for the air quality assessment information system:

- *The architecture should provide modularity, flexibility and faster deployment and development.*
- *Providing integration and communication between different components of the system through services, which facilitates easy expansion.*

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

- *The possibility of deploying the system in cloud services to ensure scalability, availability and data security.*
- *Dividing the system into logical layers (e.g., data access, business logic, and presentation layers) to ensure ease of development and support.*

A monolithic architecture can be easier to deploy and scale than distributed systems. Serverless architecture provide huge cost savings for application developers, allowing them to pay only for the instances that run their serverless applications. Microservices allow for flexible scaling of the system by adding or removing individual services depending on the needs. Service-oriented architecture promotes high modularity and convenient interaction between system components. However the most suitable solution was the service-oriented architecture. That is because service-oriented architecture helps to develop common standards for exchanging data between services, which allows for greater data consistency and management across different parts of the system. Service-oriented architecture can have a lower overall infrastructure complexity (compared to microservice architecture) because the number of services can be lower Managing a smaller number of large services can be easier than managing a large number of small microservices that require more attention to detail.

To conclude, the usage of service-oriented architectures will create a dynamic and efficient system that can adapt to changing conditions and user requirements.

Молодець Богдан Володимирович – аспірант, Дніпровський національний університет імені Олеся Гончара.
Булана Тетяна Михайлівна – доцент, кандидат технічних наук, Національний Технічний Університет «Дніпровська політехніка».

Molodets Bohdan – Postgraduate Student, Oles Honchar Dnipro National University.
Bulana Tetiana – Associate Professor, Candidate of Technical Sciences (Ph.D.), Dnipro University of Technology.