УДК 004.02 : 004.42
DOI https://doi.org/10.26661/2786-6254-2024-1-09

МІКРОСЕРВІСНА АРХІТЕКТУРА СИСТЕМ СКІНЧЕННО-ЕЛЕМЕНТНОГО АНАЛІЗУ

Кривий Я. В.
аспірант кафедри програмної інженерії
Запорізький національний університет
вул. Жуковського, 66, Запоріжжя, Україна
orcid.org/0009-0006-8745-580X
kryvyi_yav@znu.edu.ua

Лісняк А. О.
доцент кафедри програмної інженерії
Запорізький національний університет
вул. Жуковського, 66, Запоріжжя, Україна
orcid.org/0000-0001-9669-7858
a.lisnyak@znu.edu.ua

Ключові слова:
мікросервісна архітектура, скінченно-елементний аналіз, FEA-системи, монолітна архітектура, масштабованість, відмовостійкість, модифікація, розгортання, архітектурні патерни, API Gateway, Aggregator, Database per Service, Event-Driven Architecture, Publisher/ Subscriber, Backend for Frontend, NoSQL, Angular, Node.js, Python, WebGL2.

У світі швидкого технологічного розвитку ефективність і гнучкість архітектур програмної інженерії відіграють ключову роль у створенні масштабованих і відмовостійких систем. Це набуває критичного значення для систем скінченно-елементного аналізу (FEA-систем), які використовуються для моделювання складних фізичних процесів в інженерії та часто повинні обробляти великі обсяги даних. Більшість сучасних FEA-систем використовують монолітну архітектуру – традиційну модель із єдиною кодовою базою для виконання різних функцій. Такий підхід має переваги, такі як єдине середовище розробки та легше налагодження взаємодії компонентів, і суттєві недоліки: складність масштабування, низьку відмовостійкість, погане балансування навантаження, зростання часу відповіді при збільшенні обсягів даних і складність впровадження нових функцій/технологій.

Одним із можливих рішень є концепція мікросервісної архітектури, яка передбачає розбиття програмного забезпечення на невеликі незалежні компоненти (сервіси). Кожен сервіс виконує одну функцію і взаємодіє з іншими через чітко визначені інтерфейси. Оскільки вони працюють незалежно, їх можна оновлювати, змінювати, розгортати або масштабувати окремо. Це надає низку переваг: швидке розгортання, незалежність сервісів, гнучке окреме масштабування, стійкість до збоїв, технологічну гнучкість, кращу організацію та простоту тестування, переваги у хмарних середовищах. У статті порівнюються монолітні (Elmer FEM, FreeFEM), мікросервісні (SimScale) і хмарно-монолітні (ANSYS Cloud) FEA-системи за критеріями архітектури, масштабованості, відмовостійкості, розгортання та модифікації. Обґрунтовується перевага мікросервісного підходу та пропонується архітектура FEA-системи на основі патернів API Gateway, Aggregator, Database per Service, Event-Driven, Publisher/ Subscriber, Backend for Frontend.

MICROSERVICE ARCHITECTURE OF FINITE ELEMENT ANALYSIS SYSTEMS

Kryvyi Y. V.
Postgraduate Student at the Department of Software Engineering
Zaporizhzhia National University
Zhukovskoho str., 66, Zaporizhzhia, Ukraine
orcid.org/0009-0006-8745-580X
kryvyi_yav@znu.edu.ua

Lisnyak A. O.
Associate Professor at the Department of Software Engineering
Zaporizhzhia National University
Zhukovskoho str., 66, Zaporizhzhia, Ukraine
orcid.org/0000-0001-9669-7858
a.lisnyak@znu.edu.ua

Key words: *microservice architecture, finite element analysis, FEA-systems, monolithic architecture, scalability, fault tolerance, modification, deployment, architectural patterns, API Gateway, Aggregator, Database per Service, Event-Driven Architecture, Publisher/ Subscriber, Backend for Frontend, NoSQL, Angular, Node.js, Python, WebGL2.*

In a world of rapid technological development, the efficiency and flexibility of software engineering architectures play a key role in creating scalable and fault-tolerant systems. This becomes critical for finite element analysis systems (FEA systems), which are used to model complex physical processes in engineering and often need to process large amounts of data. Most modern FEA systems use a monolithic architecture – a traditional model with a single code base for performing various functions. This approach has advantages, such as a single development environment and easier interaction between components, but also significant disadvantages: difficulty scaling, low fault tolerance, poor load balancing, increased response time with increasing data volumes, and difficulty implementing new features/technologies.
One of the possible solutions is the concept of microservice architecture, which involves breaking software into small independent components (services). Each service performs one function and interacts with others through clearly defined interfaces. Since they work independently, they can be updated, modified, deployed, or scaled separately. This provides a number of advantages: fast deployment, service independence, flexible individual scaling, fault tolerance, technological flexibility, better organization and ease of testing, and advantages in cloud environments. The article compares monolithic (Elmer FEM, FreeFEM), microservice (SimScale), and cloud-monolithic (ANSYS Cloud) FEA systems by the criteria of architecture, scalability, fault tolerance, deployment, and modification. The advantage of the microservice approach is substantiated and the architecture of the FEA system based on the API patterns Gateway, Aggregator, Database per Service, Event-Driven, Publisher/ Subscriber, Backend for Frontend is proposed.

Вступ. В епоху стрімкого технологічного прогресу системи скінченно-елементного аналізу відіграють вагому роль в інженерії та промисловості. Вони застосовуються для моделювання й аналізу поведінки конструкцій, механізмів, процесів і багатьох інших систем під впливом різноманітних чинників. FEA-системи дозволяють проводити складні розрахунки, візуалізувати результати та приймати оптимальні рішення на етапах проектування, зменшуючи витрати та підвищуючи ефективність виробництва.

Проте класична монолітна архітектура, яку використовують більшість сучасних FEA-систем, має низку недоліків, серед котрих складність масштабування, низька відмовостійкість, погане балансування навантаження та зростання часу відповіді при збільшенні обсягів даних. Монолітна архітектура передбачає, що програмне забезпечення розробляється як єдиний нерозділь-ний кодовий блок, де всі компоненти тісно пов'язані між собою.

Таким чином, актуальним стає пошук альтернативної архітектури, яка б не мала вищезазна-

чених недоліків і відповідала сучасним вимогам до масштабованості, гнучкості та продуктивності програмних систем. Одним із можливих рішень є концепція мікросервісної архітектури, котра передбачає розбиття програмного забезпечення на невеликі незалежні компоненти (сервіси), здатні взаємодіяти між собою й оновлюватися окремо.

Аналіз попередніх досліджень і публікацій. Питання переходу від монолітної до мікросервісної архітектури є надзвичайно актуальним у сучасній програмній інженерії. Численні дослідження та публікації розглядають різноманітні аспекти мікросервісного підходу та його переваги порівняно з монолітною архітектурою.

У праці [6] розглядається перехід від монолітної до мікросервісної архітектури у середовищі хмарних обчислень, а також пропонується комплексний підхід до підвищення масштабованості, гнучкості та надійності обчислювальних сервісів.

Перехід від монолітної до мікросервісної архітектури розглядається у низці статей [19; 20; 23]. Автори пропонують покроковий і комплексний підхід переходу від монолітної до мікросервісної архітектури, використовуючи різні архітектурні патерни, та наводять вплив кожного з них на масштабованість, стійкість і швидкість роботи системи.

Низку статей присвячено безпосередньо процесу переходу на мікросервісну архітектуру. Наприклад, AWS [14; 15] пропонує покроковий план міграції з монолітного додатку на мікросервісну архітектуру із залученням власного інструментарію. Аналогічний підхід викладено у документації Microsoft Azure [17] і Google Cloud [16].

Дослідження [21; 22; 24] аналізують застосування архітектурних патернів, таких як API Gateway, Database per Service, Event-Driven Architecture, Publisher/Subscriber і Backend for Frontend, для забезпечення гнучкості, масштабованості та модульності мікросервісних систем.

Що стосується систем скінченно-елементного аналізу, то принципи їх роботи розглядаються у статті [5] і фундаментальних підручниках [8; 9]. Проте питання архітектури сучасних FEA-систем висвітлюється переважно у технічній документації комерційних продуктів [1–3; 7; 18], а також у дослідженні [6], присвяченому застосуванню мікросервісної архітектури для FEA-систем у хмарному середовищі.

Таким чином, аналіз джерел показує, що переваги та підходи до впровадження мікросервісної архітектури є добре вивченими загалом. Водночас питання розробки й оптимізації архітектури саме для систем скінченно-елементного аналізу ще потребує подальших досліджень із урахуванням специфіки та високих вимог до продуктивності та масштабованості таких систем.

Метою роботи є огляд і порівняння монолітної та мікросервісної архітектур у контексті систем скінченно-елементного аналізу, аналіз їх переваг, недоліків і можливостей застосування для вдосконалення FEA-систем.

Викладення основного матеріалу. Архітектурний підхід на основі мікросервісів передбачає, що програмне забезпечення складається з невеликих незалежних компонентів (сервісів). Кожен сервіс виконує одну функцію та взаємодіє з іншими сервісами через чітко визначені інтерфейси. Оскільки вони працюють незалежно, то можна оновлювати, змінювати, розгортати або масштабувати кожну службу окремо в міру необхідності [13; 19; 20]. Щоб краще зрозуміти переваги мікросервісної архітектури над монолітною, необхідно для початку більш детально розглянути останню.

Монолітні системи розробляються як єдиний нероздільний кодовий блок. Будь-які зміни, навіть мінімальні, вимагають повної перебудови та повторного розгортання всього застосунку. Із часом підтримувати чистоту модульної структури стає дедалі складніше, а зміни в одному модулі часто впливають на інші модулі. Масштабування монолітних програм може бути складним, особливо коли різні модулі мають різні потреби до ресурсів. Наприклад, один модуль може виконувати обробку зображень, що інтенсивно використовує процесор, тоді як інший може потребувати багато оперативної пам’яті. Оскільки всі модулі розгортаються разом, часто доводиться йти на компроміс у виборі апаратного забезпечення. Це означає, що для масштабування одного модуля необхідно масштабувати увесь додаток [13; 19; 20].

Типові структури монолітних систем наведено на рис. 1.

Монолітна архітектура включає в себе все, починаючи від інтерфейсу користувача та закінчуючи викликами до бази даних, в одній кодовій базі [13; 19; 20].

На відміну від монолітних систем, мікросервісна архітектура (рис. 2) передбачає, що кожен мікросервіс розгортається окремо. Це означає, що зміни в одному мікросервісі можуть бути впроваджені без впливу на роботу інших, а це дозволяє їм продовжувати нормальну роботу [13; 19; 20].

Сучасні проекти дедалі частіше створюються як набір окремих сервісів, кожен із яких можна окремо розгортати та масштабувати, що має низку значних переваг.

Мікросервіси забезпечують швидке розгортання та розвиток, оскільки запускаються швидко, збільшуючи продуктивність розробників і прискорюючи час виходу нових функцій на ринок. Вони дозволяють незалежне розгортання, коли зміни в одному сервісі можуть бути впроваджені без перебоїв для інших, забезпечуючи непе-

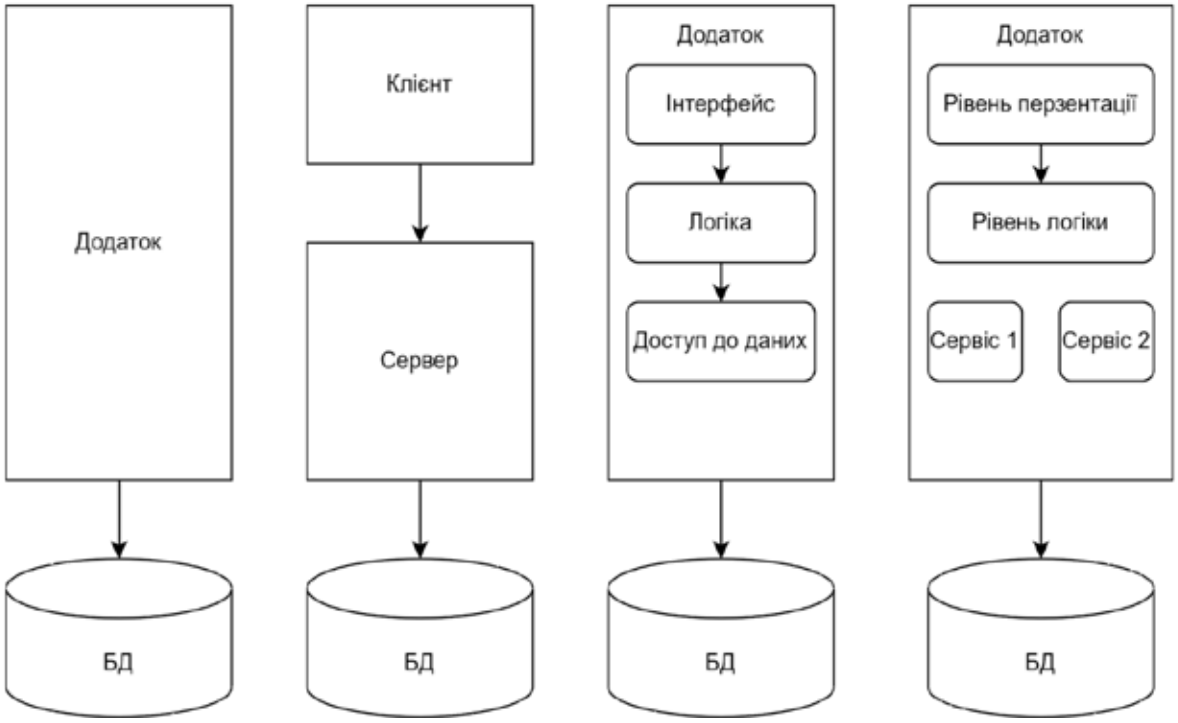

Рис. 1. Типові структури монолітних систем

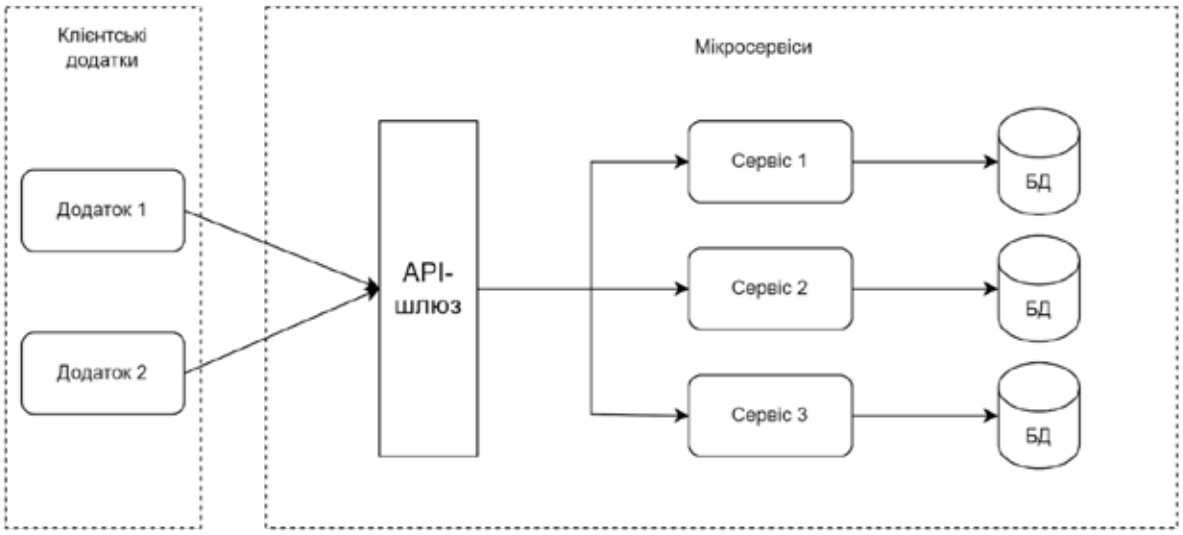

Рис. 2. Приклад мікросервісної архітектури

рервну роботу системи. Ще однією перевагою є гнучке масштабування, адже кожен мікросервіс може масштабуватися окремо відповідно до його потреб і використання ресурсів. Мікросервісна архітектура забезпечує стійкість до збоїв, оскільки проблеми в одному сервісі зазвичай не впливають на загальну роботу системи, роблячи всю архітектуру більш надійною.

Окрім цього, мікросервіси пропонують технологічну гнучкість, адже при їх додаванні або оновленні можна вибирати найоптимальніші технології для конкретних задач без обмежень через наявні вибори. Вони також забезпечують кращу організацію та легкість тестування, оскільки кожен мікросервіс виконує специфічну функцію, що робить його простішим для розуміння, підтримки та перевірки. Для мікросервісів розроблено багато хмарних сервісів із розвиненою інфраструктурою, а окремі сервіси можна легко переконфігурувати або інтегрувати у різні

додатки, наприклад, для обслуговування веб-клієнтів та API, демонструючи гнучке переналаштування.

Крім того, мікросервісна архітектура має переваги і в експлуатації. Вона дозволяє організаціям швидше адаптуватися до змін і впроваджувати нові технологічні рішення, забезпечуючи швидкість впровадження інновацій, на відміну від монолітних систем, де інтеграція нових технологій може бути ускладнена через жорстку залежність компонентів. Також у мікросервісних архітектурах збої одного сервісу не призводять до виходу з ладу всієї системи, знижуючи ризики в експлуатації порівняно з монолітними системами, де помилка у коді може вивести з ладу весь додаток.

Також мікросервіси дозволяють розробникам працювати над окремими частинами проекту незалежно, зменшуючи взаємозалежності та скорочуючи час розробки, на відміну від монолітів, де зростаюча складність коду може сповільнити цей процес, що скорочує час виходу продукту на ринок. Мікросервіси також ефективніші у плані довгострокових витрат, оскільки дозволяють масштабувати окремі сервіси, не впливаючи на решту системи, знижуючи потреби у ресурсах і витрати на інфраструктуру порівняно з монолітами, де масштабування вимагає значних інвестицій в усю інфраструктуру, що знижує загальну вартість володіння.

Отже, цей аналіз підкреслює, що мікросервісна архітектура може надати значні переваги в умовах швидких змін ринку та високих вимог до адаптивності та масштабування IT-інфраструктур, тоді як монолітні архітектури можуть зіткнутися з обмеженнями, що ускладнюють розвиток та інновації.

Порівнявши монолітну та мікросервісну архітектуру, порівняємо сучасні FEA-системи. Для порівняння оберемо мікросервісні (SimScale) [7], монолітні (Elmer FEM, FreeFEM) [2; 3] та монолітно-хмарні (ANSYS Cloud) [1; 18] системи.

Для порівняння виділимо такі критерії: архітектуру, масштабованість, відмовостійкість, розгортання та модифікацію. Результат порівняння наведемо у таблиці 1.

Проаналізувавши ці порівняння, можна дійти висновку про перевагу мікросервісної архітектури над монолітною для FEA-систем. Хоча відкритий код деяких монолітних систем і дозволяє вносити зміни, виникають проблеми із взаємодією компонентів, масштабуванням і налагодженням. Розгортання монолітної системи у хмарі може стати

Таблиця 1

Порівняння FEA-систем

Критерії / Системи	ANSYS Cloud	SimScale	Elmer FEM	FreeFEM
Архітектура	Монолітна, хмарна. Усі компоненти інтегровані у єдиний додаток, що забезпечує ефективність обчислень у хмарі.	Мікросервісна, хмарна. Модульна структура забезпечує гнучкість у розробці та підтримці окремих компонентів.	Монолітна, відкритий код. Хоча відкритий код дозволяє модифікацію, структура переважно монолітна.	Монолітна, відкритий код. Централізована структура, проте можлива адаптація завдяки відкритому коду.
Масштабованість	Хмарна інфраструктура дозволяє легко збільшувати обчислювальні потужності.	Легке додавання ресурсів завдяки мікросервісній структурі, кожен сервіс масштабується незалежно.	Можливість масштабування обмежена локальними ресурсами або налаштуваннями хмарної інфраструктури.	Переважно залежить від локальних ресурсів, із обмеженими можливостями для хмарного масштабування.
Відмовостійкість	Хмарні рішення мають вбудовані механізми відновлення, що забезпечують стійкість системи.	Незалежність мікросервісів знижує ризики збою всієї системи, кожен сервіс може відновлюватися окремо.	Відновлення залежить від локальної налаштованості та зовнішніх інструментів.	Більша схильність до збоїв через монолітну структуру, але можливе локальне відновлення.
Розгортання	Оптимізовано для простоти використання та швидкого розгортання у хмарній інфраструктурі.	Мікросервіси спрощують оновлення та розгортання окремих частин без впливу на інші.	Розгортання залежить від користувацьких налаштувань та обраної інфраструктури.	Вимагає більше часу та ресурсів для налаштування, не так ефективно, як хмарні рішення.
Модифікація	Залежить від можливостей та обмежень хмарної платформи.	Можливість легко змінювати або оновлювати окремі мікросервіси	Відкритий код дозволяє користувачам модифікувати та розширювати функціонал.	Відкритий код надає повну свободу у модифікації програми під специфічні потреби.

проміжним рішенням для підвищення гнучкості та полегшення переходу на мікросервісну архітектуру.

Більшість FEA-систем ще не перейшли на мікросервіси через простоту роботи з єдиною кодовою базою та кращу взаємодію модулів у монолітній архітектурі.

Типова монолітна FEA-система (рис. 3) складається із [5; 8; 9]:

– клієнтського інтерфейсу (UI) – веб- або десктоп-додатку для взаємодії користувача;

– серверної частини (Business Logic), що включає препроцесор для автоматизації підготовки вихідних даних, процесор для безпосереднього виконання скінченно-елементних розрахунків, і постпроцесор для аналізу отриманих числових результатів;

– системи управління базами даних (RDBMS) як сховища даних.

Для переходу на мікросервісну архітектуру необхідно розбити компоненти FEA-системи на окремі сервіси. Процес переходу з монолітної на мікросервісну архітектуру детально описано такими сервісами, як AWS, Google Cloud тощо.

При розбитті також необхідно враховувати архітектурні патерни, які дозволяють спроектувати архітектуру, що відповідає вимогам до мікросервісної архітектури.

Для проектування мікросервісної архітектури FEA-системи були обрані такі патерни:

– API Gateway – виступає єдиною точкою входу для всіх запитів від клієнтського застосунку та перенаправляє їх до відповідних мікросервісів [21; 22];

– Aggregator – може збирати дані від декількох сервісів і об’єднувати їх для надсилання єдиної відповіді клієнту [24];

– Database per Service – кожен мікросервіс використовує власну NoSQL базу даних, що забезпечує незалежність і стійкість до помилок, а також гнучкість у виборі технології бази даних [20];

– Event-Driven Architecture – виступає як центральний компонент для асинхронного обміну повідомленнями та подіями між мікросервісами, що підтримує слабку зв’язність і реактивність системи [20; 21; 24];

– Publisher/Subscriber – компоненти системи можуть публікувати події (наприклад, результати виконання завдань) у брокері повідомлень, звідки їх поширюватимуть усім зацікавленим підписникам [20; 21; 24];

– Backend for Frontend (BFF) – Aggregator виступає у ролі спеціалізованого бекенда для конкретного фронтенда (Angular UI), це дає змогу створити оптимальний інтерфейс для потреб користувача [20].

На основі обраних патернів наведемо мікросервісну архітектуру FEA-системи (рис. 4):

– UI (Angular) – фронтенд, реалізований із використанням Angular, призначений для взаємодії з користувачем;

– API Gateway – централізований вузол, який керує вхідним трафіком і направляє його до відповідних мікросервісів;

– Aggregator (Node.js) – використовується для агрегації даних від різних сервісів і відправки їх на UI;

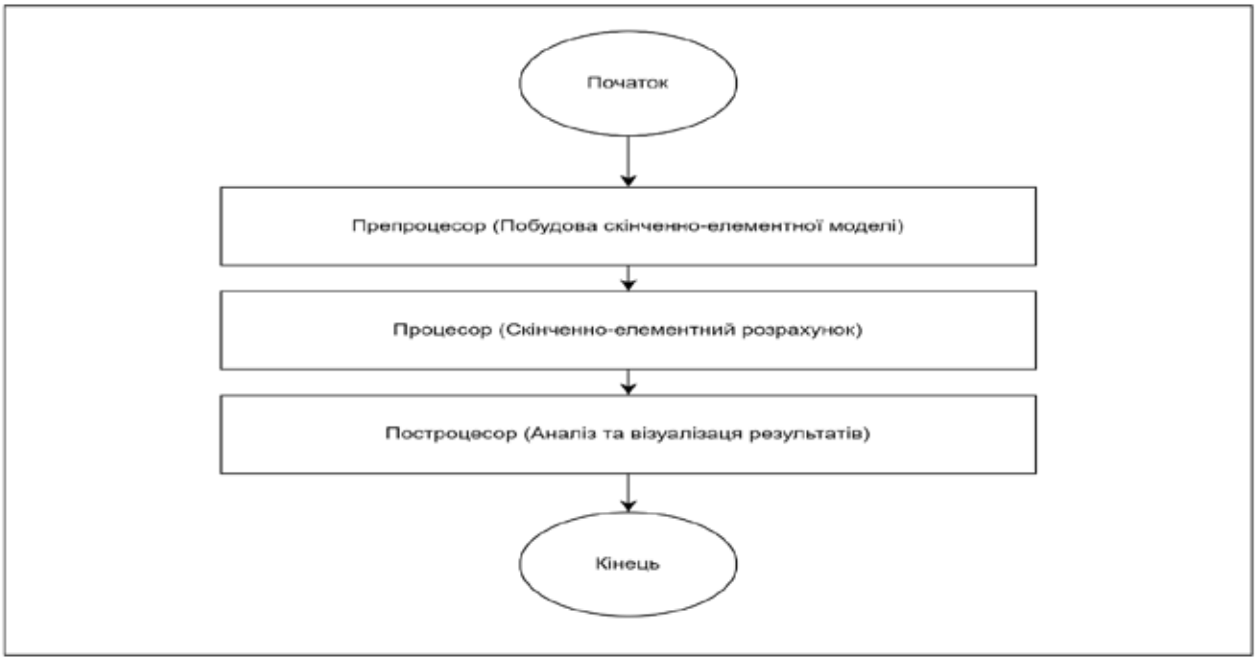

Рис. 3. Структура FEA-системи

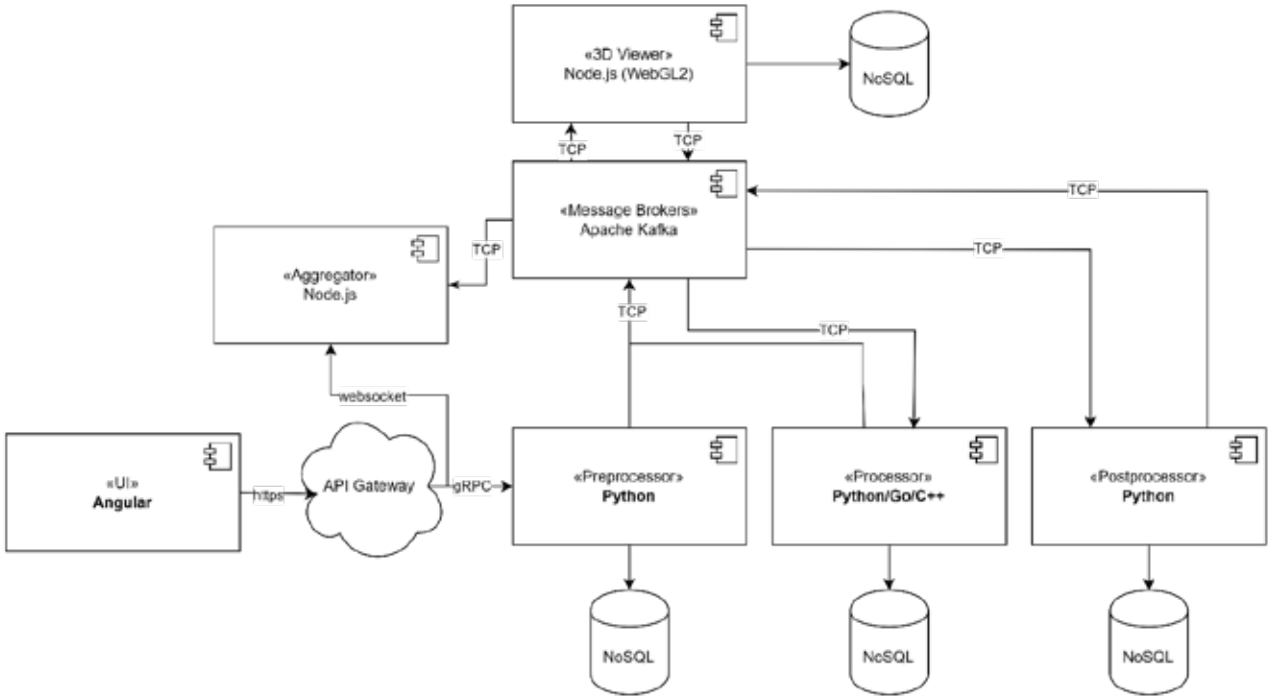

Рис. 4. Мікросервісна архітектура FEA-системи

– Message Brokers (Apache Kafka) – призначений для асинхронного обміну повідомленнями та подіями між мікросервісами;

– Preprocessor (Python) – обробляє вхідні дані перед їх передачею до процесора;

– Processor (Python/Go/C++) – основний сервіс, що виконує розрахунки FEA;

– Postprocessor (Python) – обробляє й аналізує результати розрахунків;

– 3D Viewer (WebGL2) – обробляє їх для створення візуалізацій, які можуть бути представлені як 3D-сцени або графіки;

– NoSQL – бази даних NoSQL використовуються для зберігання даних, що зумовлено їхньою гнучкістю та масштабованістю.

Висновки. У статті було проаналізовано монолітну та мікросервісну архітектури у контексті систем скінченно-елементного аналізу.

Огляд показав, що класична монолітна архітектура, яка широко використовується у більшості сучасних FEA-систем, має низку недоліків, таких як складність масштабування, низька відмовостійкість, погане балансування навантаження та зростання часу відповіді при збільшенні обсягів даних. Ці недоліки можуть стати перешкодою для ефективного функціонування та подальшого розвитку FEA-систем в умовах зростаючих вимог до продуктивності, гнучкості та масштабованості.

Мікросервісна архітектура, котра передбачає розбиття програмного забезпечення на невеликі незалежні компоненти, здатна вирішити зазначені проблеми монолітного підходу. Як показало порівняння з FEA-системами, перехід на мікросервісну архітектуру може забезпечити такі переваги, як швидке розгортання нових функцій, незалежне масштабування окремих сервісів, підвищену відмовостійкість, гнучкість у виборі технологій і легкість модифікації.

З огляду на потенційні переваги мікросервісного підходу у статті була запропонована мікросервісна архітектура для FEA-системи, спроектована з використанням таких архітектурних патернів, як API Gateway, Aggregator, Database per Service, Event-Driven Architecture, Publisher/Subscriber та Backend for Frontend. Ця архітектура складається із клієнтського інтерфейсу, мікросервісів передобробки, обчислень, постобробки та візуалізації, а також компонентів для обміну повідомленнями та баз даних NoSQL.

Таким чином, можна зробити висновок, що впровадження мікросервісної архітектури у системах скінченно-елементного аналізу є перспективним напрямом для подолання обмежень монолітного підходу та забезпечення необхідної масштабованості, гнучкості та продуктивності, проте слід враховувати, що перехід на мікросервіси вимагає ретельного проектування, дотримання відповідних архітектурних принципів і може бути пов’язаний із певними труднощами та витратами на реалізацію. Тому доцільність такого переходу слід оцінювати з урахуванням конкретних вимог та особливостей FEA-системи.

ЛІТЕРАТУРА

1. ANSYS Cloud. URL: https://www.ansys.com/products/cloud (дата звернення: 24.03.2024).
2. Elmer FEM – відкрите програмне забезпечення для моделювання. URL: https://www.csc.fi/web/elmer (дата звернення: 24.03.2024).
3. FreeFEM – відкрите ПЗ для моделювання диференціальних рівнянь в частинних похідних. URL: https://freefem.org/ (дата звернення: 24.03.2024).
4. Frey P. J.,George P.-L. Mesh Generation Application to Finite Elements. London: ISTE Publishing Company, 2019. 817 p.
5. IEEE. Innovation at work. How the Finite Element Method (FEM) and Finite Element Analysis (FEA) Work Together. URL: https://innovationatwork.ieee.org/how-the-finite-element-method-fem-and-finiteelement-analysis-fea-work-together/ (дата звернення: 09.03.2024)
6. Muhtaroglu N. Finite element analysis in a cloud computing environment : дисертація / Muhtaroglu N. Ozyegin University, 2019. 112 p.
7. SimScale – хмарна платформа інженерного моделювання. URL: https://www.simscale.com/ (дата звернення: 24.03.2024).
8. Top Finite Element Analysis (FEA) Software List, Reviews, Comparison & Price | TEC. URL: https://www3.technologyevaluation.com/sd/category/finite-elementanalysis-fea (дата звернення: 09.03.2024).
9. Zienkiewicz O.C., Taylor R.L., Zhu J.Z. The Finite Element Method: Its Basis and Fundamentals. Sixth edition. Butterworth-Heinemann, 2016. 753 p.
10. Мікросервісна архітектура для початківців. Ч. І. URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/ (дата звернення: 24.04.2024).
11. Мікросервісна архітектура для початківців. Ч. ІІ. URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-two/ (дата звернення: 24.04.2024).
12. Newman S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media, Inc, 2019. 272 p.
13. What's the difference between monolithic and microservices architecture? URL: https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/?nc1=h_ls (дата звернення: 23.04.2024).
14. Break a Monolithic Application into Microservices with AWS Copilot, Amazon ECS, Docker, and AWS Fargate. URL: https://aws.amazon.com/tutorials/break-monolith-app-microservices-ecs-docker-ec2/ (дата звернення: 23.04.2024).
15. AWS Documentation. URL: https://docs.aws.amazon.com/?nc2=h_ql_doc_do (дата звернення: 23.04.2024).
16. Google Cloud Documentation. URL: https://cloud.google.com/docs (дата звернення: 23.04.2024).
17. Azure documentation. URL: https://learn.microsoft.com/en-gb/azure (дата звернення: 23.04.2024).
18. Ansys Cloud Architecture and Security Overview. URL: https://www.ansys.com/resource-center/white-paper/cloud-security (дата звернення: 24.03.2024).
19. Mazzara M., Bucchiarone A., Dragoni N., Rivera V. Size Matters: Microservices Research and Applications. *Springer Cham.* 2020. pp. 29–42. doi: https://doi.org/10.1007/978-3-030-31646-4_2.
20. Velepucha V., Flores P. A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. *IEEE Access.* 2023. Vol. 11, P. 88339 – 88358. DOI: 10.1109/ACCESS.2023.3305687.
21. Zuo X., Su Y., Wang Q., Xie Y. An API gateway design strategy optimized for persistence and coupling. *Advances in Engineering Software.* 2020. Vol. 148, № 102878. doi: https://doi.org/10.1016/j.advengsoft.2020.102878.
22. Zhao J.T., Jing S.Y., Jiang L.Z. Management of API Gateway Based on Micro-service Architecture. *Journal of Physics: Conference Series.* 2018. Vol. 1087. Issue 3. № 1087. DOI: 10.1088/1742-6596/1087/3/032032.
23. Kalske M., Mäkitalo N., Mikkonen T. Challenges When Moving from Monolith to Microservice Architecture. *Springer Cham.* 2018. Vol 10544. P. 32–47. doi: https://doi.org/10.1007/978-3-319-74433-9_3.
24. Taibi D., Lenarduzzi V., Pahl C. Architectural Patterns for Microservices: A Systematic Mapping Study. *8th International Conference on Cloud Computing and Services Science.* 2018. P. 221–232. DOI: 10.5220/0006798302210232.

REFERENCES

1. ANSYS Cloud. Retrieved March 24, 2024, from https://www.ansys.com/products/cloud
2. Elmer FEM – open software for modeling. Retrieved March 24, 2024, from https://www.csc.fi/web/elmer
3. FreeFEM – open software for modeling partial differential equations. Retrieved March 24, 2024, from https://freefem.org/

4. Frey, P.J., & George, P.-L. (2019). Mesh generation application to finite elements. London: ISTE Publishing Company.
5. IEEE. (n.d.). How the finite element method (FEM) and finite element analysis (FEA) work together. Retrieved March 9, 2024, from https://innovationatwork.ieee.org/how-the-finite-element-method-fem-and-finiteelement-analysis-fea-work-together/
6. Muhtaroglu, N. (2019). Finite element analysis in a cloud computing environment (Doctoral dissertation). Ozyegin University.
7. SimScale – cloud engineering modeling platform. Retrieved March 24, 2024, from https://www.simscale.com/
8. Top finite element analysis (FEA) software list, reviews, comparison & price | TEC. Retrieved March 9, 2024, from https://www3.technologyevaluation.com/sd/category/finite-elementanalysis-fea
9. Zienkiewicz, O.C., Taylor, R.L., & Zhu, J.Z. (2016). The finite element method: Its basis and fundamentals (6th ed.). Butterworth-Heinemann.
10. Microservice architecture for beginners. Part I. Retrieved April 24, 2024, from https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/
11. Microservice architecture for beginners. Part II. Retrieved April 24, 2024, from https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-two/
12. Newman, S. (2019). Monolith to microservices: Evolutionary patterns to transform your monolith. O'Reilly Media, Inc.
13. What's the difference between monolithic and microservices architecture? Retrieved April 23, 2024, from https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/?nc1=h_ls
14. Break a monolithic application into microservices with AWS Copilot, Amazon ECS, Docker, and AWS Fargate. Retrieved April 23, 2024, from https://aws.amazon.com/tutorials/break-monolith-app-microservices-ecs-docker-ec2/
15. AWS documentation. Retrieved April 23, 2024, from https://docs.aws.amazon.com/?nc2=h_ql_doc_do
16. Google Cloud documentation. Retrieved April 23, 2024, from https://cloud.google.com/docs
17. Azure documentation. Retrieved April 23, 2024, from https://learn.microsoft.com/en-gb/azure
18. Ansys Cloud architecture and security overview. Retrieved March 24, 2024, from https://www.ansys.com/resource-center/white-paper/cloud-security
19. Mazzara, M., Bucchiarone, A., Dragoni, N., & Rivera, V. (2020). Size matters: Microservices research and applications. In Springer Cham. https://doi.org/10.1007/978-3-030-31646-4_2
20. Velepucha, V., & Flores, P. (2023). A survey on microservices architecture: Principles, patterns and migration challenges. IEEE Access, 11, 88339-88358. https://doi.org/10.1109/ACCESS.2023.3305687
21. Zuo, X., Su, Y., Wang, Q., & Xie, Y. (2020). An API gateway design strategy optimized for persistence and coupling. Advances in Engineering Software, 148, Article 102878. https://doi.org/10.1016/j.advengsoft.2020.102878
22. Zhao, J. T., Jing, S. Y., & Jiang, L. Z. (2018). Management of API gateway based on micro-service architecture. Journal of Physics: Conference Series, 1087(3), Article 032032. https://doi.org/10.1088/1742-6596/1087/3/032032
23. Kalske, M., Mäkitalo, N, Mikkonen, T. (2018). Challenges when moving from monolith to microservice architecture. In Springer Cham (Vol. 10544, pp. 32-47). https://doi.org/10.1007/978-3-319-74433-9_3
24. Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). Architectural patterns for microservices: A systematic mapping study. In Proceedings of the 8th International Conference on Cloud Computing and Services Science (pp. 221–232). https://doi.org/10.5220/0006798302210232.