МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
Кафедра Програмної інженерії

# КВАЛІФІКАЦІЙНА РОБОТА

## Пояснювальна записка

_____другий (магістерський)_____
(рівень вищої освіти)

____Дослідження методів оцінювання ефективності розгортання програмного___
_____забезпечення для різних класів інформаційних систем_____
(тема)

Виконав:
студент _2_ курсу, групи ІПЗм-21-2_____
_____Вискребець Д.О._____
(прізвище, ініціали)
Спеціальність 121 – Інженерія програмного___
_____забезпечення_____
(код і повна назва спеціальності)
Тип програми _____Освітньо-наукова____
Керівник __проф. Ігор ШОСТАК____
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри, проф. _____ Зоя ДУДАР

Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____

Кафедра _____Програмної Інженерії_____

Рівень вищої освіти _____другий (магістерський)_____

Спеціальність _____121 – Інженерія програмного забезпечення_____

Тип програми _____освітньо-наукова програма_____

Освітня програма _____Інженерія програмного забезпечення_____

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____»_____ 2023 р.

**ЗАВДАННЯ**

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента _____Вискребця Дениса Олеговича_____

1. Тема роботи «Дослідження методів оцінювання ефективності розгортання програмного забезпечення для різних класів інформаційних систем»

затверджена наказом університету від «___» _____ 2023р. № _____

2. Термін подання студентом роботи до екзаменаційної комісії «__» _____ 2023 р.

3. Вихідні дані до роботи *методичні вказівки до оформлення роботи, мова програмування C#, фреймворк .NET 6, середовище розробки Visual Studio 2019.*

4. Перелік питань, що потрібно опрацювати в роботі _вступ, аналіз предметної області, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, математичне формування задачі._____

# КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної галузі | 10.11.2022 | виконано |
| 2 | Постановка задачі | 20.11.2022 | виконано |
| 3 | Здійснення огляду математичних моделей | 25.12.2022 | виконано |
| 4 | Аналіз існуючих моделей прогнозування | 30.01.2023 | виконано |
| 5 | Дослідження обраних алгоритмів | 01.03.2023 | виконано |
| 6 | Написання пояснювальної записки | 15.04.2023 | виконано |
| 7 | Підготовка презентації та доповіді | 25.04.2023 | виконано |
| 8 | Перевірка роботи на плагіат та нормоконтроль | 08.05.2023 | виконано |
| 9 | Захист кваліфікаційної роботи | 19.05.2023 | виконано |

Дата видачі завдання «29» березня 2023 р.

Студент _____ Вискребець Д.О.

(підпис)

Керівник роботи _____ _____проф. Шостак І.В. ___

(підпис)

# РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 69 с., 10 рис., 4 табл., 11 джер.

ЗАДАЧА ОПТИМІЗАЦІЇ, МОВА ПРОГРАМУВАННЯ C#, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ, ПЛАТФОРМА .NET, ХМАРНІ ОБЧИСЛЕННЯ, ХМАРНІ ТЕХНОЛОГІЇ.

Об'єктом дослідження є методів оцінювання ефективності розгортання програмного забезпечення.

Метою роботи є проведення дослідження оптимальності методи оцінювання ефективності розгортання програмного забезпечення.

В результаті роботи було досліджено методи оцінювання ефективності розгортання програмного забезпечення, проаналізовані аналоги, запропонована програмна система вирішення поставленої задачі разом із документацією та виконана підготовча робота для подальших досліджень.

CLOUD COMPUTING, CLOUD TECHNOLOGIES, C# PROGRAMMING LANGUAGE, .NET PLATFORM, OPTIMIZATION PROBLEM, OBJECT-ORIENTED PROGRAMMING.

The object of the study is methods of evaluating the effectiveness of software deployment.

The purpose of the work is to conduct research on the optimality of methods for evaluating the effectiveness of software deployment.

As a result of the work, methods of evaluating the effectiveness of software deployment were studied, analogs were analyzed, a software system for solving the given problem was proposed along with documentation, and preparatory work for further research was performed.

Умови публікації пояснювальної записки

Я, Вискребець Денис Олегович, студент групи ІПЗм-21-2 здобувач вищої освіти на другому (магістерському) рівні кафедри програмної інженерії заявляю: моя кваліфікаційна робота на тему «Дослідження методів оцінювання ефективності розгортання програмного забезпечення для різних класів інформаційних систем», що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

# ЗМІСТ

# ВСТУП

У сучасному світі інформаційні технології відіграють важливу роль у розвитку бізнесу та підтримці інфраструктури. Запровадження програмного забезпечення є необхідним елементом багатьох процесів, пов'язаних з розробкою, впровадженням та підтримкою інформаційних систем. Процес розгортання програмного забезпечення може бути складним завданням, оскільки він вимагає ретельного планування та оцінки ризиків, пов'язаних з різними аспектами системи.

Зазначена обставина визначає актуальність розробки ефективних методичних засобів, а на цій основі – програмних застосунків, які можуть допомогти оцінити ефективність розгортання програмного забезпечення в залежності від особливостей та вимог різних класів інформаційних систем. Дослідження в цій області може привести до розробки нових методів та інструментів для покращення ефективності процесу розгортання програмного забезпечення.

Дана робота присвячена дослідженню методів оцінювання ефективності розгортання програмного забезпечення для різних класів інформаційних систем. У роботі проаналізовані існуючі програмні рішення та запропоновано нові методи оцінки ефективності розгортання програмного забезпечення. Окрім того, розглянуті різні класи інформаційних систем та їх особливості, що можуть вплинути на ефективність процесу розгортання програмного забезпечення.

Результати цього дослідження будуть корисні розробникам, тестувальникам та проєктним менеджерам, архітекторам програмних систем та додатків, а створені рекомендації щодо використання методів оцінки ефективності розгортання програмного забезпечення для різних класів інформаційних систем можуть бути використані при аналізі, оцінці та виборі раціонального методу розгортання на етапі проєктування.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Аналіз предметної галузі

З кожним роком зростає кількість інформаційних систем, які мають велике значення для бізнесу, громадянського суспільства, держави та інших сфер життя людей. Важливість цих систем полягає в тому, що вони дозволяють швидко та ефективно обробляти великі обсяги даних, забезпечують доступ до інформації та сприяють розвитку інформаційних технологій загалом.

Однак, розгортання програмного забезпечення (ПЗ) для цих систем може бути дуже складним та довготривалим процесом. Це може відбитися на вартості та якості розгорнутої системи, а також на часі, необхідному для розгортання. У цьому контексті, оцінювання ефективності методів розгортання програмного забезпечення для різних класів інформаційних систем має велике значення.

Розробників та архітекторів програмного забезпечення, саме вартість програмного рішення цікавить найбільше. Адже цей чинник узгоджується зі стейкхолдерами проєкту, і саме останні приймають рішення про запуск проєкту чи скорочення ресурсів, які виділяються на нього.

Зараз існує безліч методів розгортання програмного забезпечення, які мають свої переваги та недоліки. Дослідження методів оцінювання ефективності розгортання програмного забезпечення дозволить визначити, які методи найбільш підходять для конкретної інформаційної системи з урахуванням її особливостей та вимог.

Також важливо враховувати тенденції, що складаються в сфері розгортання програмного забезпечення. Наприклад, зростає популярність хмарних технологій та контейнеризації, що може суттєво вплинути на ефективність розгортання програмного забезпечення. Дослідження методів оцінювання ефективності розгортання програмного забезпечення дозволить врахувати такі тенденції та забезпечити адаптивність цих методів до нових технологій та підходів при розгортанні програмного забезпечення.

У зв'язку з тим, що інформаційні системи стають все більш складними, ефективне розгортання програмного забезпечення таких систем є також складною задачею. Дослідження методів оцінювання ефективності розгортання програмного забезпечення дозволить знайти оптимальні методи для різних класів інформаційних систем та забезпечити їх ефективну роботу.

Предметна галузь дослідження методів оцінювання ефективності розгортання програмного забезпечення для різних класів інформаційних систем є досить широкою та комплексною. Вивчення цієї галузі передбачає аналіз багатьох аспектів, які можуть впливати на ефективність розгортання програмного забезпечення.

Одним з ключових аспектів є архітектура програмного забезпечення, яка може мати значний вплив на ефективність розгортання інформаційної системи. Для різних класів інформаційних систем можуть бути використані різні архітектурні підходи, такі як монолітні, мікросервісні, сервіс-орієнтовані, та інші. Кожен з цих підходів має свої переваги та недоліки, які можуть впливати на ефективність розгортання.

Інший важливий аспект – це технології, що використовуються для розгортання програмного забезпечення. Сьогодні на ринку існує велика кількість інструментів для автоматизації розгортання, такі як Ansible, Puppet, Chef, Docker та інші. Вибір оптимальної технології залежить від різних чинників, таких як розмір системи, складність інфраструктури, тип застосунку, потреби користувачів, тощо.

Також варто звернути увагу на процес розробки та тестування програмного забезпечення. Ефективне розгортання програмного забезпечення потребує попередньої розробки, тестування та валідації системи. Якість коду, тестування та налагодження системи мають велике значення для ефективного розгортання програмного забезпечення.

Окрім того, для різних класів інформаційних систем можуть бути різні вимоги щодо ефективності розгортання програмного забезпечення. Наприклад, системи, які обробляють великий потік даних, можуть вимагати швидкого та надійного розгортання, щоб забезпечити високу продуктивність та доступність. У

той же час, інші системи можуть мати більші вимоги до безпеки та стійкості, що може вимагати більш ретельної перевірки під час розгортання.

Нарешті, важливим аспектом є здатність до масштабування системи. Якщо інформаційна система буде зростати за своїми масштабами, розгортання програмного забезпечення повинно бути здатними масштабуватися відповідно. Наприклад, якщо під час розгортання програмного забезпечення використовуються рішення, які не можуть масштабуватися, це може призвести до зниження ефективності системи та збільшення часу простою.

Отже, аналіз предметної галузі показує, що дослідження методів оцінювання ефективності розгортання програмного забезпечення є дуже актуальною темою. Вона має велике значення для розробників та інженерів, які відповідають за розгортання програмного забезпечення для різних класів інформаційних систем.

## 1.2 Визначення проблем та актуалізація вирішень

Наукові дослідження в галузі оцінювання ефективності розгортання програмного забезпечення стали об'єктом багатьох наукових робіт. Так, розглянемо кілька важливих досліджень на цю тему.

В роботі [1] автори проводять дослідження ефективності розгортання програмного забезпечення в хмарному середовищі. Вони розглядають різні методи оцінювання ефективності, включаючи час розгортання, надійність та швидкість. Результати дослідження дозволяють визначити оптимальний метод розгортання для певного типу програмного забезпечення.

В наступній роботі [2] автори також досліджують ефективність розгортання програмного забезпечення в хмарному середовищі. Вони використовують різні методи оцінювання, включаючи час розгортання, надійність та витрати. Результати дослідження вказують на те, що розгортання програмного забезпечення в хмарному середовищі може бути ефективнішим, ніж традиційні методи.

В ще одній роботі [3] автори пропонують фреймворк для оцінювання ефективності розгортання програмного забезпечення. Фреймворк включає різні метрики, такі як час розгортання, витрати та надійність. Вони також використовують статистичні методи для оцінювання різних методів розгортання. Результати дослідження показують, що ефективність розгортання програмного забезпечення залежить від вибору способу доставки і розгортання програмного додатку.

Інші дослідження, що були проведені в цій галузі, зосереджувалися на побудові моделей оцінювання ефективності розгортання програмного забезпечення. Одним із прикладів є дослідження, опубліковане в журналі «Software Quality Journal», де було запропоновано модель оцінювання ефективності розгортання програмного забезпечення на основі аналізу факторів ризику та використання методів статистичної обробки даних. Крім того, в останні роки виникла потреба в розробці методів оцінювання ефективності розгортання програмного забезпечення в хмарних середовищах. Такі дослідження були опубліковані в таких журналах, як «IEEE Transactions on Cloud Computing» та «Journal of Cloud Computing».

Узагальнюючи, можна сказати, що дослідження з оцінювання ефективності розгортання програмного забезпечення виявляються актуальними і важливими в контексті розвитку інформаційних технологій та забезпечення якості програмного забезпечення.

Якщо проаналізувати проблеми, які були описані в згаданих роботах, можна навести наступний список:

1) недостатня точність оцінювання ефективності: деякі методи оцінювання можуть бути недостатньо точними, оскільки вони не враховують всіх факторів, які можуть вплинути на результативність розгортання програмного забезпечення;
2) недостатня здатність до прогнозування: існуючі методи оцінювання можуть бути не здатними до прогнозування ефективності при розгортанні

програмного забезпечення для нових класів інформаційних систем або нових технологій;

3) недостатня увага до аспектів безпеки: деякі методи оцінювання можуть не враховувати аспекти безпеки, які можуть бути важливими при розгортанні програмного забезпечення в деяких класах інформаційних систем;
4) обмеження з точки зору масштабування: деякі методи оцінювання можуть бути обмеженими з точки зору масштабування і можуть не бути застосовними для великих систем;
5) відсутність стандартів: на сьогоднішній день відсутні стандарти, які б дозволяли порівнювати різні методи оцінювання ефективності розгортання програмного забезпечення, що ускладнює вибір найбільш підходящого методу для конкретної задачі.

Ще одна проблема, на яку звертають увагу деякі дослідники, – це використання недостатньо репрезентативних метрик оцінювання ефективності розгортання ПЗ. Часто використовуються метрики, які не враховують специфіку конкретної інформаційної системи, її масштабу, типу та характеру даних, що обробляються. Такі метрики можуть давати неточні результати та не давати повної картини про ефективність розгортання ПЗ в конкретній системі.

Крім того, – ще однією важливою проблемою є складність моделювання реальних сценаріїв розгортання ПЗ. На даний момент існуючі методи моделювання є досить спрощеними та не враховують ряд важливих факторів, таких як різні конфігурації обладнання, типи мереж, різні типи даних та їх об'єм, які можуть впливати на ефективність розгортання ПЗ.

Також існує проблема підвищення складності розгортання ПЗ при збільшенні масштабу інформаційної системи. При збільшенні кількості вузлів, які беруть участь у розгортанні, складність координації процесу розгортання збільшується, що може призводити до затримок та помилок у розгортанні ПЗ. Ця проблема особливо гостро стоїть у корпоративних системах з сотнями або тисячами вузлів.

Якщо говорити загалом про існуючі рішення, що використовуються для оцінювання ефективності розгортання програмного забезпечення, має місце низка специфічних проблем.

Наприклад, деякі дослідження використовують традиційні методи оцінювання ефективності, такі як час виконання та розмір програми, які можуть не враховувати деякі аспекти розгортання програмного забезпечення, такі як його взаємодія з іншими системами, безпека та надійність.

Також існують дослідження, що пропонують використовувати інтелектуальні алгоритми машинного навчання для оцінювання ефективності розгортання програмного забезпечення. Однак, для застосування таких алгоритмів необхідні великі обсяги даних та ресурсів обчислювальної техніки, що може ускладнити їх використання у практичних застосуваннях.

Крім того, ряд дослідників зосереджується на використанні часу розгортання як метрики ефективності, що може бути недостатнім для повної оцінки ефективності розгортання програмного забезпечення. Наприклад, деякі інформаційні системи можуть вимагати більш високої безпеки, надійності або масштабованості, тому інші метрики ефективності, які враховують ці аспекти, можуть бути більш підходящими для оцінювання їх ефективності.

Отже, необхідно розробляти нові методи та інструменти оцінювання ефективності розгортання програмного забезпечення для різних класів інформаційних систем, які враховуватимуть специфіку програмного забезпечення, на якому воно розгортається, а також забезпечуватимуть моніторинг його ефективності в реальному часі. Такі методи дозволять покращити продуктивність та якість програмного забезпечення та зменшити час розгортання ПЗ.

## 1.3 Формування мети

Метою даної роботи є дослідження вибору оптимального методу розгортання, враховуючи технічні особливості середовища виконання та вимоги до

додатку, використовуючи фінансові витрати необхідні для роботи програмного рішення як основний показник та метрику ефективності. Увагу в роботі пропонується зосередити на хмарній платформі Amazon Web Services (AWS). В ході роботи передбачається розробити програмне рішення, яке буде аналізувати роботу програмної системи, зчитувати показники додатку та пропонувати зміни засобу розгортання в залежності від потенційних витрат. Програмне рішення пропонується побудувати на основі хмарної інфраструктури AWS для забезпечення ефективного моніторингу та аналізу середовища виконання AWS Lambda функцій, з використанням метрик, отриманих з AWS CloudWatch та AWS Lambda Insights. Конкретні питання, які досліджуватимуться, включають таке: як впливає обсяг запитів на функції Lambda на продуктивність та витрати ресурсів; як впливає розмір пакету функції та кількість залежностей на час виконання функції та витрати ресурсів; як впливає розмір пам'яті, відведеної для функції, на її продуктивність та витрати ресурсів; як впливає кількість одночасних викликів до функції на її продуктивність та витрати ресурсів; як можна оптимізувати використання середовища виконання, щоб зменшити витрати та покращити продуктивність функцій Lambda.

Таким чином, мета дослідницької роботи полягає в тому, щоб дослідити ці питання та знайти практичні рішення для оптимізації використання середовища виконання AWS Lambda, для зниження витрат та підвищення продуктивності.

У результаті дослідження ми отримаємо детальний огляд і порівняння різних методів оцінювання ефективності розгортання програмного забезпечення для різних класів інформаційних систем, що дозволить зробити обґрунтований вибір найбільш оптимального методу для кожної конкретної ситуації. Також в результаті дослідження будуть виділені можливості покращення процесу розгортання програмного забезпечення для різних класів інформаційних систем, що сприятиме зменшенню часу та затрат на розгортання ПЗ, підвищенню його якості та забезпеченню більш ефективного використання його функціональності.

# 2 МАТЕМАТИЧНЕ ПРЕДСТАВЛЕННЯ

## 2.1 Багатокритеріальна задача вибору методу розгортання

Для того, щоб описати задачу вибору методу розгортання на платформі AWS як багатокритеріальну задачу вибору, необхідно спочатку визначити критерії, які відображають вимоги та обмеження проекту. Основні критерії, які можуть впливати на вибір методу розгортання, можуть включати такі:

1) час розгортання: час, необхідний для повного розгортання системи;
2) вартість: загальна вартість проекту, включаючи витрати на обладнання, програмне забезпечення та інші витрати;
3) масштабованість: здатність системи збільшуватися або зменшуватися в залежності від потреб;
4) надійність: можливість запобігати відмовам системи та швидко відновлюватися у разі їх виникнення;
5) місце збереження даних: можливість зберігати дані віддалено або локально;
6) керованість: рівень контролю, який має розробник над інфраструктурою та сервісами, що використовуються.

Крім того, можна розглядати й інші критерії, такі як доступність, безпека, продуктивність, гнучкість, простота використання та інші.

Отже, задача вибору методу розгортання на платформі AWS відноситься до класу багатокритеріальних задач вибору, де кожен критерій має вагу, яка відображає його важливість для проекту. Завдання полягає в тому, щоб вибрати метод розгортання, який забезпечує найкраще поєднання цих критеріїв, що задовольняє вимоги проекту та обмеження.

Існують декілька способів розв’язаних подібних задач. В рамках цієї роботи ми розглянемо метод, який можна звести до одного критерія, враховуючи що найважливішим критерієм для нас є фінансові витрати на підтримку програмного рішення. Для цього ми будем використовувати метод головної компоненти.

Ідея методу головної компоненти полягає в тому, що узагальнений критерій (часом його називають критерій якості) зв'язується з одним із критеріїв, що вибраний в ролі основного (головного), а на основі інших критеріїв будують обмеження. В такому випадку отримують задачу однокритеріальної оптимізації для розв'язання якої використовують відповідні методи на основі виду та особливостей цільової функції обмеження.

Описати цей метод можна наступним чином. Припустимо, нам потрібно вирішити задачу багатокритеріальної оптимізації, яка має декілька критеріїв оптимальності. Цільову функцію такої задачі можна описати так:

$$F = \{\max f_1(\bar{X}), \max f_2(\bar{X}), \dots, \min f_{n-1}(\bar{X}), \min f_n(\bar{X})\} \qquad (1)$$

де $f_1(\bar{X})$ – перша цільова функція, яка опису першу характеристику розгортання програмного забезпечення, це, наприклад, може бути максимальний час роботи системи, про який ми згадаємо нижче;

$f_2(\bar{X})$ – друга цільова функція;

$f_n(\bar{X})$ – $n$-а цільова функція;

З обмежень наступного виду

$$\begin{aligned} g_k(\bar{X}) \leq\ \alpha_k,\ \ k = (1, l), \\ h_m(\bar{X}) \leq\ \beta_m,\ \ m = (1, j), \end{aligned} \qquad (2)$$

де $g_k(\bar{X}) \leq\ \alpha_k$ – $k$-те обмеження нерівності;

$h_m(\bar{X}) \leq\ \beta_m$ – $m$-те обмеження рівності;

$\alpha_k$ та $\beta_m$ це деякі константи, $l$ та $j$ відповідно кількість обмежень нерівностей та рівностей.

На основі прикладів реальних програмних проєктів, досвіду розробників та вимог до програмної системи, ми визначили, що головним критерієм оптимальності є фінансові витрати потрібні для роботи системи. Чим нижче фінансові витрати потрібні для роботи при тих самих показниках стабільності та доступності програмного забезпечення, тим краще для стейкхолдерів, а, відповідно, і розробників проєкту. Тоді задачу багатокритеріальної оптимізації, яка ми будем розв'язувати за допомогою методу головної компоненти, можна описати так:

$$\min f_n(\bar{X}), \tag{3}$$

за умови виконання наступних обмежень:

$$\begin{gathered} f_1(\bar{X}) \geq A_1, f_2(\bar{X}) \geq A_2, \dots, f_{n-1}(\bar{X}) \leq A_{n-1}, \\ g_k(\bar{X}) \leq \alpha_k, (k = 1, l), \\ h_m(\bar{X}) = \beta_m, (m = 1, j), \\ A_{<i>} = < A_1, A_2, \dots, A_n >, \end{gathered} \tag{4}$$

де $A_1 (i = 1, 2, \dots, n)$ – граничні значення критеріїв оптимальності.

Метод головної компоненти полягає в обранні одного з критеріїв як головного для проведення оптимізації та вибору рішення. При цьому, усі інші критерії перетворюються в обмеження.

Цей метод має просту та наочну реалізацію і широко застосовується в промисловості та інженерії. Однак, основним недоліком є довільність у виборі головного критерію, що може призвести до малоефективних результатів або навіть трагічних наслідків, як це сталося у багатьох історичних випадках.

В нашому випадку, враховуючи вибір предметної галузі, професійний досвід та особливості реалізації та використання програмного забезпечення, ми можемо знехтувати недоліками та обрати цей метод для свого рішення.

Залишається обрати критерії, які будуть описувати наші альтернативи, якими є різні способи розгортання програмного забезпечення на хмарній платформі AWS, обчислити значення цільової функції, враховуючи поставлені обмеження, та зробити висновок щодо доцільності використання того чи іншого методу розгортання в залежності від умов.

## 2.2 Методи розгортання програмного забезпечення

AWS надає декілька методів для розгортання програмного забезпечення на їхній платформі. Основні технічні властивості різних методів розгортання можуть суттєво відрізнятися, залежно від конкретного методу. Розглянемо ці методи. Але для початку, виділимо ті основні технічні характеристики, які будемо використовувати для порівняння цих методів меж собою:

1) фізична пам'ять: це обсяг пам'яті зберігання даних. Часто в рамках платформи AWS це може тип пам’яті, яка доступна, тобто сервіс, який використовується для обслуговування;
2) оперативна пам'ять: це обсяг пам'яті, який може бути виділений динамічно для використання програмою;
3) максимальний час роботи: це максимальний час, протягом якого може працювати сервер без перезавантаження;
4) максимальна потужність процесору: це максимальна потужність процесору, яка доступна для використання на сервері;
5) максимальна пропускна здатність мережі: це максимальна кількість даних, які можуть бути передані через мережу за одиницю часу;
6) ціна: це вартість використання даного методу розгортання програмного забезпечення на платформі AWS за заданий проміжок часу або кількість використаних ресурсів.

Ці характеристики можуть відрізнятися для кожного методу розгортання на платформі AWS, і вони є важливими при виборі найбільш підходящого методу для конкретного застосування.

### 2.2.1 Serverless AWS Lambda

Serverless AWS Lambda – це сервіс AWS, який дозволяє виконувати код без необхідності управління серверами або інфраструктурою. Основна ідея за методом безсерверних обчислень полягає в тому, що ви можете завантажувати свій код (функції) на платформу, а вона буде автоматично масштабуватися та керувати життєвим циклом вашої програми.

Таким чином особливості використання цього методу розгортання є наступні пункти:

- serverless AWS Lambda є платформою для розгортання функцій на платформі AWS;
- використовується безсерверні підходи, що означає, що розробник не потрібно думати про інфраструктуру, вона забезпечується AWS;
- функції запускаються автоматично при певних подіях або запитах та повертають відповіді відповідно до визначеного коду;
- може бути використаний для обробки подій, реалізації бізнес-логіки, оптимізації обробки даних та багатьох інших випадків використання.

Тут і далі будемо описувати максимальні технічні характеристики, доступні при виборі кожного методу розгортання. Для цього надамо наступну таблицю (див. таблицю 1).

Таблиця 1 – технічні характеристики сервісу AWS Lambda

| Пам’ять | Оперативна пам’ять | Час роботи | Потужність процесора | Пропускна здатність мережі | Ціна |
|---|---|---|---|---|---|
| EFS | 10 240 МБ | 15 хвилин | 6 vCPU | 1 Гбіт/с | 0.06$/год |

Серед особливостей цього методу можна звернути увагу на максимальний час роботи. Тут є обмеження в 15 хвилин, яке серйозно впливає на наш вибір. Але в той же час цей метод є одним з найбільш дешевих.

### 2.2.2 AWS EC2

AWS EC2 – це віртуальні машини (анг. Virtual Machines, VM), які можуть бути розгорнути по запиту користувача. Для віртуальної машини можна обрати різну операційну систему (Windows або Linux), забезпечується гнучкий доступ до налаштування системи. Це дозволяє розробникам розгортати власні сервери з встановленим програмним забезпеченням, такі як бази даних, веб-сервери, додатки, тощо. EC2 також дозволяє масштабувати ресурси за необхідністю, наприклад, збільшувати кількість машин з підключенням до єдиного об’єкта зберігання.

Таким чином, описати цей метод розгортання програм можна так:

- amazon Elastic Compute Cloud (EC2) є сервісом, який надає можливість запускати віртуальні сервери в хмарі AWS;
- забезпечує повний контроль над віртуальною машини та інфраструктурою, що означає, що розробник повинен налаштовувати сервери, забезпечувати масштабованість та безпеку серверів;
- дозволяє розгортати будь-який тип програмного забезпечення та може бути використаний для будь-якого випадку використання, якщо розробник знає, як правильно налаштувати сервер.

Для порівняння з іншими методами розгортання наведемо технічні особливості цього сервісу (див. таблицю 2).

Таблиця 2 – технічні характеристики сервісу AWS EC2

| Пам’ять | Оперативна пам’ять | Час роботи | Потужність процесора | Пропускна здатність мережі | Ціна |
|---|---|---|---|---|---|
| EBS | 12288 Гб | Необмежений | 448 vCPU | 100 Гбіт/с | 109$/год. |

Можна зазначити, що EC2 надає найбільш потужніший спосіб розгортання програмного забезпечення порівняно з іншими методами, але і найбільш дорожчий.

### 2.2.3 AWS ECS on Fargate

AWS ECS – це контейнерний сервіс, який дозволяє розгортати і керувати контейнерами Docker. ECS дозволяє забезпечувати різні рівні доступності та масштабованості на основі вимог, та розгортувати контейнери на різних інстансах EC2 або в моделі Fargate з урахуванням певних обмежень, таких як вільний розмір диску, доступність пам'яті та процесора. Виділимо наступне:

- Amazon Elastic Container Service (ECS) є сервісом, який дозволяє запускати та керувати контейнерами Docker в хмарі AWS;
- ECS забезпечує автоматичне масштабування та управління контейнерами;
- ECS може бути використаний для розгортання мікросервісів та розподілених систем;
- ECS дозволяє налаштовувати кластери та задачі, що дозволяє розробникам реалізовувати більшу кількість сценаріїв управлінням даними.

Так само, для порівняння з іншими методами виділимо технічні характеристики та обмеження, які впливають на вибір цього рішення. Для цього подивимось на таблицю 3.

Таблиця 3 – технічні характеристики сервісу AWS ECS

| Пам’ять | Оперативна пам’ять | Час роботи | Потужність процесора | Пропускна здатність мережі | Ціна |
|---|---|---|---|---|---|
| EFS | 120 ГБ | Необмежений | 16 vCPU | 1 Гбіт/с | 0.64$/год |

Перейдемо до опису наступного методу.

### 2.2.4 AWS Elastic Beanstalk

AWS Elastic Beanstalk: це сервіс управління рівнем абстракції, який дозволяє користувачам розгортати та масштабувати веб-додатки та сервіси з використанням зручного інтерфейсу. Elastic Beanstalk автоматично керує за вас конфігуруванням, моніторингом та масштабуванням ресурсів AWS, включаючи Amazon EC2-інстанції, балансування навантаження, мережеві налаштування та бази даних. Elastic Beanstalk надає зручний інтерфейс для налаштування різних параметрів, таких як розмір і тип інстанцій, що дозволяє змінювати розмір та масштаб приложення в залежності від потреб. Технічні характеристики цього методу розгортання доступні наведені в таблиці 4.

Таблиця 4 – технічні характеристики сервісу AWS Elastic Beanstalk

| Пам’ять | Оперативна пам’ять | Час роботи | Потужність процесора | Пропускна здатність мережі | Ціна |
|---|---|---|---|---|---|
| EBS | 128 Гб | Необмежений | 16 vCPU | 10 Гбіт/сек | 1.32$/год |

Важливість цих властивостей для конкретного проекту може залежати від його потреб та вимог до продуктивності, надійності та доступності. Наприклад, якщо додаток потребує значної кількості пам'яті, AWS EC2 може бути більш підходящим варіантом, ніж Serverless AWS Lambda. Однак, Serverless AWS Lambda може бути кращим вибором для малих функцій або подій з коротким часом виконання.

# 3 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Формування вимог є важливим етапом, який визначає подальші кроки реалізації, розгортання та впровадження програмного забезпечення. Вимоги впливають на те, які система буде використовуватися користувачами. Отже, на основі задач, що стоять перед вами, можна сформулювати вимоги до програмної системи:

- надійність: програмна система повинна бути стійкою до відмов, забезпечувати коректну роботу та відповідати вимогам безпеки;
- масштабованість: програмна система повинна бути здатною працювати з великими обсягами даних та обслуговувати багатокористувацьку роботу;
- ефективність: програмна система повинна бути ефективною з точки зору швидкодії та використання ресурсів;
- модульність: програмна система повинна бути побудована з окремих модулів, які можуть бути змінені та модифіковані без зміни решти системи;
- гнучкість: програмна система повинна бути здатною працювати з різними класами інформаційних систем та досліджувати різні методи оцінювання їх ефективності;
- легкість у використанні: програмна система повинна мати інтуїтивний і зрозумілий інтерфейс користувача, щоб навіть користувачі без досвіду змогли працювати з нею.

Ці функціональні та нефункціональні вимоги будуть використовуватися при проєктуванні програмної системи, а також будуть деталізовані в ході розробки.

## 3.1 Ключові функціональні вимоги до системи

Програмна система буде складатися із програмного модуля, який буде розгорнутий в хмарному середовищі, матиме доступ до основного середовища

виконання та зможе аналізувати метрики роботи сервісів, які потрібно проаналізувати. На основі цієї інформації програма буде приймати рішення про доцільність вибору того чи іншого засобу розгортання сервісів та пропонувати більш оптимальний вибір моделі розгортання в залежності від обраної метрики, наприклад, фінансових витрат.

Основні ключові функціональні вимоги до програмної системи для вибору способу розгортання програмного забезпечення в залежності від наявних параметрів можуть бути наступними:

1) аналіз параметрів системи: програмна система повинна здійснювати аналіз основних параметрів системи, таких як тип операційної системи, наявність пам'яті, обсяг дискового простору, кількість ядер процесора та інші, для визначення оптимального способу розгортання ПЗ;
2) порівняння способів розгортання: програмна система повинна порівнювати різні способи розгортання програмного забезпечення, такі як локальна інсталяція, віртуалізація, контейнеризація, хмарні технології тощо, для визначення найбільш оптимального варіанту;
3) вибір оптимального способу розгортання: на основі аналізу параметрів системи та порівняння різних способів розгортання програмного забезпечення, програмна система повинна вибрати найбільш оптимальний спосіб для конкретної системи;
4) моніторинг процесу розгортання: програмна система повинна забезпечувати моніторинг процесу розгортання програмного забезпечення, включаючи перевірку наявності необхідних ресурсів та відслідковування помилок та відхилень;
5) автоматизація процесу розгортання: програмна система повинна надавати можливість автоматичного розгортання програмного забезпечення з використанням оптимального способу, що зменшить зусилля розробників та час на розгортання;
6) підтримка різних інформаційних систем: програмна система повинна підтримувати різні класи інформаційних систем, такі як веб-додатки;

7) модуль для оцінки та порівняння різних методів розгортання програмного забезпечення:
   - розробити алгоритми для оцінки ефективності різних методів розгортання програмного забезпечення;
   - реалізувати модуль для порівняння різних методів розгортання програмного забезпечення;
   - модуль повинен допомагати вибрати найкращий метод розгортання програмного забезпечення для конкретної інформаційної системи.
8) модуль для моніторингу процесу розгортання програмного забезпечення:
   - розробити алгоритми моніторингу процесу розгортання програмного забезпечення;
   - реалізувати модуль для моніторингу процесу розгортання програмного забезпечення;
   - модуль повинен забезпечувати візуалізацію даних процесу розгортання та давати змогу збирати статистику про розгортання програмного забезпечення.
9) модуль для збору інформації про характеристики інформаційної системи:
   - розробити алгоритми збору та аналізу інформації про характеристики інформаційної системи;
   - реалізувати модуль для збору інформації про характеристики інформаційної системи;
   - модуль повинен забезпечувати збір інформації про характеристики апаратної та програмної частини інформаційної системи.
10) модуль для оцінки ризиків:
   - розробити алгоритми для оцінки ризиків різних методів розгортання програмного забезпечення;
   - реалізувати модуль для оцінки ризиків різних методів розгортання програмного забезпечення;
   - модуль повинен допомагати вибрати метод розгортання.

В процесі розробки ці вимоги можуть бути більш деталізовані або змінені в залежності від деталей реалізації.

## 3.2 Припущення та залежності

Деякі припущення та залежності, які можуть бути прийняті в процесі розробки програмної системи, включають наступні:

1) припущення про наявність певного рівня розуміння програмування і розгортання програмного забезпечення серед користувачів системи;
2) залежність від середовища розгортання програмної системи, включаючи операційну систему та програмні середовища;
3) залежність від наявності інших програмних сервісів, необхідних для правильної роботи програмної системи, наприклад баз даних, систем контролю версій, систем моніторингу тощо;
4) припущення про стійкість мережі під час роботи системи;
5) припущення про те, що користувачі системи мають достатні права доступу до системи, щоб виконувати операції, необхідні для розгортання програмного забезпечення.

Ці припущення та залежності потрібно враховувати при розробці програмної системи та під час її експлуатації. Вони можуть впливати на функціональність системи та її ефективність.

## 3.3 Версії системи

Реліз-план для програмної системи, яка допомагає розробникам обирати спосіб розгортання програмного забезпечення, може бути наступним:

1) реліз 1.0 (перша ітерація):
   – реалізація основного функціоналу системи, включаючи вибір методу

розгортання програмного забезпечення в залежності від параметрів і отримання результатів;

- підтримка різних класів інформаційних систем;
- тестування і валідація системи.

2) реліз 1.1 (друга ітерація):
   - додаткова функціональність, що дозволяє розробникам налаштовувати параметри і отримувати більш детальну інформацію про результати;
   - покращення інтерфейсу користувача.
3) реліз 1.2 (третя ітерація):
   - підтримка інтеграції з іншими системами інфраструктури, такими як Docker, Kubernetes тощо;
   - додаткові методи оцінювання ефективності розгортання.
4) реліз 2.0 (четверта ітерація):
   - підтримка мульти-облікових записів користувачів, що дозволить користувачам використовувати систему з різних облікових записів;
   - розширення підтримки класів інформаційних систем.

Крім того, кожен реліз повинен включати покращення безпеки та виправлення помилок, виявлених у попередніх версіях. Цей реліз-план може бути змінений залежно від нових вимог та відгуків користувачів

Загалом, випуск програмного забезпечення надасть розробникам інструмент для оцінки ефективності вибору засобу розгортання програмного забезпечення. У подальших релізах буде розроблено додаткові функції для покращення програми.

## 3.4 Користувацькі обмеження

Наведу деякі можливі обмеження, які можуть вплинути на результати та висновки дослідження:

1) особливості тестового середовища: можуть виникнути проблеми з тестуванням системи на реальному середовищі через відмінності від тестового середовища, що може вплинути на результати тестування;
2) обмеження апаратного забезпечення: можуть існувати обмеження на використання певних апаратних засобів, які можуть обмежувати продуктивність системи і впливати на результати дослідження;
3) обмеження на кількість одночасних користувачів: якщо система розрахована на використання декількома користувачами одночасно, то обмеження на кількість одночасних користувачів може вплинути на продуктивність системи і впливати на результати дослідження;
4) обмеження на розмір бази даних: якщо програмна система використовує базу даних, то обмеження на розмір бази даних може впливати на продуктивність системи і впливати на результати дослідження;
5) обмеження на доступ до мережі: якщо програмна система потребує доступу до мережі, то обмеження на доступ до мережі може впливати на продуктивність системи і впливати на результати дослідження;
6) обмеження на час тестування: обмеження на час тестування може вплинути на кількість тестів, які можуть бути виконані, і впливати на точність та репрезентативність результатів дослідження.

Іншим обмеженням можуть бути технічні особливості системи, на які впливає швидкість обробки даних, наявність певних ресурсів (наприклад, обсягу пам'яті, доступної для використання), а також підтримка технологічних стеків, які використовуються в системі.

Також можуть бути обмеження з точки зору зручності використання, такі як інтуїтивно зрозумілий інтерфейс та документація, доступна для користувачів. Важливо враховувати ці обмеження під час розробки системи та узгоджувати їх зі стейкхолдерами, щоб забезпечити максимальну ефективність та задоволення від користування програмним додатком.

# 4 ОПИС ПРОГРАМНИХ РІШЕНЬ

## 4.1 Аналіз існуючих рішень

Існує багато програмних рішень, які допомагають оцінювати ефективність розгортання програмного забезпечення для різних класів інформаційних систем. Розглянемо наступні:

1) DeployHub: DeployHub – це програмне забезпечення з відкритим кодом, яке дозволяє розгортати ПЗ і відстежувати його версії. DeployHub також допомагає оцінити ефективність розгортання, враховуючи фактори, такі як швидкість розгортання, кількість помилок і доступність системи;
2) Puppet Enterprise: Puppet Enterprise – це інструмент автоматизації розгортання, який дозволяє розгортати програмне забезпечення на різних платформах. Puppet Enterprise має інтегрований моніторинг та аналіз результатів розгортання програмного забезпечення, що дозволяє оцінювати ефективність розгортання;
3) IBM UrbanCode: IBM UrbanCode – це програмне забезпечення для автоматизації розгортання, яке допомагає управляти розгортанням програмного забезпечення та забезпечує зручний інтерфейс для оцінювання ефективності розгортання. Він має інтегрований інструмент аналізу даних, що дозволяє зіставляти розгорнуті версії програмного забезпечення з вимогами користувачів;
4) XebiaLabs: XebiaLabs – це програмне забезпечення для автоматизації розгортання, яке дозволяє оцінювати ефективність розгортання, враховуючи фактори, такі як швидкість розгортання, кількість помилок і доступність системи. XebiaLabs дозволяє розгортати програмне забезпечення на різних платформах і має інтегрований моніторинг.

Ще одним прикладом існуючого програмного рішення, яке може допомогти в оцінці ефективності розгортання програмного забезпечення, є Dynatrace. Це програмне забезпечення забезпечує моніторинг роботи інформаційних систем та

додатків, яке дозволяє виявляти помилки, що стосуються розгортання програмного забезпечення.

Dynatrace використовує низку алгоритмів машинного навчання, щоб прогнозувати та забезпечувати більш точний аналіз стану системи. Він також забезпечує інформацію про швидкість виконання процесів та їх інтерактивність, що дозволяє виявляти помилки та несправності в роботі системи.

Іншим прикладом є система моніторингу та аналізу додатків AppDynamics. Вона забезпечує можливість моніторингу стану додатку, його ефективності та продуктивності, а також забезпечує інформацію про те, як додаток впливає на систему в цілому. AppDynamics дозволяє виявляти помилки та недоліки в роботі системи, а також дає змогу користувачам створювати спеціальні звіти, які допомагають визначати ефективність розгортання програмного забезпечення.

Крім того, існує ряд відкритих програмних продуктів, які можуть допомогти в оцінці ефективності розгортання програмного забезпечення, таких як Prometheus, Grafana, Nagios та інші. Вони забезпечують інформацію про стан системи, її ефективність та продуктивність, а також дають змогу виявляти помилки та буття недоліків у роботі системи.

Усі ці програмні рішення мають свої особливості та переваги, але їх здатність оцінювати ефективність розгортання програмного забезпечення відрізняється. Для дослідження ефективності розгортання програмного забезпечення для різних класів інформаційних систем можуть бути потрібні інші рішення або їх комбінації.

## 4.2 Аналіз обраних технологій

Переходячи безпосередньо до реалізації, потрібно обрати технології та інструменти, які будуть використовуватися для реалізації. Опишемо їх нижче:

- мови програмування: C# як основна мова програмування для розробки хмарних сервісів, Python як мова обробки даних, оскільки має значну

кількість бібліотек, та TypeScript як мова розробки користувацького інтерфейсу, якщо такий буде потрібен. Вибір мови програмування залежить від вимог до швидкодії та масштабованості системи, а також продиктований особливості розгортання та використання програмного забезпечення;

- фреймворки: .NET Core, Flask, Django, та інші. Фреймворки допоможуть спростити процес розробки та реалізації функціональності системи;
- бібліотеки: Pandas, NumPy, SciPy, Matplotlib, Scikit-learn. Бібліотеки допоможуть забезпечити аналітичні можливості та візуалізацію даних;
- утіліти: Docker. Утиліти допоможуть забезпечити ефективне розгортання та управління програмним забезпеченням;
- середовища розробки: Visual Studio та Visual Studio Code. Середовища розробки допоможуть забезпечити зручний процес розробки та тестування програмної системи.

Реалізація програмного забезпечення на основі фреймворку .NET Core та лямбда-функцій AWS має декілька особливостей.

Починаючи з фреймворку .NET Core, він забезпечує можливість розробки кросплатформеного програмного забезпечення, що працює на будь-якій платформі. Це забезпечує більшу гнучкість та доступність для розробників. Для реалізації лямбда-функцій AWS можна використовувати фреймворк AWS Lambda, який дозволяє створювати функції на C#, Python, Node.js та інших мовах програмування. Цей фреймворк дозволяє створювати функції, які можуть бути автоматично масштабовані в залежності від навантаження.

Для забезпечення безпеки можна використовувати сервіс AWS Identity and Access Management (IAM), який дозволяє керувати доступом до ресурсів AWS та обмежувати права доступу користувачів. Для зберігання даних можна використовувати сервіси AWS, такі як Amazon S3 для зберігання об'єктів, Amazon RDS для зберігання реляційних даних та Amazon DynamoDB для зберігання даних в форматі NoSQL.

Загалом, використання фреймворку .NET Core та лямбда-функцій AWS дозволяє створювати гнучке, масштабоване та безпечне програмне забезпечення.

Якщо говорити про аналіз даних, який буде потрібний для висновків щодо доцільності роботи користувацьких сервісів, то тут існує багато бібліотек на мові Python, які можна використовувати з цією метою:

1) NumPy: бібліотека для роботи з числовими масивами та матрицями;
2) Pandas: бібліотека для обробки та аналізу даних у вигляді табличних структур, таких як DataFrames;
3) Matplotlib: бібліотека для створення графіків та візуалізації даних;
4) Seaborn: бібліотека для візуалізації даних, яка забезпечує більш високорівневий інтерфейс, ніж Matplotlib;
5) Scikit-learn: бібліотека для машинного навчання, яка надає реалізації багатьох алгоритмів машинного навчання та інструментів для оцінки результатів.

Ці бібліотеки дозволяють здійснювати аналіз даних на мові Python, забезпечуючи зручний та потужний інструментарій для роботи з даними та їх візуалізацією.

Реалізація програмного забезпечення передбачає зберігання та обробку даних. Для цього необхідно обрати правильні методи збереження та серіалізації.

Одним з можливих варіантів серіалізації та збереження даних є використання формату JSON (JavaScript Object Notation). Цей формат є досить популярним в сфері розробки програмного забезпечення та має багато переваг, таких як читабельність, простота та зручність для роботи з даними. Для роботи з даними у форматі JSON можна використовувати різноманітні бібліотеки на мові програмування, яка буде використовуватися для реалізації програмного забезпечення. Наприклад, у Python для роботи з JSON-даними можна використовувати бібліотеку json. Вона містить методи для серіалізації та десеріалізації JSON-об'єктів та надає можливість зберігати дані у вигляді рядків або файлів.

Для збереження даних у програмному забезпеченні можна використовувати різноманітні бази даних, такі як MySQL, PostgreSQL, MongoDB та інші. У залежності від потреб проекту можна обрати найбільш підходящу базу даних та використовувати відповідні бібліотеки для роботи з нею.

Однією з особливостей роботи з даними є необхідність їх захисту. Для цього можна використовувати різноманітні методи шифрування та захисту даних. Наприклад, у Python для шифрування даних можна використовувати бібліотеку cryptography, яка надає можливість використовуват

Для розгортання програмного рішення в хмарному середовищі можна використовувати підхід Infrastructure as Code (IaC), що дозволяє описувати інфраструктуру, необхідну для роботи програмного забезпечення, в коді.

Для цього можна використати такі технології, як Terraform або CloudFormation, які дозволяють описувати інфраструктуру у вигляді коду та автоматизовувати процес її розгортання. Також можна використовувати Docker для контейнеризації додатку. Для забезпечення безпеки та масштабовності рекомендується використовувати автоматичні засоби моніторингу та розширення інфраструктури, такі як AWS Auto Scaling або Kubernetes Horizontal Pod Autoscaler.

Особливості серіалізації та збереження даних в хмарному середовищі пов'язані з використанням різних сервісів для збереження даних, таких як бази даних, об'єктні сховища тощо. Рекомендується використовувати засоби для автоматичної реплікації та резервного копіювання даних, наприклад, AWS RDS або Amazon S3, щоб забезпечити надійність та доступність даних. Також можна використовувати різні формати серіалізації даних, наприклад JSON або CSV, залежно від потреб програмного рішення та використовуваних сервісів.

Для реалізації користувацького інтерфейсу можна використовувати різноманітні технології та мови програмування залежно від вимог проекту та вмінь команди розробників. Основні технології, які використовуються для створення користувацьких інтерфейсів, включають: HTML, CSS та JavaScript – ці технології використовуються для створення веб-інтерфейсів. HTML

використовується для створення структури сторінки, CSS – для задання стилів та вигляду елементів, а JavaScript – для динамічної зміни вмісту сторінки та взаємодії з користувачем.

Для побудови інтерактивних елементів користувацького інтерфейсу можна використовувати бібліотеки, такі як jQuery та D3.js. jQuery – це бібліотека JavaScript, яка дозволяє зручно працювати з DOM-деревом, обробляти події та анімувати елементи. D3.js – це бібліотека для візуалізації даних, яка дозволяє створювати складні візуалізації з допомогою SVG та Canvas.

## 4.3 Аналіз архітектури системи

Архітектура системи програмного рішення може бути розбита на три головні складові: серверну частину, користувацький інтерфейс та базу даних.

Серверна частина програмного рішення буде розроблена на базі фреймворка .NET Core та буде складатися з низки мікросервісів, які будуть забезпечувати різні функціональні можливості програмного рішення, включаючи автоматизацію процесу розгортання програмного забезпечення, оцінювання ефективності розгортання та зберігання даних. Для забезпечення масштабованості, стійкості та надійності системи, можна використовувати технології контейнеризації, такі як Docker та Kubernetes.

Користувацький інтерфейс програмного рішення буде реалізований за допомогою веб-технологій, таких як HTML, CSS та JavaScript. Для розробки можна використовувати фреймворки, такі як React або Angular в залежності від складності користувацького інтерфейсу та кількості функцій, які потрібно надати користувачу. В першій ітерації додатку пропонується обмежитися простим інтерфейсом командного рядку.

База даних буде використовувати SQL, зокрема може бути використана база даних Amazon Relational Database Service (RDS), що забезпечує швидкий та

надійний доступ до даних та забезпечує високу стійкість та безпеку. Також можна використовувати NoSQL базу даних, таку як Amazon DynamoDB.

В цілому, архітектура системи програмного рішення описаного вище може бути представлена як архітектура, яка дозволяє забезпечити високу масштабованість, доступність та ефективність програмного забезпечення, а також дозволяє зменшити витрати на інфраструктуру, оскільки використовується хмарне середовище та serverless-технології.

## 4.4 Архітектура баз даних

Для зберігання даних в програмному рішенні, описаному вище, буде використовуватися документоорієнтована NoSQL база AWS DynamoDB. Структура бази даних буде складатися з декількох таблиць, які будуть містити дані про сервіси, метрики, історію запусків та інформацію, яка потрібна для аналітики, яка є необхідною для правильної роботи системи.

Основні таблиці, які будуть використовуватися в системі, включають:

1) таблиця «Сервіси» – містить основну інформацію про сервіси, які аналізуються додатком, таку як назва, технічні характеристики, аналітику та інше;
2) таблиця «Історія» – містить історію викликів сервісів для прогнозування;
3) таблиця «Сервіси-Історія» – містить інформацію про зв'язок між сервісами та історією їхніх запусків;
4) таблиця «Аналітика» – містить інформацію про аналітику, яка потрібна для роботи додатку;
5) таблиця "Налаштування" - містить різноманітні налаштування системи, такі як тайм-аут сесії, максимальний розмір файлів, дозволені розширення файлів та інші налаштування.

Кожна таблиця міститиме свій унікальний ідентифікатор (первинний ключ), який може бути використаний для зв'язку з іншими таблицями за допомогою

зовнішніх ключів. Для забезпечення швидкого доступу до даних та зменшення кількості запитів до бази даних буде використано кешування на рівні додатку.

Для організації доступу до даних на рівні програмного коду можна використовувати ORM-бібліотеки, такі як Entity Framework для .NET Core. Ця бібліотека надає можливість взаємодіяти з базою даних у вигляді об'єктів і використовувати мову запитів, що зрозуміла для розробника, замість написання SQL запитів вручну. Також можна буде використовувати ORM-бібліотеку для побудови складних запитів до бази даних, що дозволить швидко та ефективно отримувати необхідну інформацію. ORM-бібліотеки також забезпечують безпеку від SQL-ін'єкцій та автоматичну перевірку типів даних, що дозволяє уникнути багатьох типових помилок, пов'язаних з роботою з базою даних. Крім того, використання ORM-бібліотек спрощує роботу з базою даних, знижує час розробки і підтримки коду.

База даних буде розміщена в хмарному середовищі. За допомогою відповідних налаштувань та політик безпеки, буде забезпечена захист даних від несанкціонованого доступу.

## 4.5 Архітектура серверної частини

Серверна частина програмної реалізації наведеної вище буде базуватися на фреймворку .NET Core та лямбда функціях AWS.

Для реалізації серверної частини буде використовуватися мова програмування C#, оскільки вона є однією з найбільш підходящих мов для розробки веб-додатків на платформі .NET. Крім того, вона має велику кількість фреймворків та бібліотек, що значно полегшує розробку.

Взаємодія з базою даних буде здійснюватися за допомогою Entity Framework Core – це бібліотека .NET, яка забезпечує доступ до реляційних баз даних. Entity Framework Core дозволяє працювати з базою даних за допомогою об'єктів, що значно полегшує роботу з даними та зменшує кількість коду.

Для роботи з AWS Lambda буде використовуватися AWS SDK для .NET, який забезпечує доступ до AWS-сервісів та можливості їх використання в додатках .NET. Завдяки цьому, можна створювати лямбда-функції та забезпечувати їх зв'язок з іншими AWS-сервісами.

Взаємодія з користувачем буде здійснюватися за допомогою HTTP протоколу. Для цього буде використовуватися ASP.NET Core, який надає можливість розробляти веб-додатки та API на .NET. Для забезпечення безпеки та автентифікації буде використовуватися протокол HTTPS та токени авторизації, такі як JWT.

Для забезпечення обробки запитів буде використовуватись веб-сервер Nginx, який буде приймати запити від клієнтів та перенаправляти їх до відповідних мікросервісів. Для розробки мікросервісів буде використовуватись фреймворк .NET Core, який забезпечує швидкість та надійність обробки запитів.

## 4.6 Процес отримання метрик сервісу

AWS API дозволяє отримувати різноманітні метрики, пов'язані зі станом середовища виконання та лямбда функцій. Для отримання метрик застосовується підхід опитування метрик з використанням запитів API. Для отримання метрик лямбда функцій можна використовувати AWS CloudWatch API. Для цього необхідно виконати наступні кроки:

1) створити CloudWatch метрику для лямбда функції, яку потрібно моніторити;
2) створити ім'я метрики та опис, які дозволять ідентифікувати метрику;
3) налаштувати відстеження метрики, щоб AWS могла збирати дані про лямбда функцію;
4) використовувати AWS SDK, щоб виконати запит до CloudWatch API та отримати дані про метрику лямбда функції.

Також, для отримання метрик стану середовища виконання можна використовувати AWS X-Ray API. Для цього необхідно виконати наступні кроки:

1) додати X-Ray SDK до лямбда функції, яку потрібно моніторити;
2) налаштувати X-Ray SDK для відправлення даних моніторингу до X-Ray;
3) використовувати AWS SDK, щоб виконати запит до X-Ray API та отримати дані про стан середовища виконання.

На рисунку 1 наведений приклад коду на мові C# для отримання метрик. Проаналізуємо та опишемо цей код детальніше.

```
using System;
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

namespace MyNamespace
{
    class Program
    {
        static void Main(string[] args)
        {
            var cloudWatchClient = new AmazonCloudWatchClient();

            // Отримання метрик щодо середовища виконання
            var request = new ListMetricsRequest
            {
                MetricName = "Invocations",
                Namespace = "AWS/Lambda",
                Dimensions = new List<DimensionFilter>
                {
                    new DimensionFilter
                    {
                        Name = "FunctionName",
                        Value = "MyLambdaFunction"
                    }
                }
            };

            var response = cloudWatchClient.ListMetrics(request);
            foreach (var metric in response.Metrics)
            {
                Console.WriteLine("Namespace: " + metric.Namespace);
                Console.WriteLine("MetricName: " + metric.MetricName);
                foreach (var dimension in metric.Dimensions)
                {
                    Console.WriteLine("Dimension: " + dimension.Name + "=" + dimension.Value);
                }
                Console.WriteLine();
            }

            Console.ReadLine();
        }
    }
}
```

Рисунок 1 – приклад коду на мові C# для отримання метрик середовища

У цьому прикладі ми створюємо клієнт AmazonCloudWatchClient, використовуючи клас з бібліотеки AWS SDK для .NET. Далі, ми формуємо запит на отримання метрик за допомогою ListMetricsRequest, вказуючи потрібні параметри, такі як назва метрики (MetricName), область, в якій ми шукаємо метрики (Namespace), і фільтри за розмірами (Dimensions). Отримані метрики зберігаються у відповіді (response) в форматі ListMetricsResponse, яку ми можемо розглянути за допомогою циклу foreach та вивести на консоль.

Метрики, отримані з AWS API, можна використовувати для аналізу середовища виконання на різних рівнях. Наприклад, метрики використання пам'яті та процесора можуть допомогти виявити проблеми з продуктивністю сервера. Якщо середовище виконання має недостатньо ресурсів, це може призвести до збоїв, зниження швидкості відповіді, збільшення часу виконання запитів, та ін.

Крім того, метрики можуть допомогти виявити проблеми зі складністю коду. Наприклад, відсутність оптимізації коду може призвести до того, що він використовує надто багато пам'яті чи процесорних ресурсів. Такі проблеми можуть бути виявлені за допомогою метрик та потребувати оптимізації коду.

Крім того, метрики можуть допомогти виявити проблеми зі збіркою сміття. Наприклад, якщо при виконанні програмного коду використовується занадто багато пам'яті, це може призвести до збільшення часу, потрібного для збору сміття. Це може призвести до зниження швидкості відповіді сервера, тому виявлення та виправлення таких проблем є важливою задачею для оптимізації продуктивності програмного коду.

## 4.7 Процес аналізу отриманих метрик

Наведемо інший приклад програмного коду. Так, це буде код на мові Python, який використовує Boto3 (AWS SDK для Python) для отримання метрик середовища виконання AWS Lambda та аналізу їх для визначення ефективності.

В цьому прикладі код отримує кількість викликів для кожної з функцій та виводить середнє значення цієї метрики за останні 15 хвилин. Звідси можна зробити висновки про те, як добре функції працюють та чи є проблеми зі швидкістю або використанням пам'яті. Також можна додатково аналізувати інші метрики, наприклад, тривалість виконання функції або використання пам'яті, щоб отримати більш детальну картину про те, як працює середовище виконання AWS Lambda (див. рисунок 2).

```python
# Аналіз метрик та виведення результатів
invocations = response['MetricDataResults'][0]['Values'][0]
avg_duration = response['MetricDataResults'][1]['Values'][0]
errors = response['MetricDataResults'][2]['Values'][0]

print(f"Function: {function_name}")
print(f"Invocations: {invocations}")
print(f"Avg. Duration: {avg_duration} ms")
print(f"Errors: {errors}")
if errors / invocations > 0.05:
    print("High error rate detected.")
if avg_duration > 500:
    print("Function duration too high.")


# Аналізуємо дані метрик
metric_data = response['Datapoints']
if len(metric_data) > 0:
    avg_duration = metric_data[0]['Average']
    if avg_duration > 500:
        print(f"Function {function_name} is taking too long to execute with an average duration of {avg_duration}ms.")
    else:
        print(f"Function {function_name} is performing well with an average duration of {avg_duration}ms.")
else:
    print(f"No metric data found for function {function_name} and metric {metric_name}.")
```

Рисунок 2 – приклад коду на мові Python для аналізу метрик

Аналіз метрик середовища виконання AWS Lambda може дати корисну інформацію щодо ефективності функції Lambda та допомогти виявити потенційні проблеми в процесі виконання. Деякі висновки, які можуть бути зроблені на основі аналізу метрик, включають:

1) час виконання: Метрика часу виконання може допомогти визначити, як швидко виконується функція Lambda та чи відповідає час виконання вимогам відповідності служби. Якщо час виконання функції занадто великий, можливо, потрібно перевірити код функції та знайти способи його оптимізації;

2) інтеграція з ресурсами: Метрика використання ресурсів може допомогти визначити, як часто функція використовує певні ресурси, такі як пам'ять та процесор. Це може бути корисно для виявлення проблем, пов'язаних з недостатньою кількістю ресурсів, які можуть призвести до помилок та зниження продуктивності;
3) помилки: Метрика помилок може допомогти визначити кількість помилок, які сталися під час виконання функції Lambda, та їх причини. Це може допомогти розібратися з проблемами в коді та зробити відповідні виправлення;
4) моніторинг: Метрики моніторингу можуть допомогти відслідковувати роботу функції Lambda в реальному часі та забезпечувати повідомлення про будь-які проблеми, що виникають в процесі виконання. Це може бути корисним для виявлення проблем, які виникають під час роботи функції в режимі реального часу та для швидкого реагування на них.

Цю інформацію можна використовувати для аналізу способу розгортання програмне забезпечення, адже саме він визначає характер використання ресурсів та їх ефективність.

# 5 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ

Використовуючи реалізований програмний сервіс та алгоритм аналізу показників середовища виконання, який має на меті зниження фінансових витрат підтримки рішення, проведемо експеримент, запустивши тестовий стенд із підготовленими сценаріями навантаження та перевіримо, який результат нам запропонує наш додаток. Наші очікування складаються в тому, що у визначених сценаріях, програмний сервіс буде знаходити недоліки в моделях розгортання програмного забезпечення (моделях розгортання, які визначають технічні характеристики середовища виконання) та пропонувати відповідні зміни в моделях розгортаннях, які мають на меті знизити наші витрати. Також, наші очікування в тому, що наша модель не пропонує змін у випадку ефективно обраної моделі розгортання, таким чином, не порушуючи те, що працює правильно та вірно. Перейдемо до сценаріїв виконання.

## 5.1 Сценарій короткочасних навантажень

Першим перевірим сценарій короткочасних навантажень. Такими, наприклад, можуть бути обробки подій, таких як запити API, відправка повідомлень, сповіщення, тригери баз даних тощо, побудова мікросервісної архітектури, обробка потокових даних в реальному часі, тощо. Будь які операції, коли потрібно виконати невелику частину роботи і закінчити виконання.

Наша задача впевнитись, що при такому сценарії, система запропонує спосіб розгортання, який найбільше підходить для цього, а саме Serverless AWS Lambda підхід. Цей підхід рекомендується, адже в такому випадку немає необхідності управління серверами або інфраструктурою. Більш того, враховуючи, що оплата за використання сервісу відбувається тільки, коли лямбда функція виконує корисну роботу, фінансові витрати є низькими.

Створимо тестове середовище виконання, яке буде складатися з EC2 інстанса. Запустимо тестову програму, яка буде виконувати короткочасне, але вагоме навантаження. Запустимо наш сервіс і поглянемо на результат аналізу (див. рисунок 3).

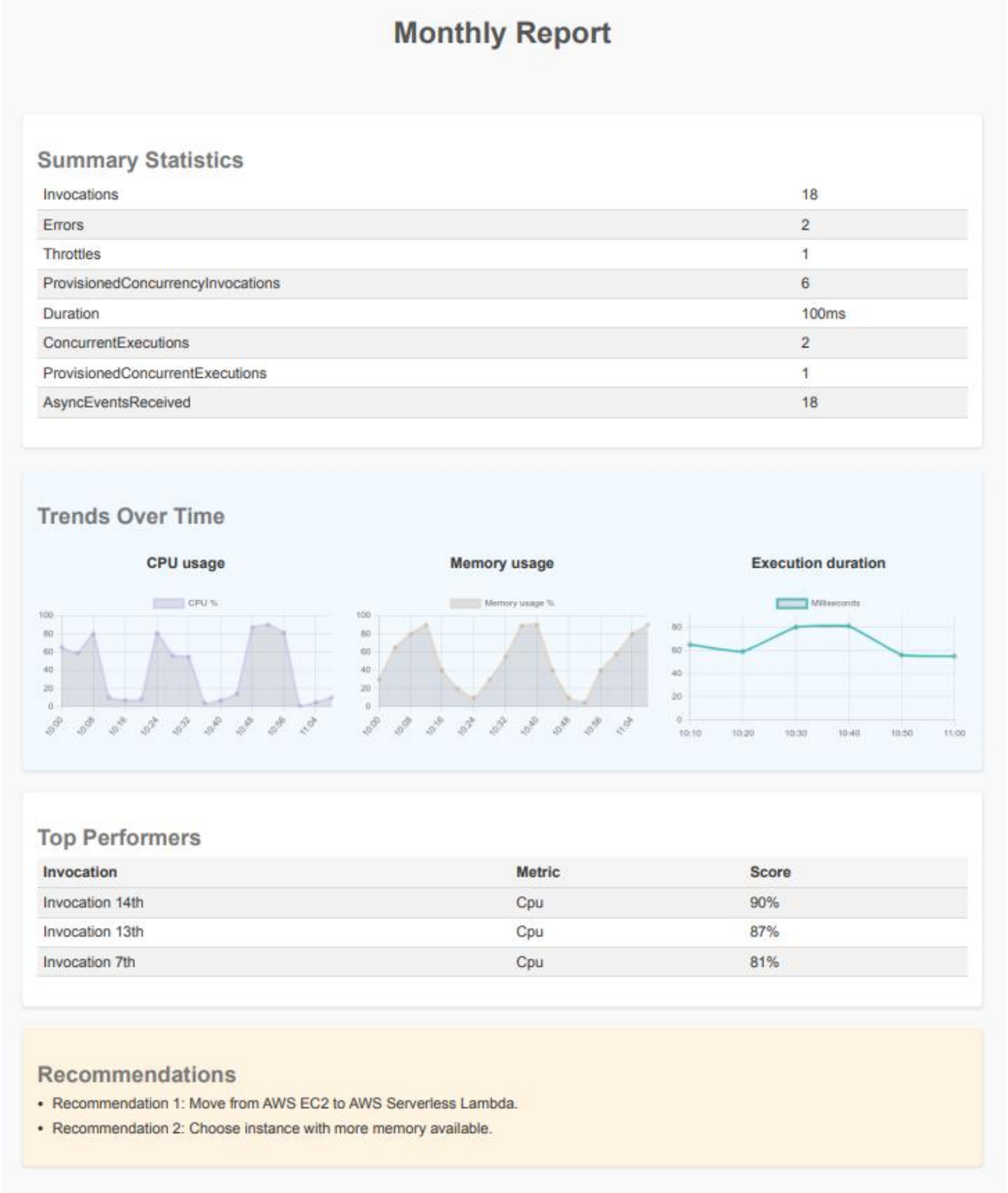



Рисунок 3 – репорт аналізу роботи тестового стенду EC2

Якщо поглянути на графіки метрик використання процесору та пам’яті, які були згенеровані по реальним даним навантаження нашого інстанса EC2, можна побачити короткі сплески до максимальних значень (див. рисунок 4).

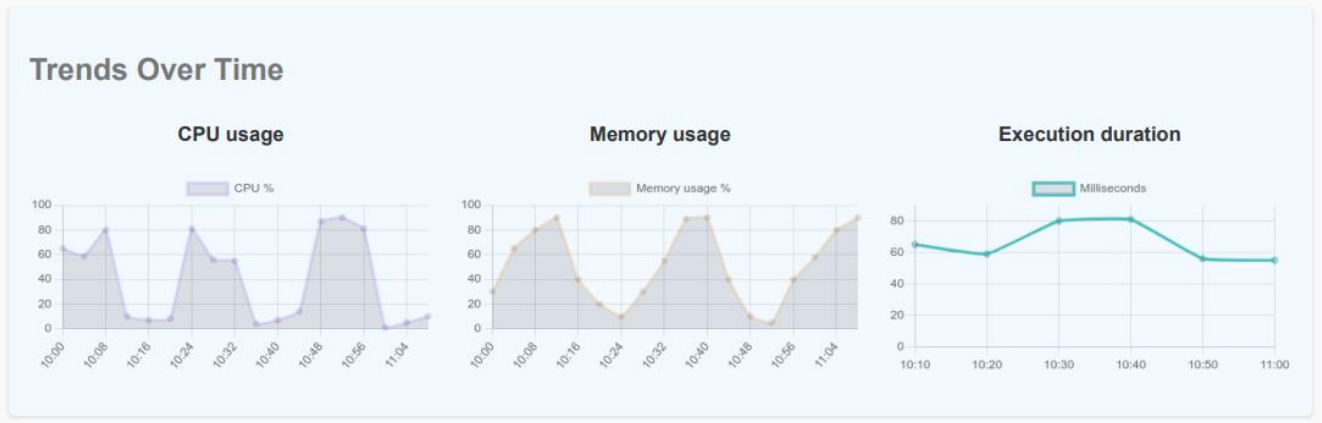



Рисунок 4 – графіки навантаження тестового стенду

Такі шаблони навантаження є вдалим прикладом для використання serverless підходу. І справді, в результаті аналізу ми отримуємо пораду зміни способу розгортання ПЗ (див. рисунок 5).

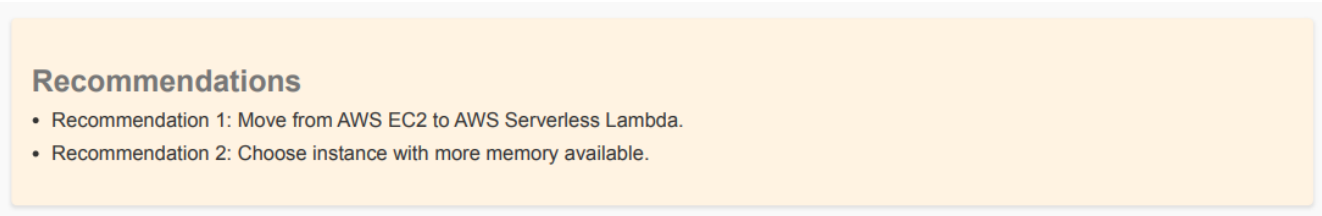


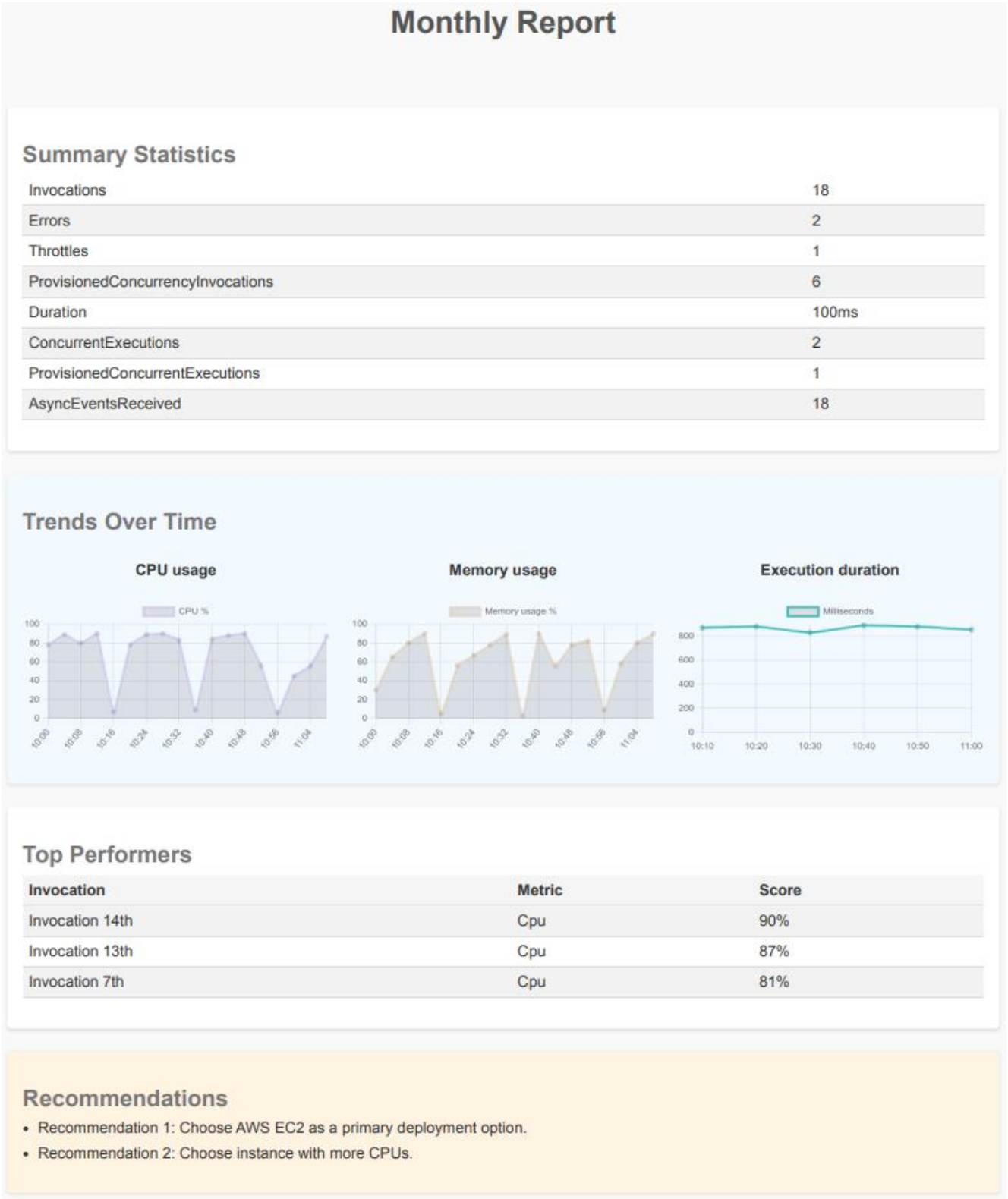

Рисунок 5 – поради щодо зміни способу розгортання в першому сценарії

Можна вважати перший сценарій тестування нашого додатку успішним. Перейдем до наступного сценарію.

## 5.2 Сценарій подовженого навантаження

В цьому випадку спробуємо перевірити протилежний випадок – навантаження постійне і використовує багато ресурсів. Тобто, використання лямбд є неможливим в принципі або не ефективним.

Повторимо наш крокі з підготовки даних та запуску. На цей раз тестовий стенд буде складатися з лямбди. Результат аналізу наведений на згенерованому репорту в рамках другого сценарію на рисунку 6.



# Monthly Report

## Summary Statistics

| | |
|---|---|
| Invocations | 18 |
| Errors | 2 |
| Throttles | 1 |
| ProvisionedConcurrencyInvocations | 6 |
| Duration | 100ms |
| ConcurrentExecutions | 2 |
| ProvisionedConcurrentExecutions | 1 |
| AsyncEventsReceived | 18 |

## Trends Over Time



## Top Performers

| Invocation | Metric | Score |
|---|---|---|
| Invocation 14th | Cpu | 90% |
| Invocation 13th | Cpu | 87% |
| Invocation 7th | Cpu | 81% |

## Recommendations

- Recommendation 1: Choose AWS EC2 as a primary deployment option.
- Recommendation 2: Choose instance with more CPUs.

Рисунок 6 – репорт аналізу роботи тестового стенду AWS Lambda

В цьому випадку можна помітити цікаву поведінку – на графіках використання ресурсів видно, що завантаження було максимальним (див. рисунок 7). Більше того, зниження використання ресурсів відбувалося кожні 15 хвилин – самі в ті моменти, коли лямбда закінчувала свою роботу, 15 хвилин є обмеженням платформи на максимальний час роботи.

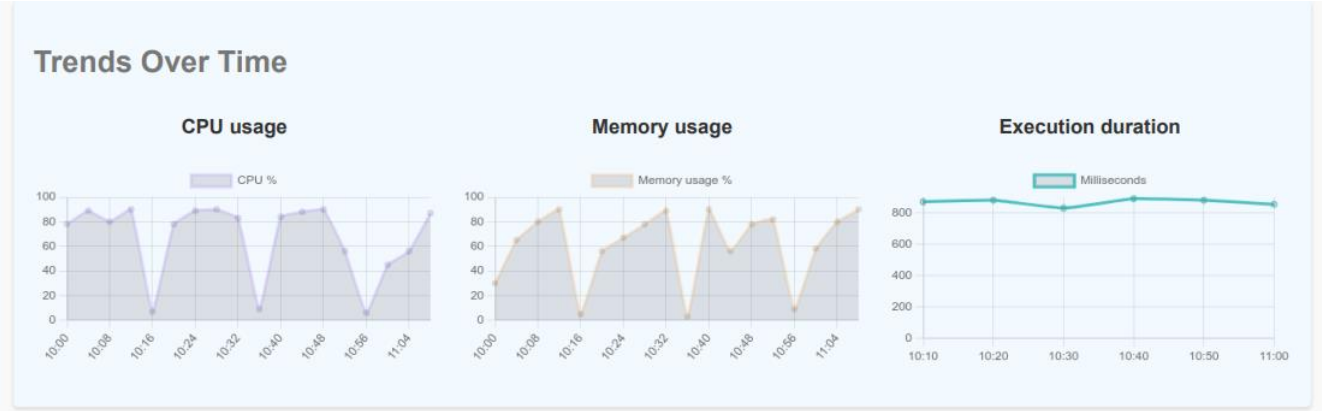



Рисунок 7 – графіки навантаження тестового стенду за другого сценарію

Тобто, наш тестовий стенд був завантажений увесь час, і був би і далі, якщо б не обмеження лямбд. Це може свідчити про потребу в зміні підходу. Поглянемо на результат аналізу, система пропонує схожий висновок (див. рисунок 8).

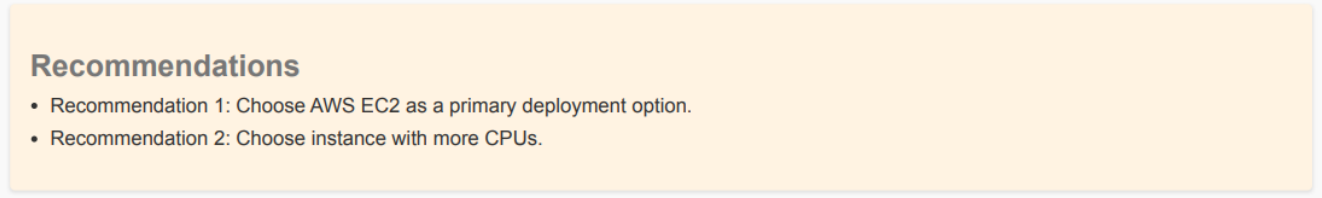



Рисунок 8 – поради щодо зміни способу розгортання в другому сценарії

Перевіримо останній сценарій аналізу роботи ПЗ і будемо переходити до висновків роботи.

## 5.3 Сценарій динамічної зміни навантаження

Останній сценарій навантаження буде полягати у рівномірному використанні ресурсів протягом довгого періоду часу. Часто, з саме таким можна стикнутися в

реальному світі. В такому випадку можна використовувати різні підходи, але пропонується варіант з AWS Fargate, адже він надає декілька абстракцій над управлінням ресурсами. Підготуємо тестовий стенд і запустимо сервіс аналізу. На рисунку 9 можна переглянути репорт підготовлений після запуску цього сценарію.

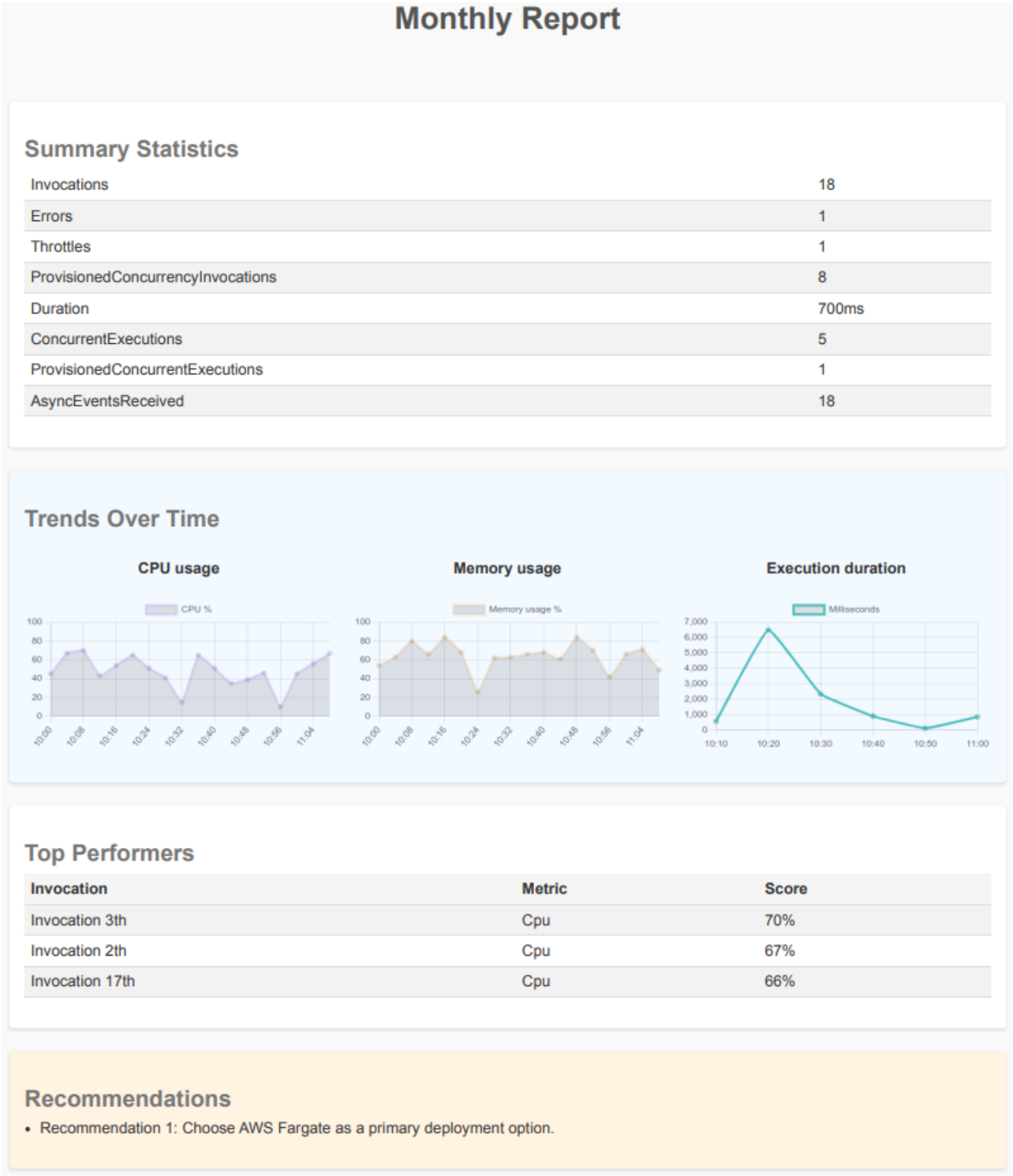
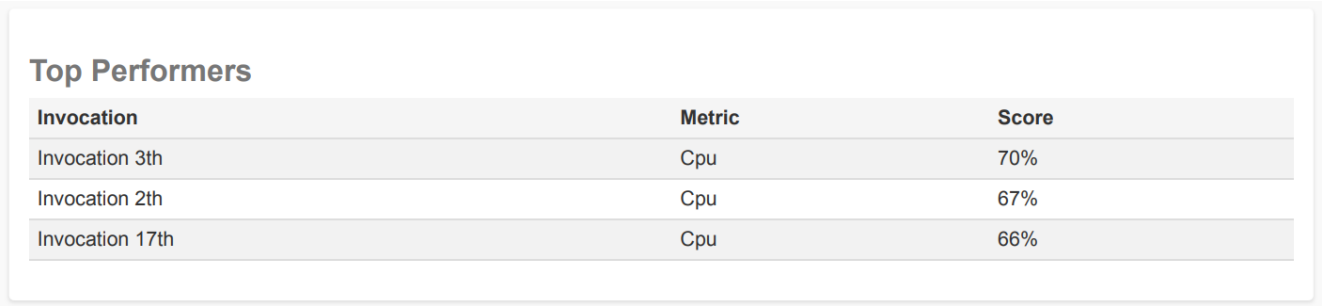



Рисунок 9 – репорт аналізу роботи тестового стенду AWS EC2

Характерно, в цьому випадку ми отримали сильний, але не максимальний, рівень завантаження ресурсів, що відображено на відповідних графіках. Це відображено і в таблиці найбільших показників в рамках цього сценарію (див. рисунок 10).



Top Performers

| Invocation | Metric | Score |
|---|---|---|
| Invocation 3th | Cpu | 70% |
| Invocation 2th | Cpu | 67% |
| Invocation 17th | Cpu | 66% |

Рисунок 10 – таблиця найбільших метрик використання ресурсів протягом тесту

Висновок, наведений внизу репорту свідчить про доцільність використання платформи розгортання AWS Fargate, що є хорошим вибором у випадку стабільних навантажень протягом довгого періоду часу з потребою зручного розгортання нових ресурсів.

# ВИСНОВКИ

Разом з переходом до широкого використання хмарної інфраструктури для розгортання програмного забезпечення відповідальністю розробників став аналіз фінансових витрат, які виникнуть при реалізації запропонованої архітектури чи архітектурного підходу. Враховуючи, що деякі технології є взаємозамінними в більшій частині випадків, як наприклад, часто, реляційну базу даних можна замінити на NoSQL і навпаки без серйозної деградації продуктивності, показник фінансових витрат є одним з визначальних факторів при виборі технології. Тому, є важливим вміти обрати правильну технологію і обґрунтувати фінансові витрати, які можуть виникнути.

В рамках цієї роботи ми дослідили інструменти оцінки ефективності розгортання програмного забезпечення в залежності від особливостей та вимог різних класів інформаційних систем. Були проаналізовані існуючі рішення, знайдені їх недоліки, запропонований новий програмний сервіс.

На основі розглянутих питань та проведеного дослідження можна зробити наступні висновки: AWS Lambda є потужним інструментом для розробки та виконання функцій, що можуть використовуватися для розв'язання різноманітних задач; з використанням метрик середовища виконання можна відстежувати різноманітні показники ефективності функцій та забезпечити оптимальну їх роботу; для забезпечення оптимального функціонування AWS Lambda необхідно розробляти ефективний код та оптимізувати параметри середовища виконання, такі як розмір пам'яті та час виконання; налагодження та підтримка AWS Lambda можуть вимагати великих зусиль, тому необхідно відвести достатньо часу для вивчення інструменту та розробки ефективного підходу до роботи з ним; за допомогою AWS CloudWatch можна відстежувати різноманітні метрики середовища виконання та забезпечити їх аналіз для покращення ефективності роботи функцій; розуміння принципів роботи та можливостей AWS Lambda

дозволяє забезпечити оптимальне використання цього інструменту та забезпечити розв'язання задач з максимальною ефективністю.

Іншим важливим висновком є те, що використання інструментів моніторингу та аналізу метрик може допомогти виявити проблеми в середовищі виконання та дозволяє проводити налагодження та оптимізацію коду, що зменшує час виконання та вартість обчислень. Крім того, збір та аналіз метрик може допомогти виявити тенденції та зміни в середовищі виконання з часом, що дає змогу приймати обґрунтовані рішення щодо планування ресурсів та масштабування сервісу.

Отже, у результаті дослідження можна зробити висновок про те, що використання моніторингу та аналізу метрик середовища виконання AWS Lambda дозволяє ефективніше управляти сервісом та підвищує продуктивність обчислень. Такий підхід є особливо корисним для проектів, де вимоги до масштабування можуть змінюватись відносно швидко, і де важливо забезпечити максимальну ефективність використання ресурсів.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Alin Calinciuc, Cristian Spoiala, Cornel Turcu. OpenStack and Docker: Building a high-performance IaaS platform for interactive social media applications [Електронний ресурс] // researchgate.net. – 2016. – Режим доступу до ресурсу: https://www.researchgate.net/publication/304021595_OpenStack_and_Docker_Building_a_high-performance_IaaS_platform_for_interactive_social_media_applications.

2. J. Beschi Raja, K. Vivek Rabinson. Iaas for Private and Public Cloud using Openstack [Електронний ресурс] // International Journal Of Engineering Research & Technology (Ijert). – 2016. – Режим доступу до ресурсу: https://www.ijert.org/iaas-for-private-and-public-cloud-using-openstack.

3. Tianju Wanga, Xiaolin Changa, Bo Liua. Performability Analysis for IaaS Cloud Data Center [Електронний ресурс] // ieeexplore.ieee.org. – 2016. – Режим доступу до ресурсу: https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/7943338.

4. H. Rabinson. Scheduled Vitrual Machine Placement in IaaS Cloud [Електронний ресурс] // ieeexplore.ieee.org. – 2018. – Режим доступу до ресурсу: https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/8985728.

5. Tianhan Gao. PAAS: PMIPv6 Access Authentication Scheme based on Identity-based Signature in VANETs [Електронний ресурс] / Tianhan Gao, Xinyang Deng, Yingbo Wang, Xiangjie Kong // ieeexplore.ieee.org. – 2018. – Режим доступу до ресурсу: https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/8367825.

6. Robert Lovas. PaaS-oriented IoT platform with Connected Cars use cases [Електронний ресурс] / Robert Lovas, Attila Csaba Marosi, Mark Emodi, Adam Kisari, Erno Simonyi, Peter Gaspar // ieeexplore.ieee.org. – 2018. – Режим доступу до ресурсу: https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/8615963.

7. Mykhailo Klymash. Increasing the Accessibility of Static Content using CDN Networks as PaaS [Електронний ресурс] / Mykhailo Klymash, Olha Shpur, Orest Lavriv, Nazar Peleh // researchgate.net. – 2019. – Режим доступу до ресурсу:

https://www.researchgate.net/publication/334768306_Increasing_the_Accessibility_of_Static_Content_using_CDN_Networks_as_PaaS.

8. Chen Zhong, Xin Yuan. Intelligent Elastic Scheduling Algorithms for PaaS Cloud Platform Based on Load Prediction [Електронний ресурс] // ieeexplore.ieee.org. – 2019. – Режим доступу до ресурсу: https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/8785600.

9. Zepeng Wen, Yan Liang, Gongliang Li. Design and Implementation of High-availability PaaS Platform Based on Virtualization Platform [Електронний ресурс] // ieeexplore.ieee.org. – 2020. – Режим доступу до ресурсу: https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/9141564.

10. Jing Zhong, Wei Liu. Research on Container Security of PaaS [Електронний ресурс] // ieeexplore.ieee.org. – 2020. – Режим доступу до ресурсу: https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/9202098.

11. Meng-Jun Hsu, Shih-Wei Chou, Hui-Tzu Min. Understanding Software-as-a-Service (SaaS) Commitment from a Client-Provider Collaboration Approach [Електронний ресурс] // researchgate.net. – 2015. – Режим доступу до ресурсу: https://www.researchgate.net/publication/290391842_Understanding_Software-as-a-Service_SaaS_Commitment_from_a_Client-Provider_Collaboration_Approach