

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
РАДІОЕЛЕКТРОНІКИ

Практична робота №2

з дисципліни

«Архітектура програмного забезпечення»

на тему: «Архітектура відомих програмних системи. Amazon»

Виконав:

ст. гр. ПЗП-22-5

Коноваленко Даніїл Миколайович

Перевірив:

ст. викладач каф. ПІ

Сокорчук Ігор Петрович

Харків 2024

**1 ІСТОРІЯ ЗМІН**

<b>Ім'я</b>	<b>Дата</b>	<b>Причина для змін</b>	<b>Версія</b>
Коноваленко Даніїл	05.04.2025	Початкова версія	1.0

## 2 ЗАВДАННЯ

1. Підготувати доповідь на тему: «Архітектура відомих програмних систем»
2. Створити та оформити слайди презентації доповіді
3. Створити та опублікувати на YouTube відеозапис доповіді
4. Оформити та завантажити на [dl.nure.ua](https://dl.nure.ua) звіт до виконаної роботи
5. При потребі створити та налаштувати у GitHub обліковий запис для облікового запису студента в домені [@nure.ua](mailto:@nure.ua)
6. Створити для виконання завдань з дисципліни GitHub репозиторій із вказаним на <https://dl.nure.ua> іменем та з головною гілкою `main`
7. Створити у корені створеного репозиторію файл `README.md` з вказаним далі змістом та вказані далі директорії для окремих завдань
8. Експортувати звіт у файл у простому текстовому форматі та завантажити цей файл у директорію **Pract2** у репозиторії GitHub
9. Завантажити у вказану далі піддиректорію увесь програмний код, який розглядається у доповіді ПЗ2

### 3 ОПИС ВИКОНАНОЇ РОБОТИ

Під час підготовки доповіді-презентації виконано комплексний аналіз еволюції архітектури Amazon, зосереджений на переході до мікросервісної архітектури та впровадженні сучасних технологічних рішень. Робота виконана у такій послідовності:

1. Аналіз переходу до мікросервісної архітектури  
Досліджено етап переходу Amazon від монолітної архітектури до мікросервісної, який розпочався у 2010 році. Встановлено, що цей перехід дозволив значно підвищити швидкість розроблення та впровадження змін, досягнувши частоти оновлень кожні кілька секунд. Мікросервісна архітектура подолала обмеження традиційних рішень, забезпечивши більшу гнучкість і масштабовість системи, що дало змогу Amazon адаптуватися до зростаючих вимог користувачів.
2. Огляд принципів мікросервісної архітектури  
Визначено ключові принципи мікросервісної архітектури, що застосовуються в Amazon:
  - кожен сервіс виконує окрему специфічну задачу, що спрощує розроблення, тестування та підтримку;
  - незалежність і автономність компонентів, що дозволяє розгортати та оновлювати сервіси незалежно;
  - комунікація між сервісами через чітко визначені API-інтерфейси, що забезпечує прозору та ефективну взаємодію.
3. Аналіз технологічних компонентів  
Розглянуто основні технологічні компоненти, які використовуються в Amazon для підтримки мікросервісів:

- Amazon EC2 – забезпечує обчислювальні потужності та масштабовану інфраструктуру для запуску сервісів;
- Amazon S3 – використовується для надійного та економічного зберігання великих обсягів даних із високою доступністю;
- Amazon API Gateway – забезпечує управління API, спрощує розроблення та підтримує безпечний доступ до сервісів.

4. Дослідження комунікації між сервісами  
Проаналізовано механізми комунікації між сервісами в Amazon. Встановлено, що використовується асинхронна взаємодія, яка дозволяє ефективно обробляти запити та забезпечує гнучкість у масштабуванні.

5. Огляд інструментів моніторингу  
Досліджено інструменти моніторингу, що застосовуються в Amazon:

- Amazon CloudWatch – для збору та аналізу метрик, логів і подій, що забезпечує видимість стану системи;
- розподілене трасування запитів для відстеження їхнього шляху через різні сервіси;
- автоматизація реакції на збої, що дозволяє системі відновлюватися без втручання людини;
- логування та аналіз продуктивності для виявлення вузьких місць і оптимізації роботи системи.

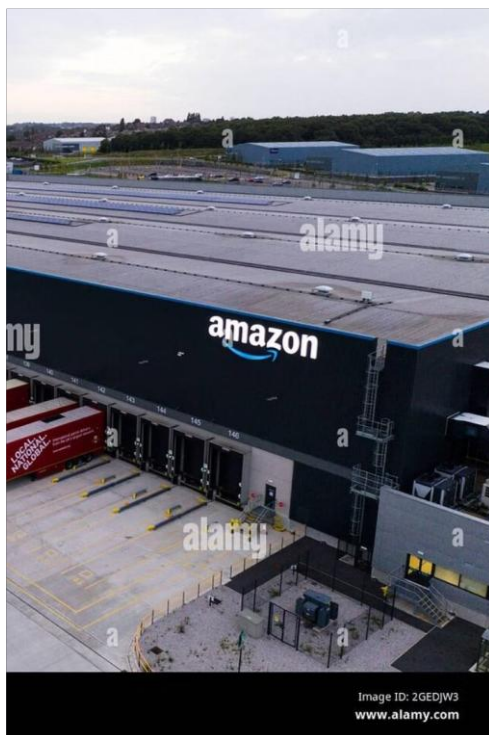
6. Оцінка безпеки та масштабованості  
Визначено, що ізоляція сервісів у мікросервісній архітектурі зменшує ризик поширення збоїв і атак. Також підкреслено важливість автоматичного відновлення після збоїв, що підвищує надійність системи.

## 4 ВИСНОВКИ

Еволюція архітектури Amazon від монолітної до мікросервісної стала важливим кроком у підвищенні ефективності розроблення та масштабованості системи. Перехід дозволив досягти високої частоти оновлень і подолати обмеження традиційних архітектурних рішень. Мікросервісна архітектура, побудована на принципах автономності, ізоляції та чітко визначених API-інтерфейсів, забезпечує гнучкість і надійність системи. Використання сучасних технологічних компонентів, таких як Amazon EC2, S3 та API Gateway, підтримує масштабовану інфраструктуру та безпечний доступ до сервісів. Інструменти моніторингу, зокрема Amazon CloudWatch і розподілене трасування, дозволяють оперативно виявляти та усувати проблеми, а автоматизація реакції на збої підвищує стійкість системи.

## Додаток А

## Додаток Б



## Архітектура програмних систем Amazon

Amazon є піонером у розробці складних програмних систем, використовуючи мікросервісну архітектуру світового рівня. Головна сторінка Amazon.com використовує понад 200 мікросервісів. Це є революційним підходом до побудови складних програмних систем, що дозволяє швидко та ефективно реагувати на потреби ринку та користувачів.

**Автор:** Даніїл Коноваленко  
**Група:** ПЗПН22-5

Рисунок Б.1 – Вступний слайд

## Еволюція Архітектури Amazon

### Перехід до мікросервісів

У 2010 році Amazon здійснила перехід від монолітної до мікросервісної архітектури. Цей перехід дозволив підвищити швидкість розробки та впровадження змін. Частота оновлень досягає однієї зміни кожні кілька секунд.

### Подолання обмежень

Мікросервісна архітектура дозволила подолати обмеження традиційних архітектурних рішень. Це забезпечило більшу гнучкість та масштабованість системи. Amazon змогла адаптуватися до зростаючих вимог користувачів.

Рисунок Б.2 – Еволюція архітектури Amazon



## Принципи Мікросервісної Архітектури



### Окрема задача

Кожен сервіс вирішує окрему специфічну задачу. Це спрощує розробку, тестування та підтримку компонентів системи.



### Незалежність компонентів

Незалежність та автономність компонентів. Кожен сервіс може бути розгорнутий та оновлений незалежно від інших.



### Чіткі API

Комунікація через чітко визначені API-інтерфейси. Це забезпечує прозору та ефективну взаємодію між сервісами.



3

Рисунок Б.3 – Принципи мікросервісної архітектури

## Технологічні Компоненти



### Amazon EC2

Для обчислювальних потужностей. Забезпечує масштабовану та надійну інфраструктуру для запуску сервісів.



### Amazon S3

Для зберігання даних. Надійне та економічне зберігання великих обсягів даних. Забезпечує високу доступність та захист даних.



### Amazon API Gateway

Для управління API. Спрощує розробку та підтримку API. Забезпечує безпечний доступ до сервісів.



### Docker

Контейнеризація з використанням Docker. Спрощує розгортання та управління сервісами. Забезпечує ізоляцію та відтворюваність середовища.



4

Рисунок Б.4 – Технологічні компоненти



## Організаційна Структура Розробки

### "Ти створюєш - ти підтримуєш"

Принцип "ти створюєш - ти підтримуєш". Команда розробників несе повну відповідальність за свій сервіс протягом усього його життєвого циклу.

### Малі команди

Малі команди (до 10 осіб). Це дозволяє забезпечити більшу ефективність та швидкість розробки. Команди мають автономію у прийнятті рішень.

### Повна відповідальність

Повна відповідальність за свій мікросервіс. Команда контролює всі аспекти розробки, розгортання та підтримки сервісу.

5

Рисунок Б.5 – Організаційна структура розробки

## Комунікація Між Сервісами



### Асинхронна взаємодія

Сервіси взаємодіють асинхронно. Це дозволяє зменшити залежність між компонентами та підвищити відмовостійкість системи.



### Черги повідомлень

Використання черг повідомлень (наприклад, Amazon SQS). Це забезпечує надійну доставку повідомлень між сервісами.



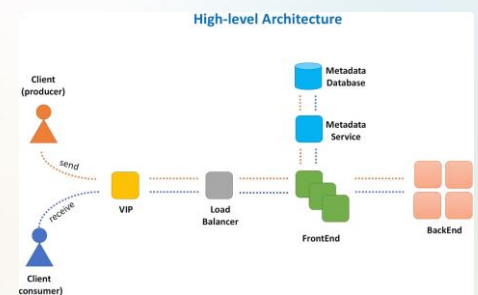
### Горизонтальна масштабованість

Горизонтальна масштабованість. Сервіси можуть бути легко масштабовані для обробки зростаючого навантаження.



### Відмовостійкість

Відмовостійкість архітектури. Система може продовжувати працювати навіть у випадку збоїв окремих компонентів.



6

Рисунок Б.6 – Комунікація між сервісами



## Інструменти Моніторингу



### Amazon CloudWatch

Для моніторингу інфраструктури. Збирає та аналізує метрики, логи та події для забезпечення видимості стану системи.



### Розподілене трасування

Розподілене трасування запитів. Дозволяє відстежувати шлях запиту через різні сервіси для виявлення проблемних місць.



### Автоматизація реакції

Автоматизація реакції на збої. Система може автоматично відновлюватися після збоїв без втручання людини.



### Логування та аналіз

Логування та аналіз продуктивності. Дозволяє виявляти вузькі місця та оптимізувати роботу системи.

7

Рисунок Б.7 – Інструменти моніторингу

## Безпека та Масштабованість

### Ізоляція сервісів



Сервіси ізольовані один від одного. Це зменшує ризик поширення збоїв та атак.

### Автоматичне відновлення



Автоматичне відновлення після збоїв. Система може автоматично відновлюватися після збоїв без втручання людини.



### Динамічне масштабування

Динамічне масштабування ресурсів. Система може автоматично збільшувати або зменшувати обсяг обчислювальних ресурсів в залежності від навантаження.




### Багаторівневий захист

Багаторівневий захист даних. Використовуються різні методи захисту даних на різних рівнях системи.

8

Рисунок Б.8 – Безпека та масштабованість




## Переваги Архітектури

- Швидкість розробки**  
Швидкість розробки та впровадження змін. Мікросервісна архітектура дозволяє швидко розробляти та впроваджувати нові функції
- Незалежне масштабування**  
Незалежне масштабування компонентів. Сервіси можуть бути масштабовані незалежно один від одного.
- Висока відмовостійкість**  
Висока відмовостійкість системи. Система може продовжувати працювати навіть у випадку збоїв окремих компонентів
- Спрощена підтримка**  
Спрощена підтримка та оновлення. Кожен сервіс може бути оновлений незалежно від інших.

9

Рисунок Б.9 – Переваги архітектури



## Майбутнє Архітектури

**90%**

**Контейнеризація**

Подальша контейнеризація. Більшість сервісів буде розгорнуто у контейнерах для спрощення управління та розгортання.

**80%**

**Serverless**

Розвиток serverless-технологій. Використання serverless-функцій для вирішення окремих задач.

**70%**

**Штучний інтелект**

Штучний інтелект в управлінні інфраструктурою. Автоматизація процесів моніторингу, масштабування та відновлення.

Постійна оптимізація архітектурних рішень. Amazon продовжує впроваджувати нові технології та підходи для забезпечення максимальної ефективності та надійності своєї архітектури.

10

Рисунок Б.10 – Майбутнє архітектури. Висновки

## Додаток В

## Приклад В.1

Приклад асинхронної комунікації між сервісами (Python з використанням asyncio):

```
01 import asyncio
02 import aiohttp
03
04 async def fetch_data(url):
05     async with aiohttp.ClientSession() as session:
06         async with session.get(url) as response:
07             return await response.json()
08
09 async def main():
10     url = "https://api.example.com/data"
11     data = await fetch_data(url)
12     print(data)
13
14 asyncio.run(main())
```

## Приклад В.2

Приклад використання Amazon API Gateway для виклику сервісу (Python з boto3):

```
01 import boto3
02
03 client = boto3.client('apigateway')
04
05 def invoke_api(api_id, stage, path):
06     response = client.test_invoke_method(
07         restApiId=api_id,
08         resourceId='resource_id',
09         httpMethod='GET',
10         pathWithQueryString=path,
11         stageName=stage
12     )
13     return response['body']
14
15 api_response = invoke_api('api_id', 'prod', '/data')
16 print(api_response)
```