ПРИКЛАДНІ ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК,
БЕЗПЕКИ ТА МАТЕМАТИКИ, 2024, № 3

УДК 004.02

ВИКОРИСТАННЯ SERVERLESS МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ ДЛЯ ВІДПРАВКИ ДАНИХ НА SFTP СЕРВЕР

Аверічев І. М. (ORCID: 0009-0008-9766-0115)
Заярнюк М. О. (ORCID 0009-0009-6675-2786)
Гаманюк І.М. (0009-0003-3904-7578)
Державний університет інформаційно-комунікаційних технологій, Київ

UTILIZING MICROSERVICES ARCHITECTURE BASED ON AWS FOR DATA TRANSFER TO AN FTP SERVER

Ihor Averichev, Maksym Zaiarniuk, Ihor Gamaniuk
State University of Information and Communication Technologies, Kyiv

Abstract. This article explores the use of serverless microservices architecture for automated data transfer to an SFTP server. The core component of the solution is an AWS Lambda function written in C#, which extracts data from a relational database, converts it to a CSV file, and sends it to an FTP server. The Lambda function ensures a continuous data flow with minimal delays, which is critical for many business processes. The article describes the architecture of the solution, integration with other system components, the development process of the Lambda function, and its optimization. The proposed solution achieves high performance, flexibility, and system reliability.
Keywords: serverless architecture, microservices, AWS Lambda, C#, SFTP server, automation, data processing.

Вступ. Мікросервісна архітектура стала невід'ємною частиною сучасних програмних систем завдяки своїм численним перевагам, таким як висока масштабованість, гнучкість та легкість в управлінні. Вона надає широкий спектр інструментів для розгортання та управління мікросервісами, дозволяючи розробникам створювати ефективні та надійні рішення.

Велика чисельність сервісів, які пропонуються для розробки мікросервісної архітектури, покриває більшість потреб, що виникають при розробці сучасного програмного забезпечення. Такі сервіси, як безсерверні обчислення (Serverless Computing), контейнеризація, та системи моніторингу допомагають ефективно вирішувати проблеми та виклики, що стоять перед розробниками програмного забезпечення. Ці сервіси забезпечують автоматичне масштабування, моніторинг та управління ресурсами, що робить їх ідеальним вибором для розробки високонавантажених систем.

У контексті задач, де фігурує передача даних, особливо у високонавантажених системах, гнучкість, простота, швидкість розробки та ціна за використання стають вирішальними факторами при виборі платформи для побудови системи, що вирішує такі задачі. Безсерверні обчислення дозволяють легко інтегрувати різні сервіси для забезпечення безперервного потоку даних з мінімальними затримками, що є критично важливим для багатьох бізнес-процесів.

Дане дослідження пропонує огляд подібних платформ для хмарних обчислень, а також реалізацію власного програмного продукту з використанням безсерверної мікросервісної архітектури для передачі даних на SFTP сервер з використанням протоколу SFTP(Secure File Transfer Protocol). Основною частиною цього рішення є Lambda функція, написана на мові програмування C#, яка виконує витяг даних з реляційної бази даних, конвертує їх у CSV файл та відправляє на SFTP сервер. У статті детально описано архітектуру рішення, яка включає різні компоненти для зберігання та обробки даних, а також процес розробки Lambda функції. Особлива увага приділена інтеграції цієї функції з іншими компонентами системи, такими як реляційна база даних для зберігання даних та тимчасове зберігання CSV файлів перед відправкою на SFTP сервер.

Аналіз останніх досліджень і публікацій. Три найбільші платформи для хмарних обчислень – AWS (Amazon Web Services), Microsoft Azure та Google Cloud Platform (GCP) – пропонують широкий спектр послуг та інструментів для розгортання та управління додатками.[1] Розглянемо основні відмінності між цими платформами, а також їхні переваги та недоліки.

AWS (Amazon Web Services) пропонує найбільший набір сервісів серед усіх хмарних провайдерів, включаючи обчислення, зберігання, бази даних, аналітику, мережі, мобільні сервіси, розробницькі інструменти, інструменти управління, IoT, безпеку та корпоративні додатки. AWS є найстарішим гравцем на ринку хмарних обчислень, що надає йому перевагу в плані зрілості та стабільності сервісів. Крім того, AWS має найбільшу кількість регіонів та зон доступності, що забезпечує високу доступність та низькі затримки для користувачів по всьому світу[2]. Однак широкий спектр сервісів може бути складним для розуміння та управління, особливо для нових користувачів. AWS часто критикують за високі ціни, особливо для невеликих компаній та стартапів[2].

Microsoft Azure відрізняється відмінною інтеграцією з продуктами Microsoft, такими як Windows Server, Active Directory та SQL Server[2], що робить його привабливим вибором для компаній, які вже використовують ці продукти. Azure активно розвиває гібридні рішення, що дозволяють інтегрувати локальні та хмарні ресурси, надаючи більше гнучкості для користувачів. Крім того, Azure має велику кількість дата-центрів по всьому світу, що забезпечує високу доступність та низькі затримки. Проте, як і AWS, Azure може бути дорогим, особливо для невеликих компаній. Інтеграція та налаштування сервісів Azure можуть бути складними для нових користувачів, особливо для тих, хто не знайомий з продуктами Microsoft [3].

Google Cloud Platform (GCP) пропонує інноваційні сервіси в галузі аналітики, машинного навчання та штучного інтелекту, такі як BigQuery, TensorFlow та Google AI. GCP відомий своєю високою продуктивністю та швидкістю, що досягається завдяки потужній інфраструктурі Google. Крім того, GCP пропонує конкурентні ціни та гнучку цінову політику, включаючи знижки за тривале використання та миттєве використання (preemptible VMs). Однак, порівняно з AWS та Azure, GCP пропонує менший набір сервісів, що може обмежувати вибір для деяких користувачів. GCP є відносно новим гравцем на ринку хмарних обчислень, що може вплинути на зрілість та стабільність деяких сервісів[4].

За даними, що наведені у [5] ринкові частки основних хмарних провайдерів на 2024 рік за даними Statista. Згідно з цими даними, AWS займає 33% ринку, Microsoft Azure – 22%, Google Cloud Platform - 10%, а інші провайдери – 35% (рис.1).

Кожна з трьох основних платформ хмарних обчислень – AWS, Microsoft Azure та Google Cloud Platform – має свої унікальні переваги та недоліки. Вибір платформи залежить від конкретних потреб та вимог бізнесу. AWS пропонує найширший спектр сервісів та найвищу зрілість, Azure забезпечує відмінну інтеграцію з продуктами Microsoft та гібридні рішення, а GCP надає користувачу інноваційні сервіси та високу продуктивність.

Мета і задачі дослідження. Метою даної роботи є розгляд, аналіз та впровадження безсерверної мікросервісної архітектури для розробки автоматизованої відправки даних на FTP сервер за допомогою платформи для хмарних обчислень. Дослідження має на меті показати ефективність використання безсерверної архітектури для обробки та передачі даних, а також забезпечити високу продуктивність, гнучкість та надійність системи. Для досягнення мети поставлено такі завдання:

- проаналізувати існуючі платформи хмарних обчислень;
- оцінити можливості кожної платформи в контексті підтримки безсерверної архітектури та мікросервісів;

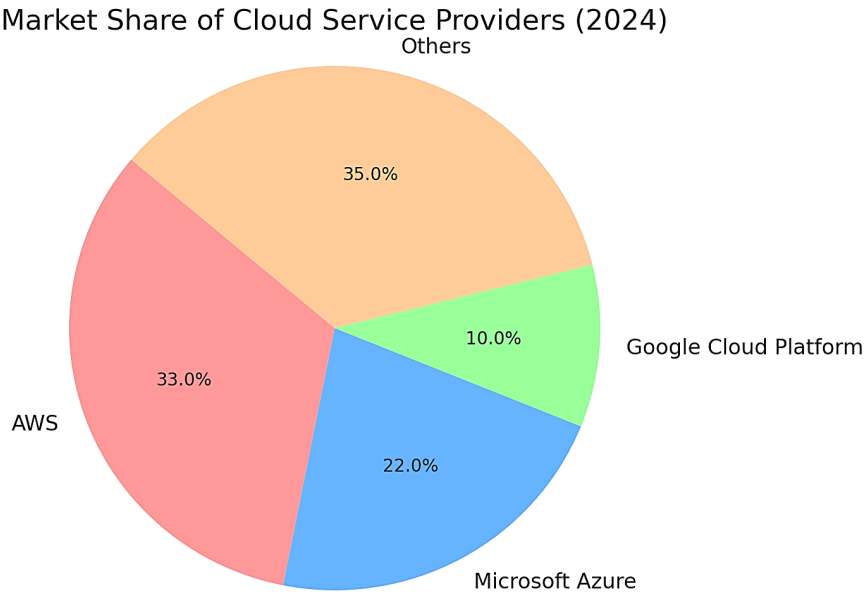

Рис. 1. Ринкові частки основних хмарних провайдерів на 2024 рік.

- визначити переваги та недоліки кожної платформи в контексті реалізації безсерверних мікросервісів;
- на основі проведеного аналізу та порівняння обрати платформу, яка забезпечить оптимальну підтримку безсерверної архітектури та мікросервісів;
- врахувати фактори, такі як доступність сервісів, вартість, легкість інтеграції та можливості масштабування;
- спроектувати безсерверну мікросервісну архітектуру, яка включає компоненти для обробки та передачі даних;
- визначити основні функціональні блоки системи та їх взаємодію;
- розробити AWS Lambda функцію на мові програмування C#, яка витягує дані з реляційної бази даних, конвертує їх у CSV файл та відправляє на SFTP сервер;
- впровадити логування та моніторинг для відстеження роботи системи та виявлення можливих проблем.

Результати дослідження. Відповідно до тенденцій використання хмарних провайдерів, було обрано AWS в якості платформи хмарних обчислень. Адже попри усі складнощі, що можуть виникнути під час роботи з цим хмарним провайдером, кількість ресурсів на яких є відповіді на потенційні запитання та проблеми є достатньо великою, що спрощує і пришвидшує процес розробки. Також, AWS має хорошу інтеграцію з різними технологіями, що не обмежує розробку, на відміну від, напркилад, Microsoft Azure, та пропонує широкий вибір сервісів, на відміну від GCP[1].

Було спроектовано архітектуру додатку. На рисунку 2 зображено архітектуру додатку. Застосунок складається з AWS Lambda компоненту, що називається FTP Publisher Lambda-частини де була описана логіка отримання даних, їх конвертація та отримання даних для отримання доступу до серверу та бази даних. Розробка була проведена за допомогою використання мови програмування C#, але через гнучкість Lambda сервісу подібний додаток можна реалізувати за допомогою будь-якої мови програмування. Також на схемі зображені: Aurora DB – реляційна база даних з якої будуть підтягуватись дані для запису на SFTP сервер; Secrets Manager – компонент який відповідає за зберігання даних з інформацією про облікові дані для доступу до SFTP серверу та бази даних.

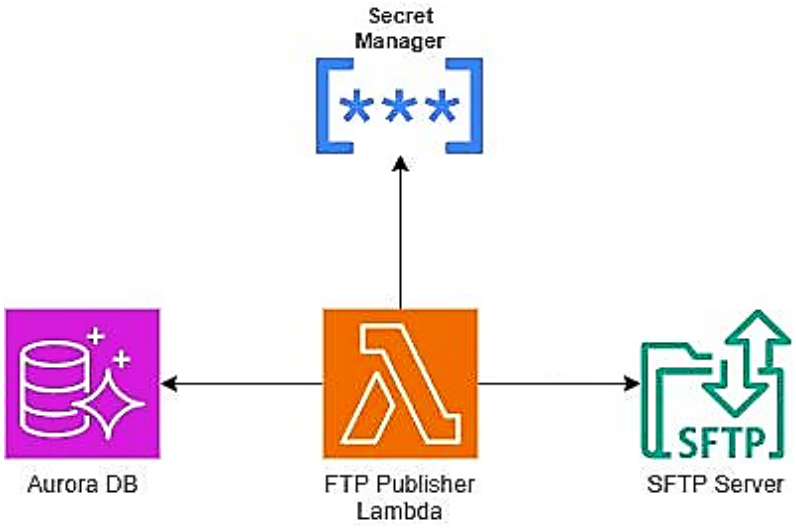

Рис. 2. Архітектура додатку.

Додаток взаємодіє між компонентами наступним чином: спочатку відбувається виклик лямбда функції на AWS, після чого, всередині цієї функції відбувається запит на стягування актуальних даних з бази даних Aurora DB, для того, щоб мати змогу під'єднатись, перед цим кроком відбувається запит до AWS Secrets Manager, з якого ми отримуємо облікові дані для доступу до бази даних, далі, отримані дані конвертуються у потрібний формат, для програмного застосунку що розглядається в даній роботі, дані конвертуються у .csv формат, після чого виконується ще один запит до AWS Secrets Manager з якого вже беруться облікові дані для доступу до SFTP серверу, після чого відбувається відправка даних з виористанням протоколу SFTP беспосередньо на сервер.

Важливо зазначити, що вибір протоколу передачі даних не грає особливого значення в даному контексті, SFTP – це альтернатива до FTP(File Transfer Protocol), яка додає додатковий рівень захищеності при файлів[6].

Оскільки мовою розробки додатку була обрана мова програмування C#, розробка була проведена в ООП стилі.

Точкою входу в додаток став метод ActualFunctionHandlerAsync, що зображений на рисунку 3.

В цьому методі розпочинається сесія логування даних викликом методу LogSessionStart на 40 стрічці. Цей метод забезпечує збереження інформації про запит, включаючи ідентифікатор сесії та кореляційний ідентифікатор для відстеження запитів. На 43 стрічці відбувається виклик основного методу роботи додатку, HandleCSVAsync, результат роботи якого записується у змінну result. Ця змінна має значення true, якщо процес передачі даних пройшов успішно, і false, якщо щось пішло не так і процес завершився невдачею. Для забезпечення стійкості додатку та обробки можливих помилок, весь блок коду обгорнуто у конструкцію try-catch. У випадку виникнення виключення, яке не є RetryNeededException, воно буде оброблено та залоговано за допомогою методу ExceptionLogger.Log. Це дозволяє зберігати інформацію про помилки та коректно завершувати роботу додатку, що підвищує надійність системи. Такий підхід до розробки забезпечує високу продуктивність та надійність Lambda функції, що є критично важливим для безперервного та стабільного обміну даними між системами.

Основна логіка додатку описана в методі HandleCSVAsync. Цей метод зображений на рисунку 4.

```
36        private async Task<bool> FunctionHandlerAsync(ILambdaContext context)
37        {
38            try
39            {
40                LogSession.Start(ServiceConstants.ServiceName, _version, _env, _region, correlationId: context.AwsRequestId,
41                    sessionId: context.AwsRequestId);
42
43                var result:bool = await _csvExportMessageHandler.HandleCSVAsync();
44
45                return result;
46            }
47            catch (Exception ex) when (ex is not RetryNeededException)
48            {
49                ExceptionLogger.Log(ex);
50            }
51            return false;
52        }
53    }
```

Рис. 3. Метод ActualFunctionHandlerAsync.

```
22        public async Task<bool> HandleCSVAsync()
23        {
24            var dateTime = DateTime.UtcNow.ToString( format: "yyyyMMddTHHmmss");
25
26            try
27            {
28                logger.LogInformation( message: "{Handler}: Starting CSV export process.", params args: nameof(CsvExportMessageHandler));
29
30                logger.LogInformation( message: "{Handler}: Starting getting customers from the database.", params args: nameof(CsvExportMessageHandler));
31                var (getCustomersFailed:bool, customers:AuroraDbCustomerEntity[], getCustomersFailure) = await ftpRepository.GetAllCustomersAsync();
32
33                if ( getCustomersFailed)
34                {
35                    logger.LogError( message: "{Handler}: There is problem when importing data from the database.", params args: nameof(CsvExportMessageHandler));
36                    logger.LogError( message: $"{nameof(CsvExportMessageHandler)}: {getCustomersFailure.Message}");
37
38                    return false;
39                }
40
41                var mappedCustomers:IEnumerable<CustomerDto> = customers.MapToContactCustomers(logger);
42                var customerCsvFileData:byte[] = await GenerateCsvFileFromDtoListAsync(mappedCustomers);
43                var customerCsvFileName = $"accounts_{dateTime}.csv";
44                await UploadCsvFileAsync(customerCsvFileData, customerCsvFileName);
45
46                logger.LogInformation( message: "{Handler}: CSV export process completed successfully.", params args: nameof(CsvExportMessageHandler));
47                return true;
48            }
49            catch (Exception ex)
50            {
51                logger.LogError(ex, message: "{Handler}: Error occurred during CSV export process.", params args: nameof(CsvExportMessageHandler));
52                return false;
53            }
54        }
```

Рис 4. Метод HandleCSVAsync.

Метод HandleCSVAsync виконує основну роботу з обробки та передачі даних. На початку методу встановлюється поточний час у змінну dateTime. Далі, за допомогою try-catch блоку, здійснюється логування початку процесу експорту даних та отримання клієнтів з бази даних за допомогою методу GetAllCustomersAsync, опис цього методу зображений на рисунку 5.

```
2 usages  Maksym Zaiarniuk
public class FTPRepository
{
    private readonly IDbContextFactory<AuroraDbContext> _contextFactory;

    Maksym Zaiarniuk
    public FTPRepository(IDbContextFactory<AuroraDbContext> contextFactory)
    {
        _contextFactory = contextFactory;
    }

    1 usage  Maksym Zaiarniuk
    public async Task<IFailableResult<AuroraDbCustomerEntity[], AuroraDbFailure>> GetAllCustomersAsync()
    {
        await using var context = await _contextFactory.CreateDbContextAsync();

        return await context.Set<AuroraDbCustomerEntity>().ToArrayAsync().TryHandleAsync();
    }

```

Рис 5. Клас FTPRepository.

Якщо виникає помилка при отриманні даних з бази, метод LogError залоговує повідомлення про помилку, і метод повертає false. Якщо дані успішно отримані, вони конвертуються у формат DTO (Data Transfer Object) за допомогою методу MapToContactCustomers.

Наступним кроком є генерація CSV файлу з отриманих даних за допомогою методу GenerateCsvFileFromDtoListAsync, що зображений на рисунку 6.

```
1 usage  Maksym Zaiarniuk +2
private async Task<byte[]> GenerateCsvFileFromDtoListAsync<T>(IEnumerable<T> data)
{
    using var memoryStream = new MemoryStream();
    await using (var streamWriter = new StreamWriter(memoryStream, Encoding.UTF8))
    await using (var csvWriter = new CsvWriter(streamWriter, CultureInfo.InvariantCulture))
    {
        csvWriter.Context.TypeConverterOptionsCache.GetOptions<DateTime?>().Formats = new[] { "yyyy-MM-dd HH:mm:ss.fff" };
        await csvWriter.WriteRecordsAsync(data);
    }
    return memoryStream.ToArray();
}

```

Рис 6. Метод GenerateCsvFileFromDtoListAsync.

Потім файл завантажується на FTP сервер за допомогою методу UploadCsvFileAsync, код класу що містить в собі цей метод зображений на рисунку 7. Якщо процес пройшов успішно, логування повідомляє про успішне завершення процесу експорту даних, і метод повертає true. У випадку виникнення будь-якої помилки, вона буде залогована, і метод поверне false, що забезпечує стійкість системи та можливість відстеження помилок.

```
1 usage   Maksym Zaiarniuk +2
private async Task UploadCsvFileAsync(byte[] csvData, string fileName)
{
    var ftpSettings = await ftpProvider.GetFtpSettingsFromSecretsManagerAsync();
    var path = $"Import/MasterData/{fileName}";
    using SftpClient client = new(ftpSettings.FtpHost, port:Convert.ToInt32(ftpSettings.FtpPort), ftpSettings.FtpUsername, ftpSettings.FtpPassword);
    try
    {
        client.Connect();
        if (client.IsConnected)
        {
            var stream = new MemoryStream(csvData);
            client.UploadFile(stream, path);
            client.Disconnect();
        }
    }
    catch (Exception e) when (e is SshConnectionException || e is SocketException || e is ProxyException)
    {
        Console.WriteLine($"Error connecting to server: {e.Message}");
    }
    catch (SshAuthenticationException e)
    {
        Console.WriteLine($"Failed to authenticate: {e.Message}");
    }
    catch (SftpPermissionDeniedException e)
    {
        Console.WriteLine($"Operation denied by the server: {e.Message}");
    }
    catch (SshException e)
    {
        Console.WriteLine($"Sftp Error: {e.Message}");
    }
}
```

Рис 7. Метод UploadCsvFileAsync.

Клас FTPRepository відповідає за отримання даних з бази даних. У конструкторі класу ініціалізується фабрика контексту _contextFactory, яка використовується для створення екземплярів контексту бази даних.

Метод GetAllCustomersAsync асинхронно отримує всі записи клієнтів з бази даних. За допомогою фабрики створюється контекст бази даних, і потім дані витягуються у вигляді масиву об'єктів AuroraDbCustomerEntity.

Якщо процес отримання даних пройшов успішно, метод повертає результат, інакше обробляє помилку. Цей підхід забезпечує високу продуктивність та надійність Lambda функції, що є критично важливим для безперервного та стабільного обміну даними між системами.

Метод GenerateCsvFileFromDtoListAsync призначений для конвертації списку об'єктів у формат CSV та повернення результату у вигляді масиву байтів. Він використовує бібліотеку CsvHelper для зручної та ефективної роботи з CSV файлами.

На початку виконання методу створюється MemoryStream, який використовується для зберігання CSV даних у пам'яті. За допомогою конструкції await using метод створює

StreamWriter, що записує дані у MemoryStream з кодуванням UTF-8, а потім CsvWriter, що використовує StreamWriter для запису даних у форматі CSV. Для обробки об'єктів типу DateTime метод налаштовує формат дати і часу через TypeConverterOptionsCache, встановлюючи формат як "yyyy-MM-dd HH:mm:ss.fff". Потім метод асинхронно записує всі об'єкти зі списку data у CSV файл за допомогою WriteRecordsAsync. Після завершення запису метод конвертує вміст MemoryStream у масив байтів та повертає його як результат. Для підвищення швидкодії використовується MemoryStream, що дозволяє уникнути затримок, пов'язаних із записом на диск.

Всі операції запису реалізовані асинхронно за допомогою await, що дозволяє ефективно використовувати ресурси системи і підвищує продуктивність, особливо при обробці великих обсягів даних.

Бібліотека CsvHelper забезпечує високопродуктивну обробку CSV файлів і спрощує процес запису, що знижує кількість можливих помилок і підвищує ефективність. Налаштування форматів для DateTime дозволяє уникнути додаткових операцій форматування при записі дат у CSV, що також сприяє підвищенню швидкодії. Використання конструкції using забезпечує автоматичне звільнення ресурсів після завершення операцій, що знижує навантаження на систему і запобігає можливим витокам пам'яті.

Метод UploadCsvFileAsync відповідає за завантаження CSV файлу на FTP сервер. На початку методу асинхронно отримуються налаштування FTP з секретного менеджера за допомогою методу GetFtpSettingsFromSecretsManagerAsync, що зображений на рисунку 8.

```
3 usages  Maksym Zaiarniuk
public class FtpSettingsProvider(ISecretsManager<FtpSettings> secretsManager, ILogger<FtpSettingsProvider> logger)
{
    1 usage  Maksym Zaiarniuk
    public async Task<FtpSettings> GetFtpSettingsFromSecretsManagerAsync()
    {
        var ftpSecretsManagerName :string  = EnvVariableHelper.GetEnvironmentVariableValue("SECRETS_MANAGER_NAME");
        ftpSecretsManagerName += "-secrets";
        logger.LogInformation( message: "Getting FTP connection credentials from {secret_manager_name} secret manager", ftpSecretsManagerName);
        return await secretsManager.GetNamedSecrets(ftpSecretsManagerName);
    }
}
```

Рис 8. Метод GetFtpSettingsFromSecretsManagerAsync.

Далі, шлях до файлу формується на основі назви файлу. Для підключення до FTP сервера використовується клас SftpClient з бібліотеки ssh.net, який ініціалізується параметрами підключення, включаючи хост, порт, ім'я користувача та пароль.

Ця бібліотека забезпечує зручний і безпечний спосіб роботи з SFTP. Метод обгорнутий у блок try-catch для обробки можливих помилок під час підключення та передачі файлу. Якщо підключення успішне, дані файлу завантажуються за допомогою потоку пам'яті (MemoryStream), і файл завантажується на сервер через метод UploadFile. Після завершення передачі з'єднання розривається викликом методу Disconnect.

Якщо виникають помилки підключення, автентифікації або дозволів, вони обробляються відповідними блоками catch, які виводять повідомлення про помилку в консоль. Це забезпечує можливість діагностики та усунення проблем, що можуть виникнути під час передачі файлу на FTP сервер.

Клас FtpSettingsProvider відповідає за отримання налаштувань SFTP з секретного менеджера. Він використовує інтерфейс ISecretsManager<FtpSettings> для взаємодії з секретним менеджером та інтерфейс ILogger<FtpSettingsProvider> для логування. Конструктор класу ініціалізує залежності secretsManager та logger. Метод

GetFtpSettingsFromSecretsManagerAsync відповідає за отримання налаштувань FTP. Він починається з отримання назви секретного менеджера з середовища за допомогою EnvVariableHelper.GetEnvironmentVariableValue, до якого додається суфікс "-secrets". Потім метод логує інформацію про отримання облікових даних для підключення до FTP через секретний менеджер. Виклик методу secretsManager.GetNamedSecrets виконує асинхронне отримання налаштувань з секретного менеджера та повертає їх.

Висновки. В результаті дослідження було розроблено ефективне рішення для автоматизованої відправки даних на FTP сервер за допомогою безсерверної мікросервісної архітектури. Використання AWS Lambda функції, написаної на мові програмування C#, дозволило забезпечити високу продуктивність та гнучкість системи. Інтеграція з іншими компонентами AWS, такими як Amazon RDS(Aurora DB) та Amazon Secrets Manager забезпечила надійність та безпеку передачі даних.

Рішення дозволяє легко масштабувати систему, забезпечуючи стабільну роботу навіть при високих навантаженнях. Важливою перевагою є також зниження витрат на обчислювальні ресурси завдяки використанню безсерверної архітектури. Оптимізація методу генерації CSV файлів та передача даних на FTP сервер з використанням бібліотеки ssh.net забезпечили високу швидкодію та надійність.

Майбутні дослідження можуть бути спрямовані на покращення розробленої системи, наприклад використання Amazon EventBridge Scheduler для автоматичного тригера функції раз на годину, що дозволить ще більше автоматизувати процеси та підвищити ефективність системи. Запропоноване рішення може бути адаптоване для різних сценаріїв передачі даних, що робить його універсальним інструментом для багатьох бізнес-процесів.

Бібліографія

1. Highlight the Features of AWS, GCP and Microsoft Azure that Have an Impact when Choosing a Cloud Service Provider. *International Journal of Recent Technology and Engineering*. 2020. Т. 8, № 5. С. 4124–4132. URL: https://doi.org/10.35940/ijrte.d8573.018520 (дата звернення: 23.06.2024).
2. Wittig A., Wittig M. Amazon Web Services in Action, Third Edition: An in-Depth Guide to AWS. Manning Publications Co. LLC, 2022.
3. Praveen Borra. Exploring Microsoft Azure's Cloud Computing: A Comprehensive Assessment. *International Journal of Advanced Research in Science, Communication and Technology*. 2022. С. 897–906. URL: https://doi.org/10.48175/ijarsct-5807c (дата звернення: 23.06.2024).
4. Geewax J. J. (. Google Cloud Platform in Action. Manning Publications Co. LLC, 2018.
5. Global cloud infrastructure market share 2024 | Statista. *Statista*. URL: https://www.statista.com/statistics/967365/worldwide-cloud-infrastructure-services-market-share-vendor/#:~:text=In%20the%20first%20quarter%20of,with%2010%20percent%20market%20share (дата звернення: 06.06.2024).
6. SFTP vs. FTP: Understanding the Difference. *Integrate.io*. URL: https://www.integrate.io/blog/sftp-vs-ftp-understanding-the-difference/ (дата звернення: 11.06.2024).

References

1. Highlight the Features of AWS, GCP and Microsoft Azure that Have an Impact when Choosing a Cloud Service Provider. *International Journal of Recent Technology and Engineering*. 2020. Vol. 8, no. 5. P. 4124–4132. URL: https://doi.org/10.35940/ijrte.d8573.018520 (date of access: 23.06.2024).
2. Wittig A., Wittig M. Amazon Web Services in Action, Third Edition: An in-Depth Guide to AWS. Manning Publications Co. LLC, 2022.
3. Praveen Borra. Exploring Microsoft Azure's Cloud Computing: A Comprehensive Assessment. *International Journal of Advanced Research in Science, Communication and Technology*. 2022. P. 897–906. URL: https://doi.org/10.48175/ijarsct-5807c (date of access: 23.06.2024).
4. Geewax J. J. (. Google Cloud Platform in Action. Manning Publications Co. LLC, 2018.
5. Global cloud infrastructure market share 2024 | Statista. *Statista*. URL: https://www.statista.com/statistics/967365/worldwide-cloud-infrastructure-services-market-share-vendor/#:~:text=In%20the%20first%20quarter%20of,with%2010%20percent%20market%20share (date of access: 06.06.2024).
6. SFTP vs. FTP: Understanding the Difference. *Integrate.io*. URL: https://www.integrate.io/blog/sftp-vs-ftp-understanding-the-difference/ (date of access: 11.06.2024).