Київський національний університет імені Тараса Шевченка Факультет інформаційних технологій
Кафедра програмних систем і технологій

На правах рукопису

УДК_004.83____

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема

РОЗРОБКА ВІРТУАЛЬНОГО ПОМІЧНИКА З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ АНАЛІЗУ AWS ІНФРАСТРУКТУРИ

Спеціальність 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент
Рибалочка М.С.

_____
(підпис) (розшифровка підпису) (дата)

Науковий керівник
Меркулова К.В.

_____
(посада) (підпис) (розшифровка підпису) (дата)

Допускається до захисту з питань нормоконтролю
Завідувач кафедри
Бичков О.С.

_____
(підпис) (розшифровка підпису) (дата)

Київ - 2022

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри програмних систем і технологій
_____(О.С.Бичков)
“____”_____ 20___р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ

СТУДЕНТУ

_____Рибалочці Михайлу Сергійовичу_____
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної магістерської роботи “Розробка віртуального__ помічника з використанням штучного інтелекту для аналізу AWS інфраструктури”

_____

затверджені наказом вищого навчального закладу від „__”_____20__р. No_____

2. Строк здачі студентом закінченої роботи _____

3. Вихідні дані до роботи навчальні посібники, статті, Інтернет-ресурси_____

_____

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити) _____

Аналітична частина:_____

- обґрунтувати актуальність розробки AWS чат-боту для аналізу статистики____ витрат;_____
- дослідити існуючі види чат-ботів для аналізу статистики витрат;_____
- визначити переваги та недоліки існуючих рішень;_____
- визначити концепції системи, яка буде створена;_____

Практична частина:_____

- спроектувати архітектуру AWS чат-боту для аналізу статистики витрат;_____
- розробити AWS чат-бот для аналізу статистики витрат;_____
- запустити проект в роботу._____

5. Перелік графічного матеріалу (з точним забезпеченням обов'язкових креслень)
діаграми хмарної інфраструктури; UML діаграми: класів, процесів, прецедентів, об'єктів, взаємодії; діаграма CI/CD процесів.

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1. Telegram чат-боти	К.В. Меркулова		
2. Хмарні сервіси AWS	К.В. Меркулова		
3.1 Бізнес вимоги	К.В. Меркулова		
3.2 Життєвий цикл системи	К.В. Меркулова		
3.3 Архітектура системи	К.В. Меркулова		
3.4 Опис CI/CD	К.В. Меркулова		
3.5 Технічне завдання	К.В. Меркулова		
3.6 Тестування та перевірка	К.В. Меркулова		
3.7 Аналіз статистики витрат клієнтів	К.В. Меркулова		

7. Дата видачі завдання _____

Керівник _____ Меркулова К.В.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Рибалочка М.С.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів роботи	Термін виконання етапів роботи	Примітка
1. Створення календарного плану робіт	8.11.2021 — 15.11.2021	Виконано
2. Написання розділу 1	1.12.2021 — 10.01.2022	Виконано
3. Написання розділу 2	17.01.2022 — 1.03.2022	Виконано
4. Написання розділу 3	2.03.2022 — 11.04.2022	Виконано
5. Написання розділу 4	11.04.2022 - 15.04.2022	Виконано
6. Виправлення помилок	2.05.2022 — 20.05.2022	Виконано

Студент – магістр _____Рибалочка М.С.__

(підпис) (розшифровка підпису)

Керівник роботи _____Меркулова К. В..__

(підпис) (розшифровка підпису)

АНОТАЦІЯ

Курсова робота: 54 с., 3 формули, 20 рис., 2 таблиці, 3 додатки, 21 джерело.

Тема: “Розробка віртуального помічника з використанням штучного інтелекту для аналізу AWS інфраструктури”.

Об’єкт дослідження: аналітика витрат користувачів.

Предмет дослідження: витрати користувачів на надані сервіси хмарою AWS.

Мета роботи: дослідження та розробка додатку з використанням технологій штучного інтелекту для повідомлень про надання використаних сервісів в публічній хмарах.

Методи дослідження: Інформаційний метод – проведено дослідження роботи та структури Telegram чат-ботів. Розробка системи – досліджувалась робота чат-боту в рамках serverless технологій.

Результати дослідження: проаналізовано стан альтернативних рішень для аналізу статистики витрат. Протестований офіційний мобільний додаток від AWS. Створено технічне завдання до системи. Розроблено віртуального помічника на основі Telegram чат-боту для аналізу статистики витрат користувачів. Реалізовано аналітичний модуль для виявлення аномального використання сервісів.

Висновок: Розроблено додаток для надання повідомлень статистики витрат, описано практичне обґрунтування проекту, додано аналітичний модуль для виявлення аномалій. Розроблений чат-бот представляє з себе сукупність:

- клієнтська частина у вигляді месенджеру Telegram, Slack;
- логіка обробки запитів розроблена за технологією безсерверних обчислень;
- взаємодія чат-боту і Telegram за допомогою REST API викликів.

Ключові слова. Месенджери, аналітика, аналітичні показники, прогнозування, архітектура, хмарна інфраструктура, хмарні обчислення.

ANNOTATION

Course work: 54 pages, 3 formulas, 20 pictures, 2 tables, 3 appendices, 21 sources.

Topic: "Development of a virtual assistant with using artificial intelligence to analyse AWS infrastructure."

Object of research: user cost analysis.

Subject of research: the cost of users for services provided by the AWS cloud.

Purpose: research and development of an application using artificial intelligence technologies for reports on the provision of used services in public clouds.

Research methods: Information method — research of work and structure of Telegram chatbots. System development — the work of a chatbot in the framework of serverless technologies was studied.

Research results: the state of alternative solutions for the analysis of cost statistics is analysed. Tested official mobile application from AWS. A technical task for the system has been created. Developed a virtual assistant based on the Telegram chatbot to analyse user cost statistics. An analytical module for detecting abnormal use of services has been implemented.

Conclusion: An application for providing cost statistics reports has been developed, a practical justification of the project has been described, and an analytical module for detecting anomalies has been added. Developed chatbot is a set of:

- client part in the form of messenger Telegram, Slack;
- request processing logic is developed by serverless computing technology;
- interaction of chatbot and Telegram using REST API calls.

Keywords. Messengers, analytics, analytical indicators, forecasting, architecture, cloud infrastructure, cloud computing.

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ

AWS	Amazon Web Services — сервіс з надання послуг публічної хмари.
SAM	Serverless Application Model — феймворк розроблений AWS для покращення роботи з serverless технологіями.
EC2	Elastic Compute Cloud — сервіс AWS з надання обчислювальних ресурсів — віртуальних машин.
RDS	Relation Database Service — сервіс AWS з надання послуг хостингу та адміністрування реляційних баз даних.
CI/CD	Continuous Integration and Continuous Deployment — підхід до розробки з автоматизацією однорідних процесів.
API	Application Programming Interface — інтерфейс сторонніх додатків для розробників, який допомагає інтегрувати або розробляти додатки на основі вже існуючих сервісів.
AMCB	AWS Management Console Bot — скорочена назва розроблювального додатку.
ПЗ	Програмне забезпечення.

ЗМІСТ

ВСТУП

Ідея та потреба розробки віртуального помічника для аналізу статистики витрат виникла наприкінці 2020 року. Помічник розроблявся, як для зовнішнього, так і для внутрішнього використання. З метою аналітики ринку компанії, дослідження витрат клієнтів та надання персоналізованих рекомендацій було прийнято рішення розробити власний чат-бот на платформах AWS та Telegram.

Метою роботи є зменшення часових та ресурсних затрат для здійснення аналізу та оптимізації хмарних інфраструктур клієнтів.

Об’єкт дослідження – аналітика витрат користувачів.

Предмет дослідження – витрати користувачів на надані сервіси хмарою AWS.

Методи дослідження. Інформаційний метод – проведено дослідження роботи та структури Telegram чат-ботів. Розробка системи – досліджувалась робота чат-боту в рамках serverless технологій.

Новизна результатів. Проведено дослідження з аналізу витрат користувачів в сервісах AWS. За браком існуючої інформації про Serverless Telegram чат-боти був створений новий метод розробки чат-ботів.

Практичне значення одержаних результатів. Можливість аналізувати інфраструктуру клієнтів в автоматичному режимі та надавати сповіщення. Дана система звільнить велику частку людських ресурсів для подальших проектів.

Особистий внесок студента. Проаналізовано стан альтернативних ринкових рішень для вирішення поставленої задачі та створено рішення для реалізації поставлених задач.

Апробація результатів кваліфікаційної магістерської роботи. Розроблений AWS чат-бот для аналізу статистики витрат клієнтів для компанії Elcore, часткове впровадження в компанії Zeller. Публікації: Тези даної роботи були опубліковані на конференціях «Об’єднані наукою», «9-та Східно-Європейська конференція Математичні та програмні технології Internet of Everything».

РОЗДІЛ 1

ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз ролі чат-ботів в месенджерах

Месенджери прийшли на зміну соціальним мережам, а сучасні боти приходять на заміну web-додаткам. Головний критерій проекту — створити інтерфейс, через котрий можна було отримувати статистику рахунків від хмарного провайдера, як приклад — AWS, кожному зареєстрованому користувачу. Цей інтерфейс повинен був бути реалізований за допомогою популярних месенджерів, які використовуються в IT компаніях для корпоративного зв'язку: Telegram, Slack, MS Teams. Саме тому розробка власного бота була найдоцільнішим варіантом.

1.1.1. Telegram

CEO компанії Павло Дуров представив світу першого Telegram бота у вересні 2015 року, а також програмний інтерфейс — Telegram API [1], через котрий можна було створити свого власного бота. Сьогодні Telegram має найбільшу кількість різноманітних додатків у вигляді ботів. Вони можуть допомогти вибрати подарунок, знайти квартиру, записатися до перукаря, відслідковувати стан якості повітря та багато іншого.

Кожен бот створюється через головний чат-бот The BotFather та всі адміністративні операції: створення, додавання/редагування/видалення команд, підключення/відключення додаткових функцій — проводяться через нього.

27 квітня 2021 року, з виходом версії Telegram API 2.0, з'явилася підтримка платіжних систем. Це допомогло розробникам створювати різні додатки-боти з можливістю інтернет-платежів, створювати підписку на власні сервіси. Також дане оновлення надало можливість створювати власні Інтернет-магазини в месенджері. Після виходу оновлення «Payments 2.0» актуальна версія Telegram Bot API мала відмітку 5.2.

Різноманітні технологічні впровадження, а також популяризація ботів завдяки додатку Telegram була основним критерієм вибору даного меседжера та його інструментів, як першу платформу для розробки та розвитку проекту.

1.1.2. Slack

Slack є популярним корпоративним месенджером, який почав своє існування в лютому 2014 року. З самого початку месенджер позиціонувався як заміна корпоративній пошті та стандартним засобам корпоративного зв'язку. І можна сказати їм це вдалося. В Slack реалізовані зручні дзвінки як один з одним так і з участю великої кількості осіб; в месенжджері релізована можливість публічних та приватних чатів/каналів в яких користувачі можуть зручно передавати інформацію, відмічати конкретних людей чи робити розсилку для всіх учасників. Також слід зазначити зручну форматизацію тексту повідомлень за стандартами markdown.

В Slack є окремий магазин додатків, який дозволяє швидко і зручно інтегрувати зовнішні сервіси та отримувати інфомацію з них чи пересилати її один одному з інформативним попереднім переглядом.

Slack, як і Telegram, має свій зручний та легкий API для розробки та інтергації власних додаків користувачами. Slack тісно взаємодіє з AWS — ви можете за допомогою сервісу AWS Chatbot надсилати важливі повідомлення напряму до Slack каналів.

В процесі аналізу платформ для інтеграції месенджер Slack посів друге місце.

1.1.3. MS Teams

Microsoft Teams — корпоративний меседжер, котрий входить в пакет послуг Office365. Перша версія додатку датована 14 березня 2017 року.

За функціоналом MS Teams нічим не поступається Slack. В ньому також є можливість організовувати канали, проводити онлайн конференції та обмінюватися повідомленнями, файлами. Щоправда можливостей форматування тексту буде трішки менше, але всі найголовніші пункти присутні.

З переваг можна виділити повну інтеграцію з сервісами Microsoft такими як: Azure, Office365 тощо. Також з виходом Windows 11 MS Teams інтегрований в систему, як месенджер за змовчуванням (прийшов на заміну Skype), але в обмеженій версії.

Також присутній магазин додатків для покращення інтеграції з іншими сервісами. Для ботів розроблено окреме API — Teams bot API, та SDK. Розробникам ботів, котрі використовують в роботі мову програмування C# та інтегрують додатки

за допомогою Azure відкривається більше можливостей завдяки єдиній екосистемі Microsoft.

Через інший спосіб розробки та інтеграції чат-ботів, функціонал для MS Team було вирішено реалізовувати останнім.

1.1.4. Аналіз логіки роботи чат-ботів в месенджерах

Звичайний бот загалом має дуже просту структуру (Рисунок 1.1).

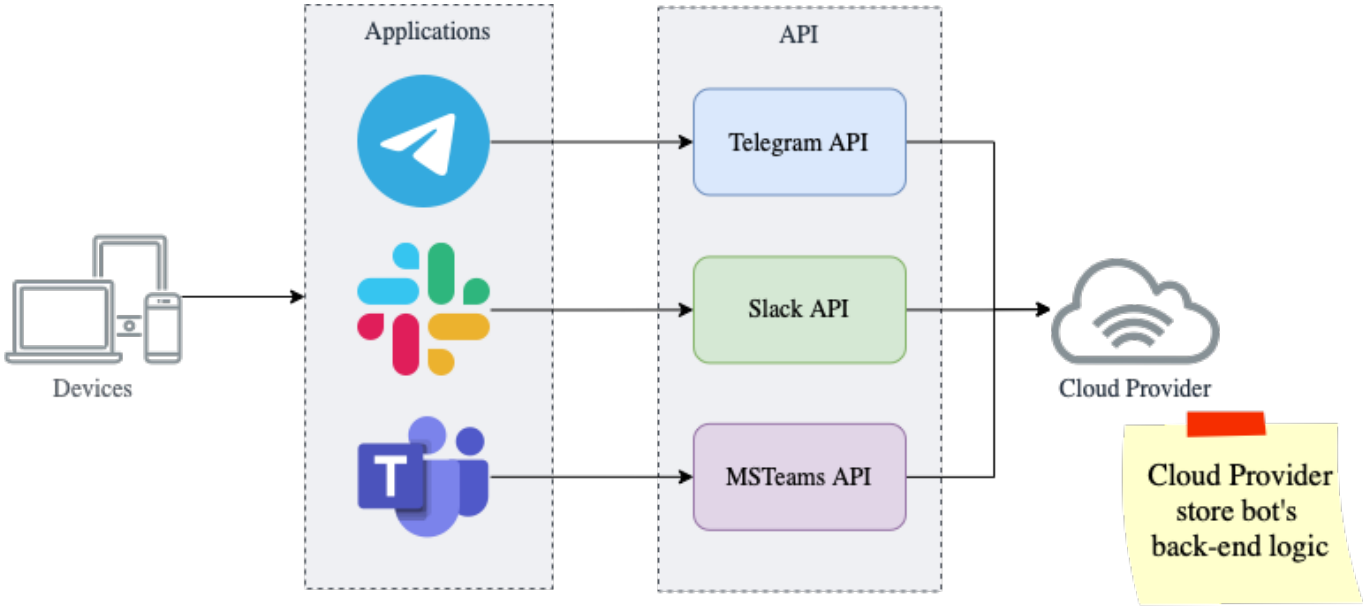


Рисунок 1.1. Верхнерівнева архітектура чат-ботів Telegram

У кожного з користувачів боту має бути встановлений клієнт-додаток: Telegram, Slack, Teams. Кожен раз, коли користувач відправляє повідомлення до чат-боту, воно направляється до дата центрів месенджеру, де в свою чергу мають обробитися логікою чат-боту, яка знаходиться на сервері — робота API.

Сервери обробки повідомлень можуть бути декількох видів:

- Власний виділений сервер (on premise).
- Виділена віртуальна машина в хмарного провайдера.

В свою чергу повідомлення можуть оброблюватися двома способами:

1. Періодичним опитуванням серверів месенджеру на наявність нових повідомлень (long pooling). В такому випадку API направляє до серверу чат-боту усі повідомлення, але не більше 100 повідомлень в одному запиті (це зв’язано з обмеженнями передачі інформації через протокол). У відповідь API очікує унікальний номер повідомлення (message id). Цей унікальний номер

слугує маркером для відправки наступних повідомлень на обробку. Рекомендується зберігати цей номер на сервері для запобігання дублювання повідомлень після непередбачуваного збою в роботі чат-боту.

2. Відправка кожного нового повідомлення до серверу чат-боту (webhook). Такий варіант обробки повідомлень є рекомендованим за документацією Telegram Bot API. Основними перевагами такого методу є:
   1) миттєва доставка повідомлення до боту та його обробка;
   2) можливість надсилати відповідь на повідомлення безпосередньо у відповіді на запит.

Основним обмеженням такого методу є наявність URI, який повинен бути захищений SSL сертифікатом для забезпечення коректної роботи HTTPS протоколу. Є можливість використовувати самопідписані SSL сертифікати, проте їх потрібно буде передати до Telegram API. Власне на цей URI будуть направлені нові повідомлення на обробку. Повідомлення надсилаються REST POST запитами, а повідомлення про їх успішну обробку є код 200 у тілі відповіді на запит.

1.2. Аналіз хмарних рішень для створення чат-ботів

Перші компанії, котрі почани надавати послуги публічної хмари, з'явилися в 2006 році. Це надало змогу вивести розвиток серверних технологій на новий рівень.

Всі обчислювальні потужності були віртуалізовані, це дало можливість більш гнучко налаштовувати та конфігурувати інфраструктури будь-якого розміру і складності.

Користувач, в свою чергу, отримав можливість запустити своє рішення за лічені секунди, можливість великого вибору обчислювальних потужностей та відмову від довгого очікування обладнання та подальших витрат на його обслуговування.

Хмарні обчислення надають можливість гнучко підходити до використання ресурсів: отримувати більше, коли це потрібно та віддавати зайве, коли навантаження на віртуальну машину майже відсутнє. Це дало можливість значно скоротити витрати на обслуговування інфраструктури.

Першою компанією, яка почала займатися хмарними обчисленнями і є лідером в них і до цього дня є Amazon Web Services (AWS). До трійки лідерів з надання послуг публічної хмари входять: AWS — Amazon, Azure — Microsoft, GSP — Google.

Serverless computing [2] — це підхід до використання хмарних сервісів тільки в момент потреби користувача. Обчислювальні ресурси створюються тільки в момент виклику самого сервісу і після виконання операції відразу знищуються.

Дана технологія дала новий виток розвитку хмарних технологій, а саме FaaS («функція як послуга») (Рисунок 1.2). Першою реалізацією FaaS була Amazon Lambda, представлена компанією AWS в 2014 році. Невдовзі аналогічні рішення з'явилися і у інших конкурентів на ринку.

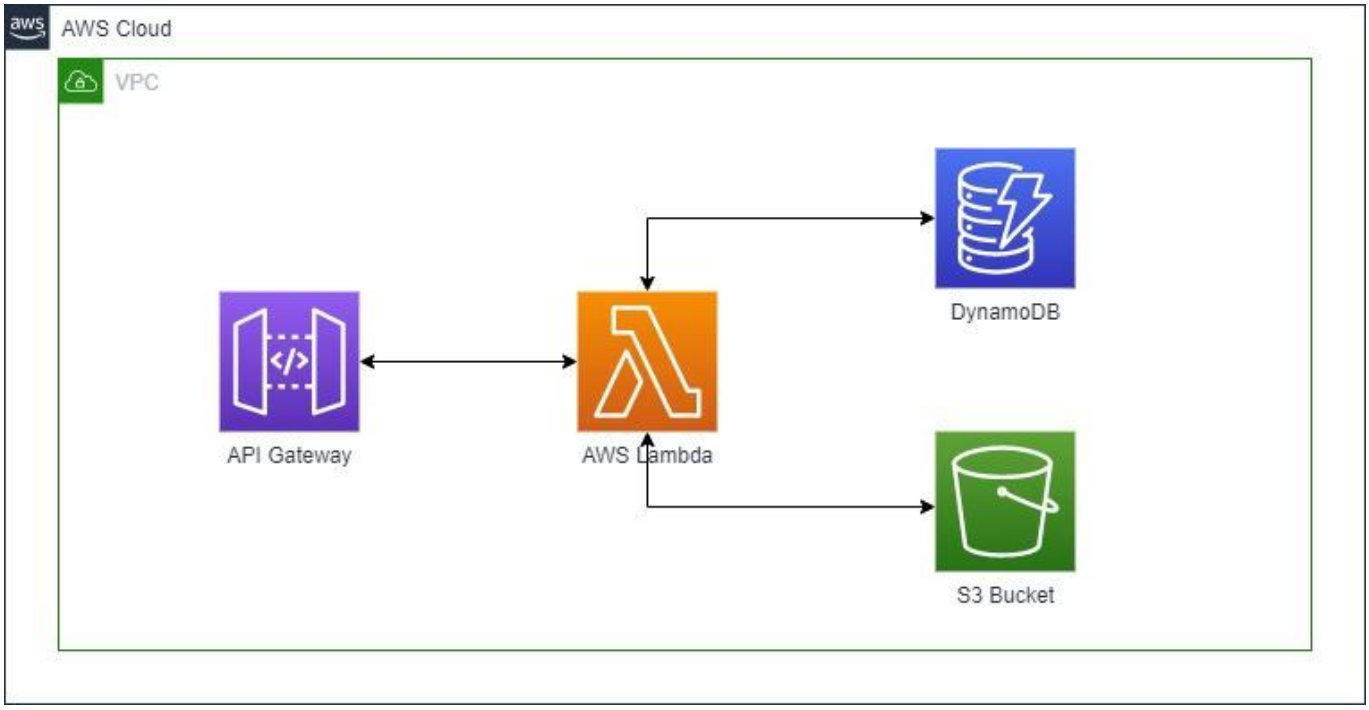


Рисунок 1.2. Базова архітектура використання serverless додатку

Головна особливість технології полягає в тому, що додаток розбивається на незалежні компоненти — функції, які виконуються тільки в момент виклику за мінімальну кількість часу з мінімальним використанням обчислювальних ресурсів.

Масштабування ресурсів в публічних хмарах було можливим в двох варіантах:

- вертикальне — збільнення розміру віртуального ресурсу, на якому запущено сервіс;
- горизонтальне — поступове додавання копій сервісу з ідентичними налаштуваннями для паралельної обробки запитів.

При використанні безсерверного підходу питанням виділення ресурсів та їх масштабуванням займається сам сервіс публічної хмари. У свою чергу це дозволяє обробляти будь-яку кількість запитів, яке може призвести користувач на додаток.

Основні відмінності звичайниз хмарних обчислень та безсерверних обчислень наведені в Таблиці 1.1.

Таблиця 1.1

Порівняння Cloud Computing і Serverless

Хмарні обчислення	**Serverless**
Використовує заздалегідь визначені конфігурації обчислювальних потужностей для віртуальних машин.	Споживає ту кількість обчислювальних ресурсів, які потрібні для виконання операції.
Вимагає наявність ОС або системи віртуалізації.	Завжди працює в віртуалізованому середовищі.
Вимагає спеціального налаштування для реалізації відмовостійкості продукту.	За змовчуванням має можливість до масштабованості та відмовостійкість.
Вимагає додаткової адміністрації ресурсів і регулярного оновлення.	Позбавлена недоліку конкурента, так як не займає ресурси в період простою.

Коштовість додатків розроблених за serverless технологією значно менша ніж використання навіть мінімальної віртуальної машини в публічній хмарі. Це ще більше наближає користувачів до головної ідеї публічних хмар — «pay as you go», тобто, споживати тільки стільки ресурсів, скільки потрібно і платити за ті ресурси, які були використані.

Для реалізації хостингу додатку був обраний AWS. Вибір обумовлений наступними критеріями:

1. AWS займає найбільшу частку ринку серед хмарних провайдерів (~ 33%).
2. Це хмарний провайдер, котрий заснував сервіс FaaS (Function as a Service).
3. Більшість клієнтів використовує сервіси AWS в своїй роботі.

1.3. Аналіз існуючих рішень

Для аналізу було зроблено вибірку серед найпопулярніших статей за даною тематикою, пошук можливих схожих додатків в магазинах месенжерів. Було виділено:

- для месенджеру Telegram чат-боти: Awsmanagementservice, AwsManageBot, AWS bot;
- для месенджеру Slack: AWS Chatbot;
- для месенджеру MS Teams: Cloudinspot;
- власна розробка від AWS;
- статті з описом власної реалізації сповіщень про використання ресурсів.

Результати проведеного аналізу конкурентів наведено в Таблиці 1.2.

Таблиця 1.2

Результати аналізу конкурентних рішень

Додаток	**Заключення**
Awsmanagementservice	Чат-бот не працює.
AwsManageBot	Чат-бот не працює.
AWS bot	Чат-бот виводить привітання, команд чи інструкції використання немає
AWS Chatbot	Власна розробка від AWS для Slack. Допомагає отримувати сповіщення за попередньо заданими параметрами. Що стосується сповіщення про використання коштів, можливо створити ліміт, після якого буде відправлено повідомлення про його настання. Загалом додаток визначено, як не дуже гнучкий для використання.
Cloudinspot	Чат-бот розроблений для MS Teams. Допомагає отримувати інформацію про використання базових сервісів AWS. Можливості отримати статусу використаних коштів не було виявлено.
Мобільний додаток від AWS	Є можливість моніторингу та керування сервісами AWS, перегляд використаних коштів. Мінусом є відсутність єдиного перегляду при використанні багатьох акакунтів AWS, не зручний інтерфейс при використання SSO.
Інтернет статті	Є опис декількох варіантів сповіщень: через пошту, СМС та месенжери. В більшості випадків статті вимагають від користувача знання платформи та, як мінімум, мови програмування Python3 на базовому рівні.

В результаті виявлено одне конкурентне рішення від самого хмарного провайдера AWS.

РОЗДІЛ 2

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕАЛІЗАЦІЇ ВІРТУАЛЬНОГО ПОМІЧНИКА

2.1. Постановка задачі та бізнес вимоги

Головна ідеєя чат-боту: створити програмне забезпечення, яке буде з деякою періодичністю відправляти повідомлення користувачам до чатів в різних месенджерах про стан рахунку за використані ресурси в публічній хмарі.

Після попереднього аналізу технічної частини, вирішено розширити функціональні можливості чат-боту та розбити їх впровадження на декілька етапів.

До першого етапу ввійшли компоненти для реалізації мінімального функціоналу для закриття першочергової потреби користувачів:

- сповіщення кожного дня про стан рахунку в хмарі AWS;
- можливість дізнатися стан рахунку за певний період календарних днів;
- можливість додавати чат-бот до групових чатів і можливість самостійно управляти доступом чат-боту у власній консолі AWS.

Наступними кроками розробки було призначено:

- оптимізувати видачу запиту для клієнтів в форматі зображення;
- створити декілька планів підписок для кінцевих клієнтів;
- розширити функціональність чат-боту на інші месенджери: Slack, MS Teams;
- створити аналітичний модуль з використання технологій штучного інтелекту для реалізації віртуального помічника з аналізу інфраструктури;
- додавання статистики витрат;
- створення власного логотипу продуку.

2.1.1. Сценарії використання системи

- Потенційний користувач 1

“Система повинна надсилати повідомлення кожного дня до мого персонального акаунта в Telegram. В повідомленні має бути зазначено скільки використано коштів за останню добу та за поточний місяць.”

- Потенційний користувач 2

“Повинна бути функція, котра дозволить самостійно налаштовувати періодичність автоматичних повідомлень в чат-боті. Наприклад надсилати стан рахунку один раз на день чи тиждень.”

- Потенційний користувач 3

“Було-б дуже чудово отримувати статистику витрат за певний проміжок часу. Наприклад показати у відсотках наскільки збільшився рахунок в порівнянні з попереднім місяцем. Це дуже допоможе швидше аналізувати та оптимізувати витрати.”

- Потенційний користувач 4

“Можливість додати чат-бот до групового чату команди або компанії. Це допоможе адміністраторам системи чітко бачити картину витрат та швидко реагувати на різні зміни у системі.”

- Потенційний користувач 5

“Можливість реалізації боту в різних месенджерах значно допоможе його поширенню серед кінцевих клієнтів.”

2.1.2. Аналіз ринку та альтернативних рішень

Основним конкурентом чат-боту, буде офіційний мобільний додаток AWS Console. Додаток дозволяє дивитися різноманітні метрики системи, раніше створені дошки моніторингу та стан рахунків. З мінусів додатку можна виділити необхідність створювати для кожного співробітника окремого користувача в консолі AWS та відсутність єдиної точки входу: на мобільних пристроях можна користуватися додатком, на ПК потрібно заходити до консолі AWS. Якщо включати всі вимоги до безпеки акаунтів, це може займати велику частку часу в роботі. Також слід відзначити деяку обмеженість в роботі мобільного додатку — при посиланні адміністративних команд до системи через додаток вони не виконуються, хоча такий функціонал в роботі зазначено. Огляд всіх альтернативних рішень та їх аналіз наведено в Розділі 1 Таблиця 1.3.1.

2.1.3. Вимоги до ресурсів розробки

- Операційні система — Windows 10 + WSL.
- Середовище розробки — Visual Studio Code.
- Пакет Atlassian:

- Jira — для відслідковування задач,
- Confluence — для електроного документообігу,
- BitBucket — система контролю версій.

- SDK — AWS SDK для мови програмування Python.
- AWS SAM [5] — фреймворк для роботи з serverless додатками.
- AWS CLI — консольний інтерфейс для роботи з сервісами AWS.
- Docker — програмне забезпечення для збірки та тестування ПЗ.
- База даних — AWS DynamoDB.
- Створений акаунт в хмарі AWS для розгортання інфраструктури.

2.2. Життєвий цикл програмного забезпечення

Розробка, супровід, розвиток програмного забезпечення та його інтеграція з хмарним провайдером та месенджерами буде виконуватися однією людиною. Найдоцільнішим варіантом життєвого циклу програмного забезпечення в даному випадку є каскадна модель (Рисунок 2.1) з ітеративним підходом.

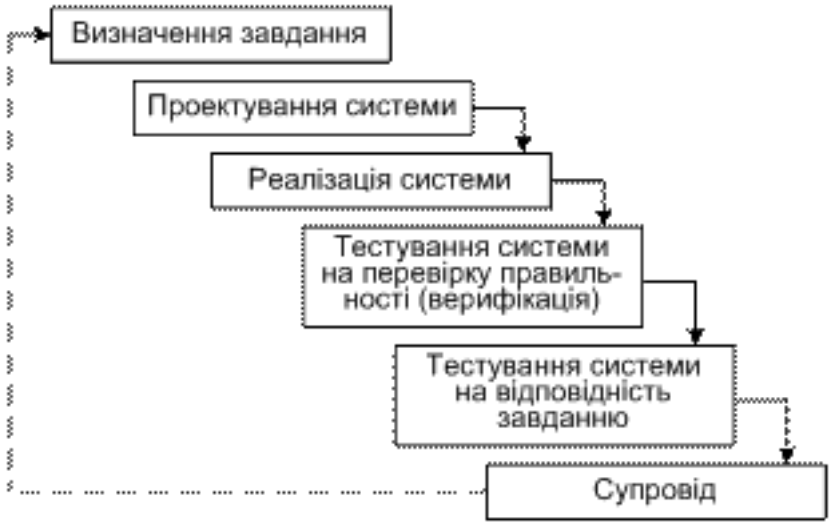


Рисунок 2.1. Каскадна модель ЖЦ ПЗ

На кожній ітерації визначаються цілі, проводиться розробка та тестування додатку. Результатом кожної ітерації є нова версія програмного продукту з підготованою документацією.

Спіральна модель життєвого циклу гарно підходить для розробки стартапів з невеликою кількістю людей в команді, або для роботи однієї людини над проектом.

До переваг каскадної моделі життєвого циклу можна віднести:

- фіксований набір стадій;
- кожна стадія завершується новою версією ПЗ з документацією;
- наступна стадія починається лише після закінчення попередньої.

З недоліків моделі можна виділити:

- негнучкість;
- перехід до нової фази можливий тільки після завершення попередньої фази;
- набір фаз фіксований;
- важко реагувати на зміни вимог;
- модель життєвого циклу заснована дуже давно та може бути не актуальною в багатьох варіантах розробки ПЗ.

На даний момент виконано 4 версії чат-боту:

1. *Версія 0.0.4* — перша ітерація чат-боту:
   a. Реалізація обміну повідомленнями.
2. *Версія 0.0.5* — відмова від сторонніх бібліотек:
   a. Прийняте рішення використовувати тільки офіційне Telegram API через відсталість зовнішніх бібліотек за функціоналом.
   b. Реалізація обміну повідомлень через відповіді на REST POST запити.
3. *Версія 0.0.7* — розширення функціоналу чат-боту:
   a. Додавання головних команд до чат-боту (реєстрація, довідка, видалення аккаунту).
   b. Додана підтримка роботи з груповими чатами.
   c. Налаштовані процеси безперервної розробки CI/CD.
4. *Версія 0.0.9* — автоматичні сповіщення про стан рахунку:
   a. Створення окремої функції, яка за Грінвичем надсилає повідомлення про баланс до клієнтів.
5. *Версія 0.1.0* — актуальна стадія розробки:
   a. Додавання статистики витрат.
   b. Реалізація більш простого способу додавання боту до групових чатів.
   c. Реструктуризація бази даних.
   d. Початок роботи над віртуальним помічником.

2.3. Архітектура AWS Console Management Bot

Хостинг провайдером для програмного забезпечення був обраний AWS. Вибір спричинений наступними критеріями:

- професійне знання сервісів AWS розробником;
- можливість відкрити безкоштовний акаунт на 1 рік;
- можливість реалізації додатку з мінімальними витратами на обслуговування інфраструктури.

2.3.1. Архітектура хмарної інфраструктури

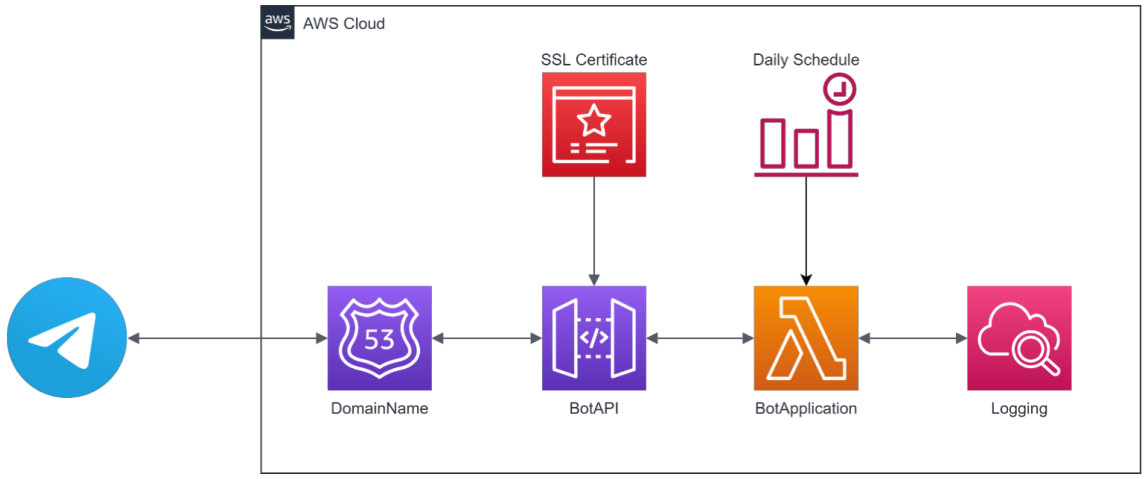


Рисунок 2.2. Верхнерівнева архітектура інфраструктури чат-боту

Головними вимогами в розробці архітектури інфраструктури є:

- Використання технологій хмарних провайдерів.
- Мінімізувати витрати в період простою програмного забезпечення.

Виходячи з вимог для реалізації функціоналу на вибір було надано два варіанти:

1. Зарезервувати один інстанс з невеликою кількістю обчислювальних ресурсів, а при потребі масштабувати його.
2. Розробити додаток за технологією безсерверних обчислень (serverless). В такому випадку потреба в постійних ресурсах виключається зі списку.

Остаточним рішенням стала розробка додатку за допомогою serverless. Переваги, недоліки та базова архітектура таких додатків наведена в Розділі 1 пункт 1.2. Враховуючи обмеження роботи додатку з використанням технології webhook-у, наведено в Розділі 1 пункт 1.1.4, виділяються наступні сервіси AWS для розробки:

1. *Amazon Route 53* [6] — сервіс адміністрації доменних імен. Через нього було зарезервовано домени для чат-боту, а також налаштоване URI для налаштування webhook через Telegram API.
2. *AWS Certificate Manager* [7] — сервіс котрий надає можливість створити власний внутрішній SSL сертифікат, який можна використовувати в інших сервісах AWS.
3. *Amazon API Gateway* [8] — сервіс, котрий реалізує роботу API в AWS. Завдяки цьому сервісу можна налаштувати роботу REST запитів, логіку обробки вхідних повідомлень від Telegram API та надати права доступу для користувачів. Також саме через цей сервіс підключається SSL сертифікат для коректної роботи додатку.
4. *Amazon Lambda* [9] — сервіс, який в собі реалізує технологію FaaS та зберігає всю виконуючу логіку чат-боту.
5. *Amazon CloudWatch* [10] — сервіс моніторингу роботи системи. Дозволяє будувати різноманітні графіки, а також зберігає звіти про роботу програми.
6. *AmazonEventBridge* [11] — сервіс,котрийвідповідаєзаавтоматизаціюпроцесів в AWS. Використовується для реалізації автоматичної відправки повідомлень до клієнтів.
7. *Amazon DynamoDB* [12] — сервіс для зберігання No-SQL таблиць.

На Рисунку 2.3.1.1 зображена архітектура інфраструктури з вище переліченими сервісами AWS. *Короткий опис інфраструктури:* клієнт, відправляючи повідомлення до чат-боту, запускає процес відправки запиту до бізнес логіки додатку. Через збережений URI (домене ім’я) в Telegram API в форматі REST POST запиту повідомлення надходить до BotApi, де маршрутизується до відповідної функції додатку. Функція в свою чергу зберігається в BotApplication.

В результаті, отримано легко масштабовану та економічну інфраструктуру для стабільної роботи чат-боту, котра буде розгортатися в публічній хмарі AWS.

Вимоги до платформи розгортання:

1. Захищеність.

2. Ізольованість.

3. Моніторинг ресурсів.

4. Автоматизація процесів.

2.3.2. Архітектура програмного забезпечення

З попередніх розділів вже відомо, що розробка додатку була узгоджена за технологією serverless. Для реалізації даного підходу нам знадобиться розуміння технологій AWS Serverless Application Model та Infrastructure as a Code.

AWS Serverless Application Model (AWS SAM) — це фреймворк розроблений компанією AWS для полегшення роботи розробників з безсерверними додатками.

Фреймворк надає наступні переваги:

- Конфігурація системи для єдиного розгортання.
- Додаток AWS CloudFormation.
- Впровадження найкращих практик в роботі з serverless додатками.
- Можливість локального тестування та відлагоджування.
- Глибока інтеграція з сервісами розробки AWS.

Infrastructure as a Code (Інфраструктура як код) [13] — досить поширений підхід серед розробників, котрі мають справу з хмарними сервісами. Головна ідея полягає в тому, що усі сервіси публічної хмари можна розгорнути за допомогою єдиного шаблону у форматі JSON або YAML. Це значно полегшує роботу розробників в плані розгортання інфраструктури. Всі зміни легко переглянути, збільшується можливість повторного використання коду, а також зменшується час на повторне розгортання подібної інфраструктури для тестування нових функцій в майбутньому.

Основним сервісом, який займається розгортанням інфраструктури за заданими шаблонами є AWS CloudFormation [14]. Сервіс є абсолютно безкоштовним у використанні, проте слід пам’ятати, що сервіси, котрі буде запускати AWS CloudFormation можуть бути платними.

Також до програмного забезпечення можна винести наступні вимоги:

1. Підтримка API.
2. Використання REST.
3. Автомасштабованість ресурсів.
4. Зовнішній інтерфейс для моніторингу.

Для кращого розуміння внутрішньої архітектури додатку, скористаємося UML діаграмами з деякими уточненнями: діаграма класів буде замінена на діаграму функцій, буде наведена скорочена діаграма прецедентів, відсутністю діаграми пакетів та діаграми активності.

Діаграма прецедентів системи (Рисунок 2.3)

- Специфікація прецеденту «login/logout»

1. *Короткий опис*

Клієнт ініціює процес реєстрації/видалення свого акаунту у чат-боті.

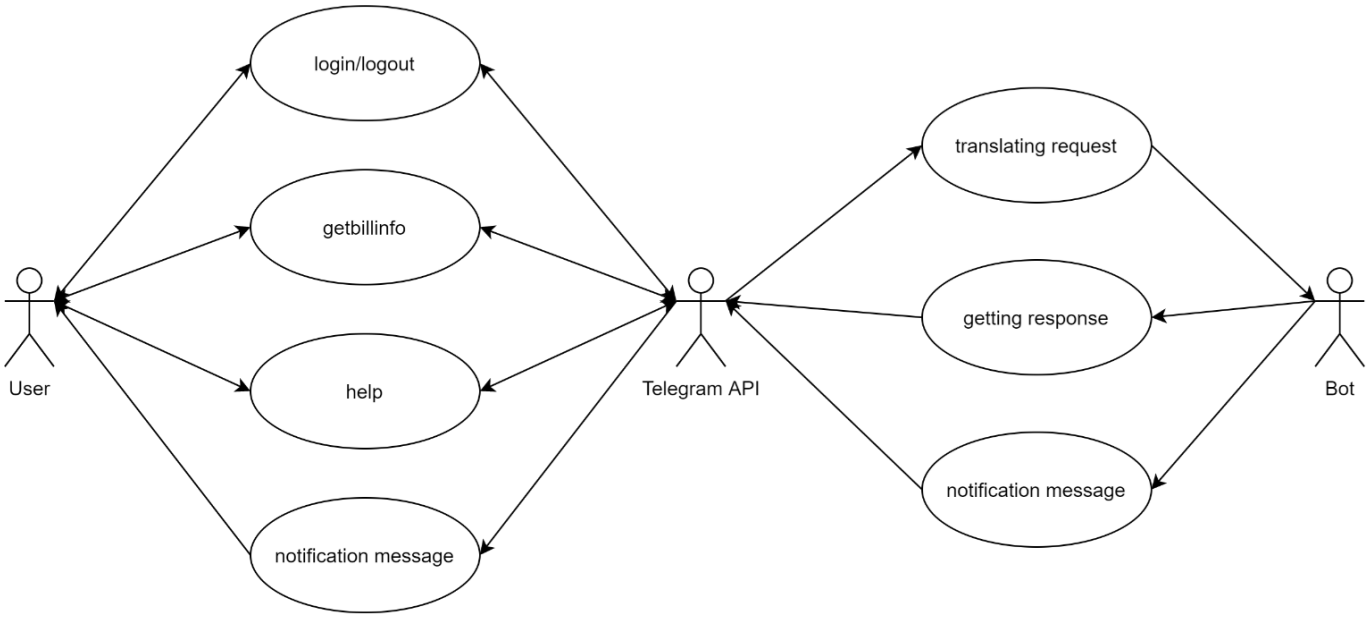


Рисунок 2.3. Діаграма прецедентів чат-боту АМСВ

2. Суб'єкт — Користувач, Telegram API, Bot.

3. Передумова — запустити чат-бот у додатку.

4. Основний потік

4.1. Користувач обирає команду "login" (з введеними даними входу) або "logout".

4.2. Telegram API транслює запит на сторону чат-боту.

4.3. Чат-бот проводить реєстрацію/видалення аккаунту до/з системи, та відправляє відповідь.

4.4. Telegram API транслює відповідь до клієнта.

4.5. Клієнт отримує відповідь з результатом операції.

5. Альтернативні потоки

А1. Якщо операція невдала до клієнту надходить повідомлення з результатом роботи запиту та інструкції по виправленню помилки.

6. Постумови

Успішна реєстрація/видалення аккаунту користувача чат-боту.

- Специфікація прецеденту «getbillinfo»

1. Короткий опис

Користувач натискає команду “getbillinginfo”. В результаті очікує отримати результат в якому буде надано актуальну інформацію про його витрати в консолі AWS.

2. Суб’єкт — Користувач, Telegram API, Bot**.**

3. Передумова — пройти попередню реєстрацію користувача.

4. Основний потік

4.1. Користувач натискає кнопку «getbillinginfo» або А1.

4.2. Telegram API транслює запит на сторону чат-боту.

4.3. Чат-бот бере актуальну дату та надсилає запит до скаут ну клієнта з метою отримати стан рахунку.

4.4. Після отримання успішного результату чат-бот формує відповідь та відправляє її або А2.

4.5. Telegram API транслює відповідь до клієнта.

4.6. Клієнт отримує повідомлення в якому зазначено рахунок за останню добу та за поточний місяць.

5. Альтернативні потоки

А1. Користувач разом з командою вводить діапазон дат за який він хоче отримати інформацію. (Запит має вигляд: /getbillinfo 2021-01-01 2021-05-01)

А2. Якщо результат містить в собі помилку, до користувача надсилається інформація про власне помилку та інструкції для її вирішення.

6. Постумови

Отримання повідомлення в якому відображається актуальний стан рахунку користувача.

- Специфікація прецеденту «notification message»

1. Короткий опис

Чат-бот формує повідомлення про актуальний стан рахунків користувачів та надсилає їх до кожного клієнта відповідно.

2. Суб'єкт — Користувач, Telegram API, Bot.

3. Передумова — пройти попередню реєстрацію користувача.

4. Основний потік

4.1. Спрацьовує трилер часу в системі.

4.2. Чат-бот запускає функцію, яка для кожного клієнта надсилає повідомлення про стан рахунку.

4.3. Telegram API транслює повідомлення до клієнта.

5. Постумови

Клієнт отримує повідомлення до свого аккаунт в Telegram про актуальний стан свого рахунку в AWS.

Діаграма класів системи (Рисунок 2.4)

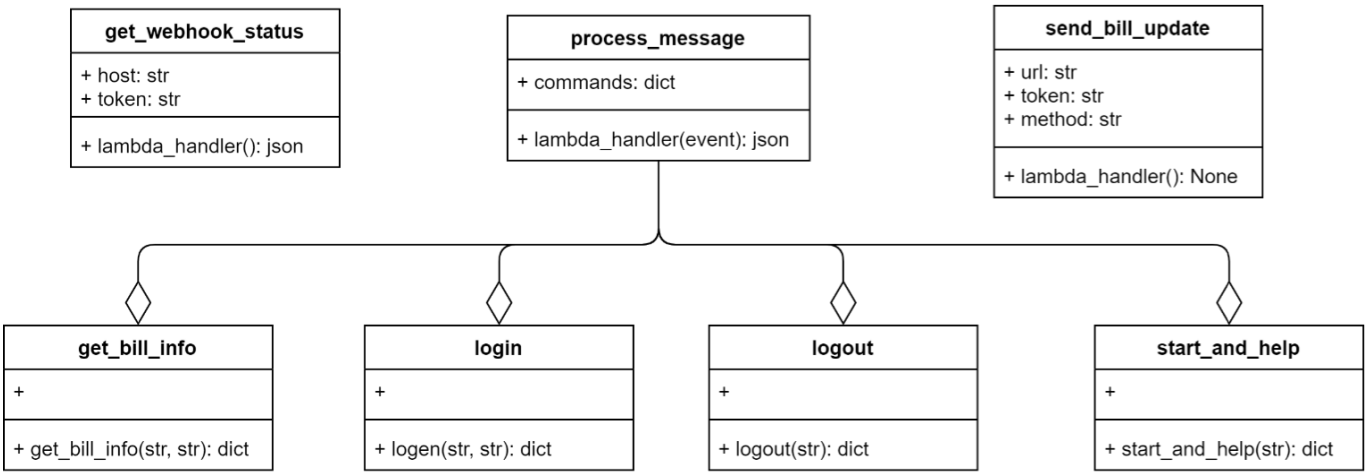


Рисунок 2.4. Діаграма функцій чат-боту AMCB

Чат-бот AWS Console Management Bot (ACMB) має 3 основні AWS Lambda функції: get_webhook_status, process_message, send_bill_update.

Функція get_webhook_status надає можливість отримати в JSON форматі статус каналу між Telegram API та бізнес логікою чат-боту. Запит повертає кількість повідомлень доступних до обробки, назву та час останньої помилки, URI до чат-боту. Для роботи функції було додано окремий виклик REST API з методом GET.

Функція process_message реалізує ядро чат-боту. Всі запити від користувачів надсилаються до цієї функції. Запити форматуються та відправляються на обробку

до вкладених функцій (див. Рис. 6). Після обробки, у тілі відповіді на запит відправляється інформація для користувача та Telegram API.

Функція send_bill_update — самостійна функція, котра надсилає кожному зареєстрованому користувачеві статус його рахунку за останню добу та поточний місяць.

Діаграма об'єктів системи (Рисунок 2.5)

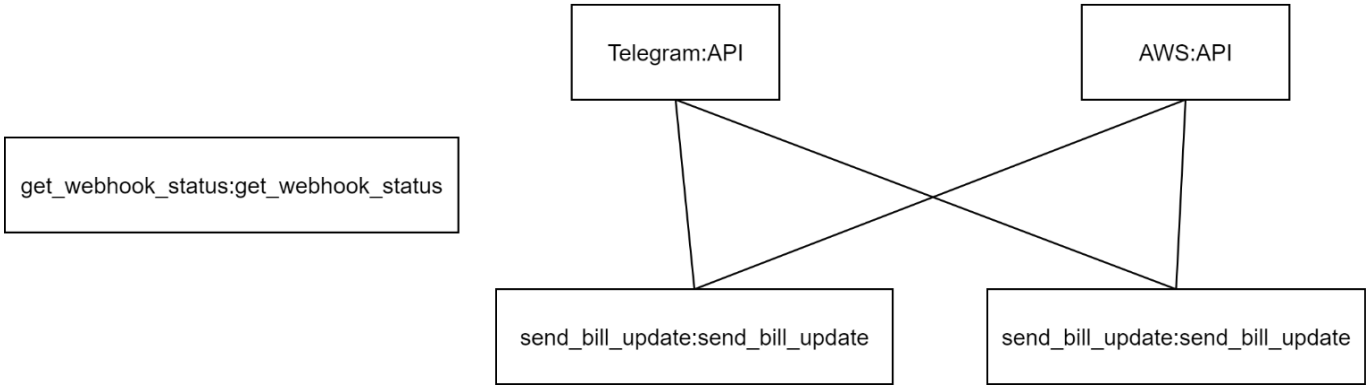


Рисунок 2.5. Діаграма об'єктів чат-боту AMCB

Діаграма об'єктів візуалізує використання вище описаних функцій та бібліотек в системі та їх взаємодію між собою.

Об'єкт Telegram є реалізацією власної бібліотеки, яка була створена для зменшення повторень та уніфікації коду при взаємодії з Telegram Bot API. На даний момент містить в собі методи відправки повідомлень.

Об'єкт AWS є реалізацією власної бібліотеки, яка була створена для зменшення повторень та уніфікації коду при взаємодії з сервісами AWS. Зараз реалізована можливість взаємодіяти з AWS Cost Management та Amazon DynamoDB.

Об'єкт get_webhook_status є реалізацією однойменної функції. Використовується розробниками та адміністраторами системи для розуміння роботи каналу між чат-ботом та Telegram API. Викликається через окремий GET запит системи.

Об'єкт process_message є реалізацією однойменної функції. Є головною функцією-форматором в обробці запитів користувачів. На даний момент реалізовані вкладені функції для обробки mandatory команд, а також обробка запитів на реєстрацію або видалення акаунту та команда отримання статусу рахунку.

Об'єкт send_bill_update є реалізацією однойменної функції. Реалізує відправлення повідомлень про стан рахунку до користувачів в заданий час.

***Діаграма взаємодії* системи**

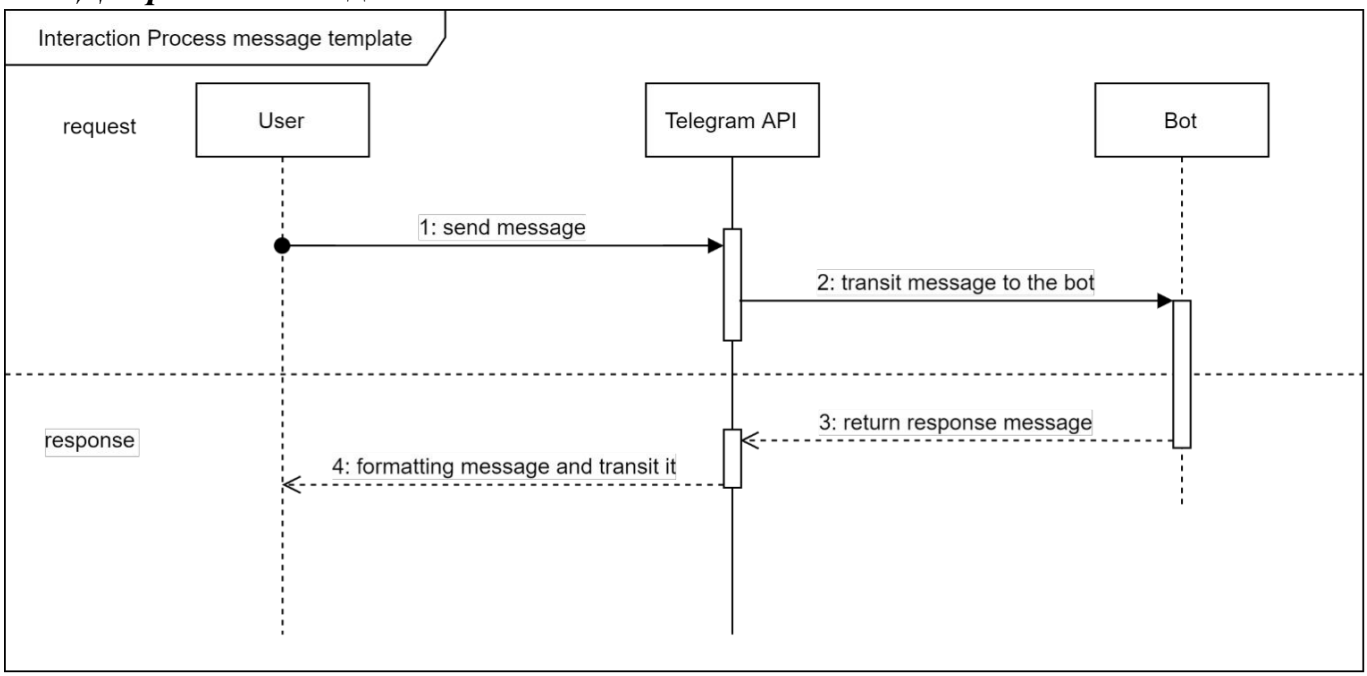


Рисунок 2.6. Діаграма взаємодії користувача та чат-боту

Діаграма взаємодії користувача з системою описує кроки обробки запитів клієнта. Час обробки інформації в хмарі не має перевищувати 5 с.

На діаграмі зображені наступні кроки:

1. Користувач надсилає повідомлення до чат-боту.
2. Telegram API транслює запит для виконання.
3. Функції чат-боту розпізнають команду та надсилають результат обробки.
4. Telegram API форматує повідомлення та доставляє його користувачеві.

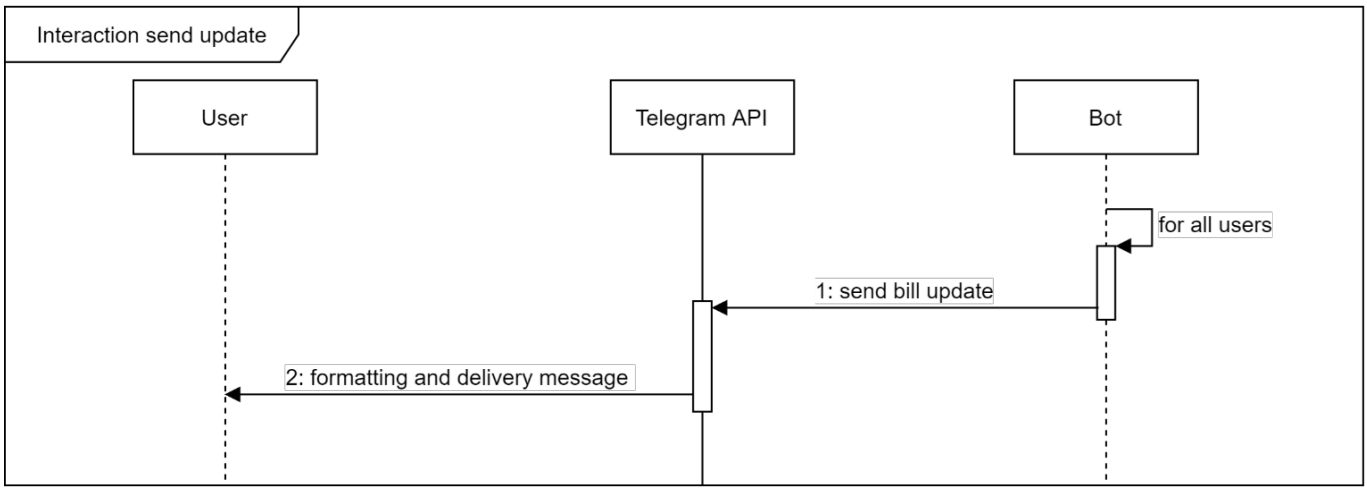


Рисунок 2.7. Діаграма взаємодії чат-боту з користувачем в індивідуальних сценаріях

Діаграма взаємодії описує роботу чат-боту з користувачами в режимі автоматичного сповіщення. Час обробки інформації в хмарі не має перевищувати 5 с для одного користувача.

На діаграмі зображені наступні кроки:

1. Чат-бот отримує список користувачів з БД.
2. Для кожного користувача система отримує актуальну інформацію про стан рахунку клієнта в хмарі AWS та надсилає повідомлення.
3. Telegram API форматує повідомлення та доставляє його користувачеві.

2.4. Опис CI/CD процесів

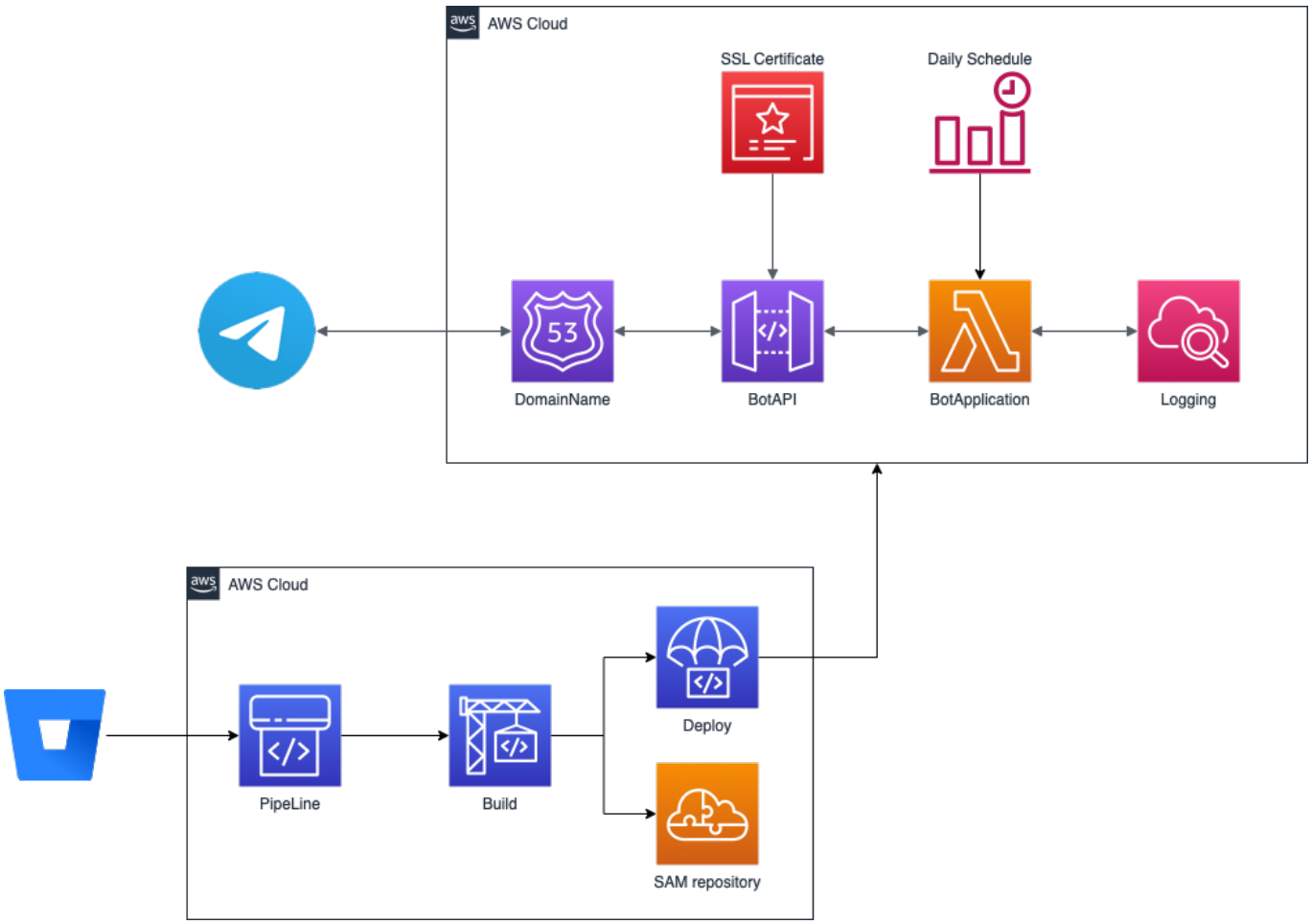


Рисунок 2.8. CI/CD процеси в розробці чат-боту AMCB

CI/CD процеси — Continuous Integration/Continuous Deployment – неперервна інтеграція/розгортання — практика програмного забезпечення, яка полягає у виконанні частих автоматизованих складань проекту для якнайшвидшого виявлення та вирішення інтеграційних проблем. Впровадження процесів CI/CD допомагає

зменшити кількість повторюваної роботи та зменшити людський фактор в розробці ПЗ.

На рисунку 2.8 зображено побудовану інфраструктуру CI/CD процесів. Головним віддаленим репозиторієм був обраний сервіс Atlassian BitBucket. До нього підключені сервіси AWS, котрі відслідковують зміни та запускають процеси:

- Amazon Code Pipeline [15] — головний сервіс котрий відслідковує зміни коду та запускає сценарії збірок.
- Amazon CodeBuild [16] — допоміжний сервіс для збірки бібліотеки, вихідного коду та шаблону розгортання інфраструктури.
- Amazon CodeDeploy [17] — допоміжний сервіс, котрий, за раніше створеним шаблоном, розгортає необхідну інфраструктуру в хмарі.
- AWS SAM Repository [18] — репозиторій, котрий зберігає релізи додатку. Потрібен на випадок розгортання інфраструктури на потужностях замовника.

На даний момент реалізовано безперервна збірка та оновлення додатку. На наступних кроках є впровадження автоматичних тестів.

2.5. Технічне завдання

2.5.1. Призначення системи

Для можливості аналізувати статистику власних витрат клієнтами компанії, буде створено нову версію чат-боту та оновлено інтерфейс роботи з системою.

2.5.2. Зміст системи (межі системи)

Система має мати можливість отримувати доступ до рахунку користувача. Права чат-боту в межах акаунту клієнта обмежені. Також клієнт може додати власні правила за бажанням. Всі параметри входу мають буди захищеними від компрометації. Завдяки цьому чат-бот зможе надавати регулярні повідомлення про витрати користувача та проводити статистичний аналіз витрат.

2.5.3. Огляд системи

Зміст системи: чат-бот складається з додатку: Telegram, Slack MS Teams, що виступає в роді клієнту (саме в додатку користувач може знайти чат-бот та почати роботу з ним завдяки раніше створеним командам) та публічної хмари AWS, яка являється надавачем сервісів для обробки всіх операцій чат-боту.

Функції системи: можливість створювати та видаляти акаунти користувачів, надавати інформацію про стан рахунку в акаунті публічної хмари користувача, аналіз статистики витрат користувача.

Характеристики користувачів: підходить для використання будь-якою людиною в якої є створений аккаунт в хмарі AWS.

Терміни та визначення

ACMB — AWS Console Management Bot.

AWS SAM — Amazon Web Services Serverless Application Model.

Терміни виконання розробки версії чат-боту 0.1.0 — до кінця липня 2022 року.

2.5.4. Посилання на репозиторій

https://github.com/Rybalochka- MS/AWSManagementConsoleBot

2.5.5. Системні вимоги

2.5.5.1. Функціональні вимоги

- Система має бути масштабованою.
- Мати єдиний файл налаштування взаємодії з зовнішніми системами.
- Система має надавати змогу створювати та видаляти користувачів.
- В системі потрібно реалізувати збір та аналіз статистики витрат користувачів.

2.5.5.2. Вимоги до usability

- UI системи має бути зручним та зрозумілим. В майбутньому командний інтерфейс повинен бути замінений на взаємодію кнопками.
- Команди та відповіді не мають буди перенасиченими додатковою інформацією. Все повинно бути стисло і лаконічно, для зрозумілої та швидкої роботи системи.

2*.5.6. Вимоги до продуктивності*

Система має підтримувати автомасштабованість, бути оптимізованою з боку використання оперативної пам’яті.

2*.5.7. Інтерфейс (взаємодія) системи*

- Основним інтерфейсом взаємодії є HTTPS протоколи.
- Основними видом передачі даних є JSON.

- Взаємодія між системою та користувачем виконується за допомогою REST запитів.

2.5.8.Операції системи

Відсутні, додаток працює за допомогою безсерверних технологій.

2.5.9. Стан системи

Відкрита продуктивна система, доступ до якої налаштовується при першому запуску чат-бота.

2.5.10. Фізичні характеристики

Система використовує хмарні обчислення AWS для розгортання та роботи.

2.5.11. Умови оточення

Доступ до чат-боту має проводитися через виділений URI. Для розробників та адміністраторів системи має бути створено окремих користувачів з доступом до консолі AWS.

2.5.12. Вимоги до безпеки

- База даних має бути зашифрованою.
- Система має мати захищені інтерфейси входу.
- Доступ до консолі AWS повинен видаватися згідно компетенцій співробітника.

2.5.13. Управління інформацією

Будь-яке розголошення роботи системи, а саме власних розробок, та використовуваних даних носить конфіденційний характер.

2.5.14. Політики і правила

Розробка та інтелектуальна власність належить Рибалочка Михайлу Сергійовичу. Розголошення будь-якої конфіденційної інформації суворо заборонено.

2.5.15. Вимоги до обслуговування системи протягом її життєвого циклу

Чат-бот розробляється за каскадною моделлю життєвого циклу. В цій моделі можна виділити два етапи: розробку та підтримку системи. До розробки системи входить додавання або вдосконалення існуючих функцій системи. До супроводу додавання в ітерації розробки задач для виправлення помилок, які виникли в ході виконання бізнес процесів додатку. Для

покращення якості чат-боту було впроваджено CI/CD процеси.

2.5.16. Вимоги до упаковки, навантаження-розвантаження, доставку і транспортування

Система має мати централізований файл розгортання інфраструктури. За допомогою налаштованих CI/CD процесів вихідних код збирається у бінарний файл та розгортається в хмарі AWS. Також на потребу користувача у власному хостингу системи реалізована можливість зберігання додатку у AWS SAM Repository.

2.6. Вимоги до тестування

На етапі розробки код має покриватися документацією та unit-тестами.

Мінімальні критерії оцінки:

- Покриття документацією — 60-80%
- Покриття тестами — 50-80%
- Реалізація перевірки відповідності має бути автоматизована та налаштована шляхом CI/CD процесів.

РОЗДІЛ 3

АНАЛІЗ ТА ВИБІР МЕТОДІВ ДЛЯ СТВОРЕННЯ АНАЛІТИЧНОГО МОДУЛЯ ПЗ

3.1. Перші кроки та ідеї створення нового модулю

Після випуску бета-версії додатку та отримання зворотнього зв’язку, почала готуватися нова версія додатку.

Основними побажанням користувачів стала можливість налаштовувати періодичність та час сповіщень. Додатково проаналізувавши нішу ринку програмного забезпечення, що займається моніторингом систем, до дошки задач додали створення можливості прогнозувати наступні витрати користувачів на певний проміжок часу. Після опитування користувачів, свормувалася структура нового сповіщення: витрати за останій тиждень, загальні витрати за поточний місяць та прогноз витрат на наступний тиждень. Такий розподіл дозволяє краще оцінювати ситуацію та реагувати на зміни.

В основу формули передбачення витрат взято три параметри:

- Витрати за поточні дати в минулому місяці — для включення можливостей періодичних та систематичних робіт: оновлення, тестування, аналіз тощо.
- Витрати за останій тиждень — актуальна інформація роботи системи.
- Витрати за останю добу — детальна інформація по використаним сервісам для подальшого аналізу.

Значенням додатково надали вагу для покращення результатів передбачення:

- Витратам за поточні дати у минулому місяці — 20%.
- Витратам за поточний тиждень — 30%.
- Витратам за останю добу — 50%.

Враховуючи попередні параметри виводимо функцію прогнозу (3.1):

$$K = \sum x \cdot \alpha = (\frac{a}{7} \cdot 0{,}2) + (\frac{b}{7} \cdot 0{,}3) + c \cdot 0{,}5 \qquad (3.1)$$

де: x — один з параметрів, α — вага параметру. В результаті отримуємо прогноз використання ресурсів за один день. Для отримання прогнозу на тиждень використовуємо формулу: $P = K \cdot 7$. Приклад прогнозу наведено на рисунку 3.1.

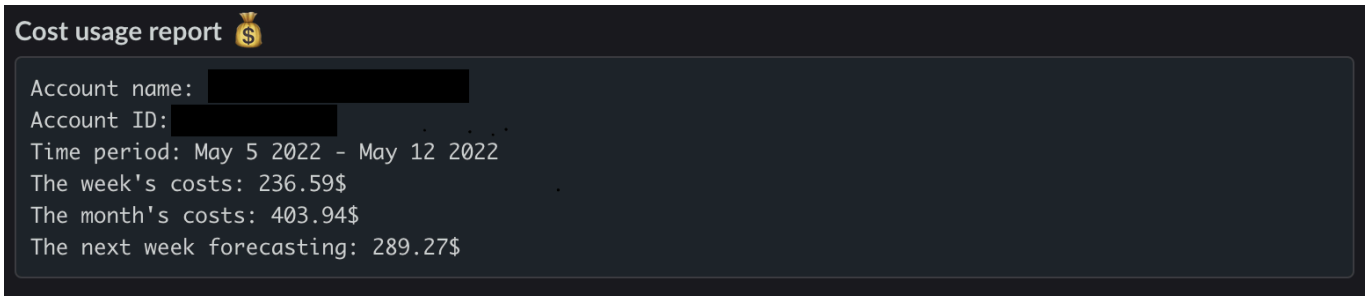


Рисунок 3.1. Звіт про використання ресурсів з прогнозуванням

3.2. Проблеми пов'язані з розробкою та інфраструктурою

Після реалізації базового функціоналу настало питання перевірки системи на варіант безпеки та цілісність роботи. Скориставшись офіційними джерелами AWS, провівши стрес тестування та аналіз несправностей, прийнято рішення по впровадженю наступних змін:

- Для вирішення періодичного дублювання повідомлень, збільшити час очікування виконання функції до 10 секунд.
- Для покращення безпеки та продуктивності, внести зміни в архітектуру:
  - Всі виконувані модулі мають бути ізольовані в привантій хмарі.
  - Використовувати тільки приватну версію API Gateway.
  - Використовувати сервіс AWS StepFunctions для розподілення логіки додатку та стабільності роботи.

Оновлена архітектура додатку наведена на рисунку 3.2.

3.3. Ідея створити аналітичний модуль помічника

Просто надавати інформацію про використання ресурсів було замало. Постало питання, чим можна допомогти користувачеві в аналізі своїх витрат. Іноді трапляються непередбачувані речі, коли використання ресурсів різко збільшується — аномалії. Виявлявлення, який сервіс спичинив цей скачок та надавати про нього інформацію, взято до наступного етапу розробки.

Розберемося, які данні надходять до системи, які данні можуть надходити в майбутньому.

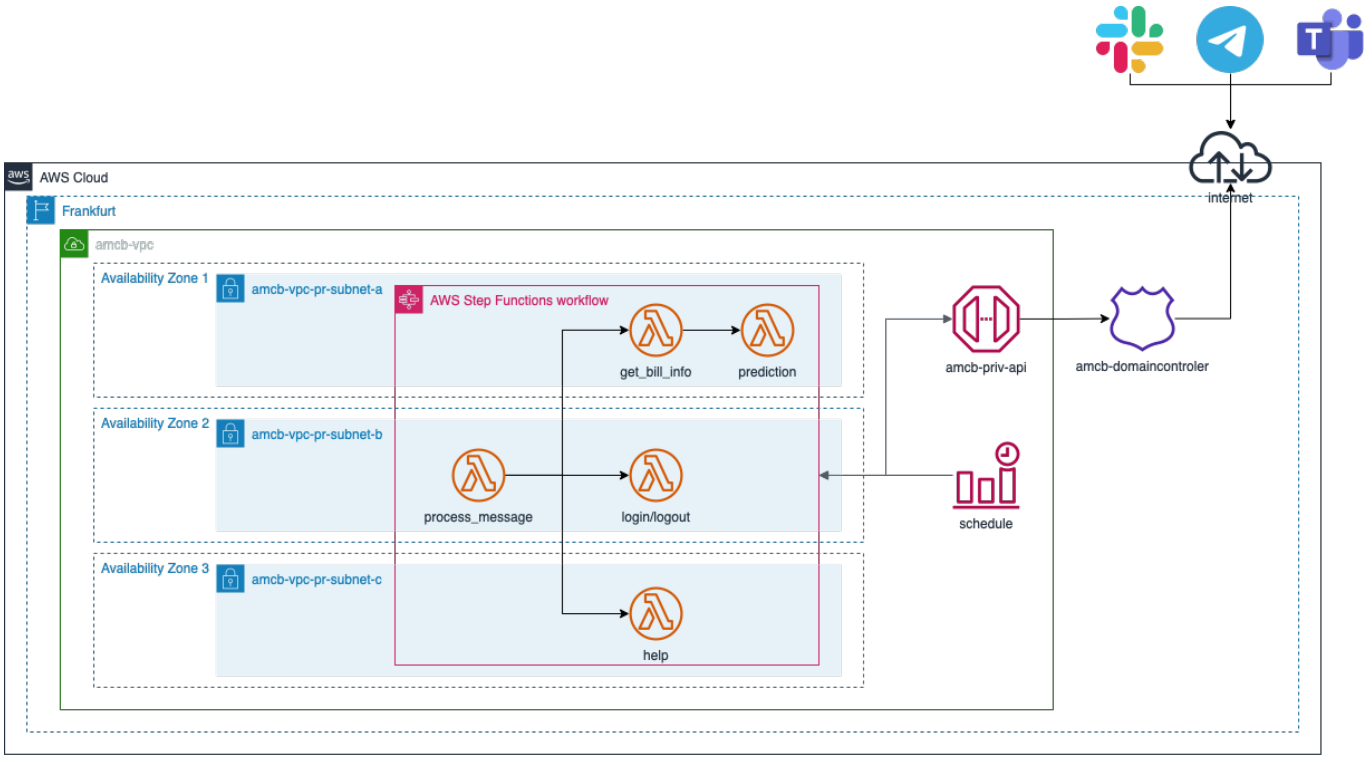


Рисунок 3.2. Оновлена архітектура додатку

На даний момент часу бізнес логіка чат-боту отримує рахунок клієнта за минулий день, тиждень та минулий місяць. Данні є загальними та їх дуже просто деперсоналізувати. На основі представлених даних клієнтів можна створити невеликий банк даних.

Для кожного користувача буде виводитись статистика приросту витрат за день та місяць при звітності системи. Аномалія розраховуєтья за формулою (3.2):

$$\Delta = \frac{x_i - x_{i-1}}{x_i} \tag{3.2}$$

– де x_i це рахунок на поточний день за актуальний місяць/день.

Якщо:

- $\Delta = 0$ — в акаунті користувача немає запущених сервісів;
- $\Delta = 1$ — витрати користувача стабільні;
- $1 > \Delta > 0$ — використання сервісів зменшилося;
- $\Delta \geq 1$ — використання сервісів збільшилося.

Кожного дня запускається запланована функція, котра збирає дані та перевіряє їх на аномалії. Якщо виявляється нетипова поведінка системи запускається наступний процес з пошуку сервісу, котрий збільшив свої потужності останнім

часом. Пошук виконується за алгоритмом: подається запит на отримання детального рахунку з урахуванням сервісів; для кожного сервісу проводиться аналіз на виявлення аномалій; помічені сервіси зберігаються для подального звіту; формується звіт у вигляді *сервіс - використання коштів до інциденту - використання коштів на даний час* (Рисунок 3.3). Такий підхід зміг значно покращити взаємодію користувача та системи, а також полегшити роботу адміністраторів систем.

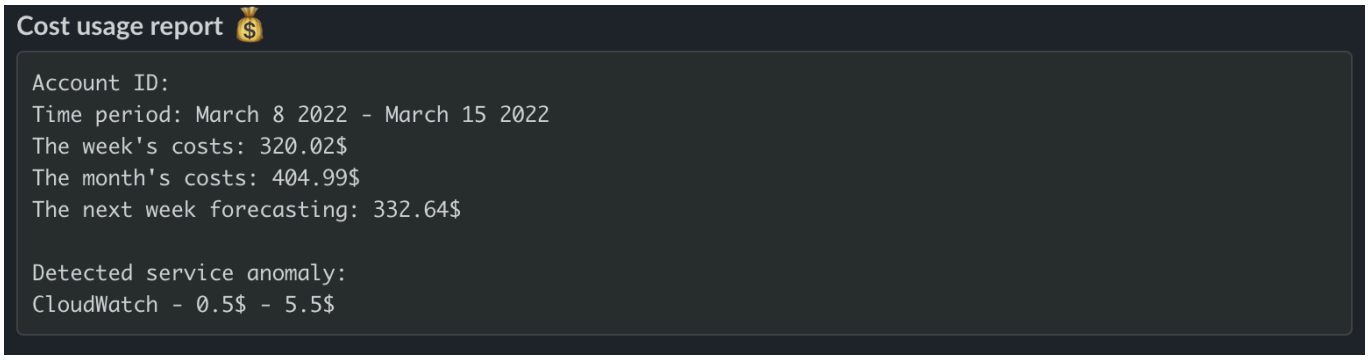


Рисунок 3.3. Приклад повідомлення з виявленням аномалії (кошториси приведені $/година)

Крайнім етапом стала реалізація рекомендацій з покращення роботи у випадку виявлення аномалій. Кожного місяця запускається збір статистики об’єму використаних сервісів користувачем. Для клієнтів, які за місяць збільшили свої витрати, буде запропоновано пройти опитування створене на базі сервісів, що вони використовували. Метою опитування є надання рекомендацій оптимізації інфраструктури з попередньою згодою користувача. Приклад опитування та рекомендації наведені в Додатку Б.

Всі проведені опитування та статистичні дані деперсоналізовуються та зберігаються в системі для подальшого використання в тренуванні ML моделі.

Для моделі штучного інтелекту було обрано:

- *Тип*: модель прямого розповсюдження.
- *Тренування*: стохастичний градієнтний спуск.
- *Похибка*: середньоквадратична.

Ці параметри при порівнянні з іншими варіантами є найбільш швидкими в опануванні та показують задовільний результат. Першим кроком став збір тестових статистичних даних. Для цього серед користувачів додатку було проведено

опитування та запит на згоду обробки персональних даних. Всі аналітичні дані зберігаються в NoSQL базі даних. Приклад використання наведено на рисунку 3.4.

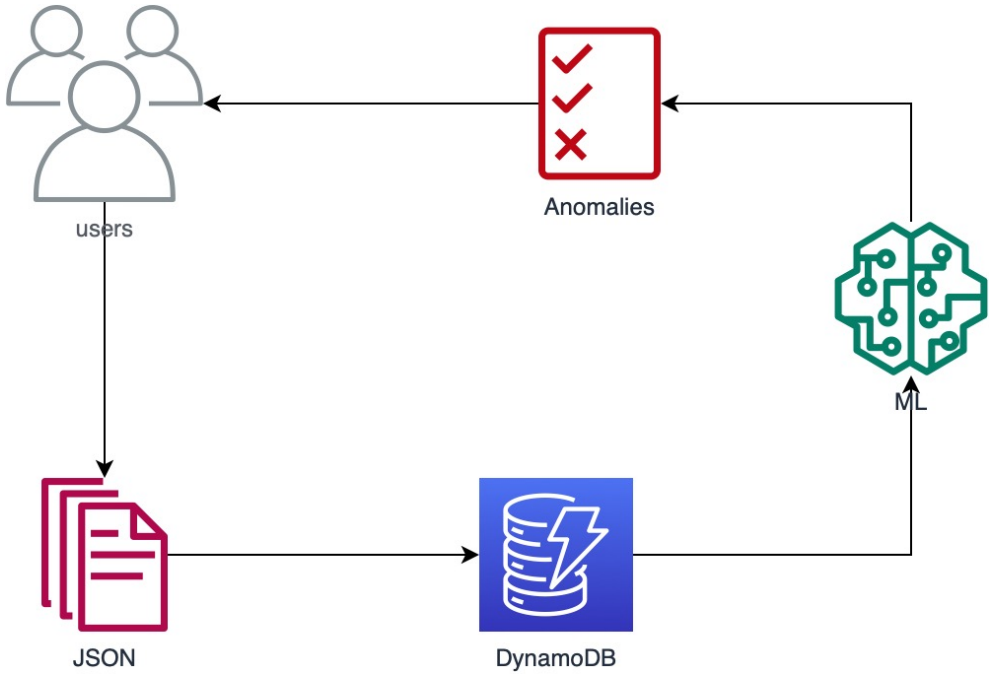


Рисунок 3.4. Приклад використання аналізу отриманих даних

Провівши перший аналіз даних, для виявлення слабих частин (Рисунок 3.5) було виправлено структуру даних та ігнорування/виправлення “пустих” даних.

service ↑	2021-09-30	2021-06-23	2021-06-10	2021-10-12	2021-07-08	2021-11-02	2021-09-06	2021-02-09	2021-07-05	2021-11-25	2021-05
Amazon API Gateway	9.0755997...	0.0002828...	0.0001335...	0.0000333497	0.0000704066	0.0000259...	0.0000148223	0.0000678...		0.0000037055	0.0006
Amazon Athena							0.00005	0.229775			
Amazon Cloud Directory											
Amazon CloudFront	0.0005108...	0.0002762...	0.0003232...	0.0000433182	0.0000197688	0.0000039...	0.0000145301		0.0000282...	0.0000024974	0.0003
Amazon Cognito	0		0			0					
Amazon DynamoDB	0.336802045	0.0072032...	0.000040165	0.0000188581	0.0000023358	0.0000032...	0.000002745	0.00000183	0.000000915	0.0000010675	0.0001
Amazon EC2 Container Reg...	0.0034928...			0.0068536008		0.0070820...				0.0073835709	
Amazon Elastic Compute Cl...		2.085800443	0.1941949...				0.6198886974			0.005004189	1.6532
Amazon Elastic Container S...	18.984120...	1.2433215...								0.0035577999	
Amazon Elastic File System	0.0000000...	0.0000001...	0.0000000...	0.00000006	0.0000002343	0.0000010...	0.0000000624	0.0000000...	0.0000002...	0.0000017112	0.0000
Amazon Elastic Load Balanc...		2.9726144...								0.027	
Amazon ElastiCache				2.5480358403		4.5574596...				4.5574596072	
Amazon Elasticsearch Servi...		0.6405782...	3.1459033...								
Amazon Forecast			0				7.9760673961				
Amazon FSx											
Amazon Kinesis Firehose			0.0000660...								

Рисунок 3.5. Графічне представлення отриманих даних

На етапі тренування моделі використовувалися декілька значень ітерацій: 150, 300, 600, 1000. Найкращий результат показала модель при тренуванні в 300 ітерацій (Рисунок 3.6). Приклад роботи моделі наведено на рисунку 3.7.

```
INFO:root:Epoch 0 loss: 0.337
INFO:root:Epoch 10 loss: 0.243
INFO:root:Epoch 20 loss: 0.239
INFO:root:Epoch 30 loss: 0.235
INFO:root:Epoch 40 loss: 0.231
INFO:root:Epoch 50 loss: 0.226
INFO:root:Epoch 60 loss: 0.221
INFO:root:Epoch 70 loss: 0.215
INFO:root:Epoch 80 loss: 0.209
INFO:root:Epoch 90 loss: 0.202
INFO:root:Epoch 100 loss: 0.193
INFO:root:Epoch 110 loss: 0.184
INFO:root:Epoch 120 loss: 0.174
INFO:root:Epoch 130 loss: 0.162
INFO:root:Epoch 140 loss: 0.149
INFO:root:Epoch 150 loss: 0.135
INFO:root:Epoch 160 loss: 0.121
INFO:root:Epoch 170 loss: 0.107
INFO:root:Epoch 180 loss: 0.093
INFO:root:Epoch 190 loss: 0.081
INFO:root:Epoch 200 loss: 0.071
INFO:root:Epoch 210 loss: 0.061
```

Рисунок 3.6. Похибка при тренуванні моделі

```
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
WARNING:root:Service 'Amazon Elastic Compute Cloud - Compute' use more costs than usually. Please check it.
ERROR:root:Detect anomaly usage in 'Amazon Elastic Compute Cloud - Compute' service. Amount: 1.890
ERROR:root:Detect anomaly usage in 'Amazon Elastic Compute Cloud - Compute' service. Amount: 3.090
ERROR:root:Detect anomaly usage in 'Amazon Elastic Compute Cloud - Compute' service. Amount: 2.231
WARNING:root:Service 'Amazon Elastic Compute Cloud - Compute' use more costs than usually. Please check it.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
WARNING:root:Service 'Amazon Elastic Compute Cloud - Compute' use more costs than usually. Please check it.
ERROR:root:Detect anomaly usage in 'Amazon Elastic Compute Cloud - Compute' service. Amount: 1.993
ERROR:root:Detect anomaly usage in 'Amazon Elastic Compute Cloud - Compute' service. Amount: 2.917
ERROR:root:Detect anomaly usage in 'Amazon Elastic Compute Cloud - Compute' service. Amount: 0.163
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
INFO:root:Cost usage in normal range.
```

Рисунок 3.7. Робота моделі при виявлені аномалій

Після тренування моделі, проведено тест на реальних даних. Модель відпрацювала коректно та вивела очікуваний результат. Для тесту було взято дані одного з користувачів в котрого був інцидент з сервісом моніторингу. Графічно зображені данні наведено на рисунку 3.8.

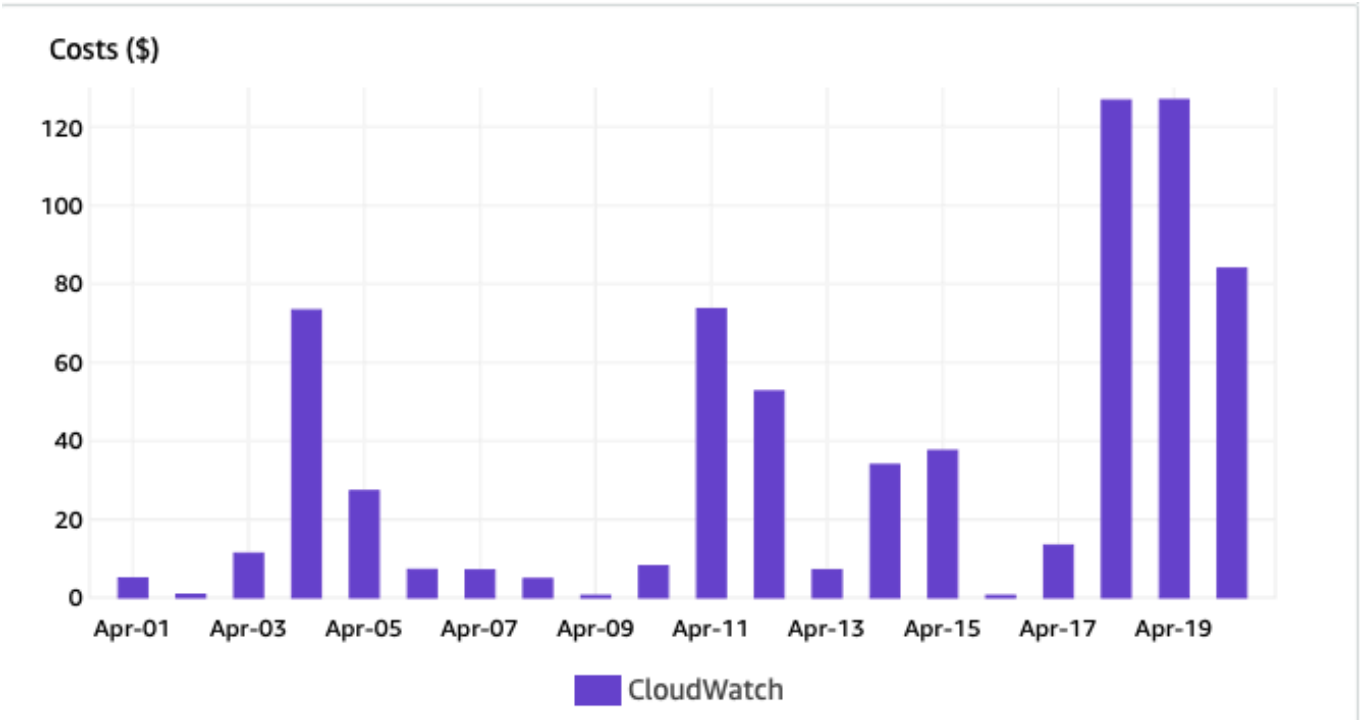


Рисунок 3.8. Дані користувача в період виявлення аномалії

З похибок моделі виявилося швидке звикання до змін. Кінцеві дані для моделі були вже звичними. Похибку можна списати на статистичні дані, так як ціль моделі швидко надати інформацію про зміни користувачеві в той момент, як вони почалися.

Наступним кроком реалізації моделі є можливість аналізувати паралельно декілька сервісів одночасно пов’язавши їх результат між собою. Це необхідно для комлексного аналізу інфраструктури та більш точного пошуку проблеми. Ціль реалізації впровадити повідомлення-рекомендації для вирішення знайдених проблем (аномалій). Для цього було складено алгоритм (Рисунок 3.9) та виведено формулу для аналізу (3.3):

$$A = \sum \frac{x_i - x_{i-1}}{x_i} \cdot \sum \frac{y_i - y_{i-1}}{y_i} \cdot \ldots \cdot \sum \frac{z_i - z_{i-1}}{z_i} \tag{3.3}$$

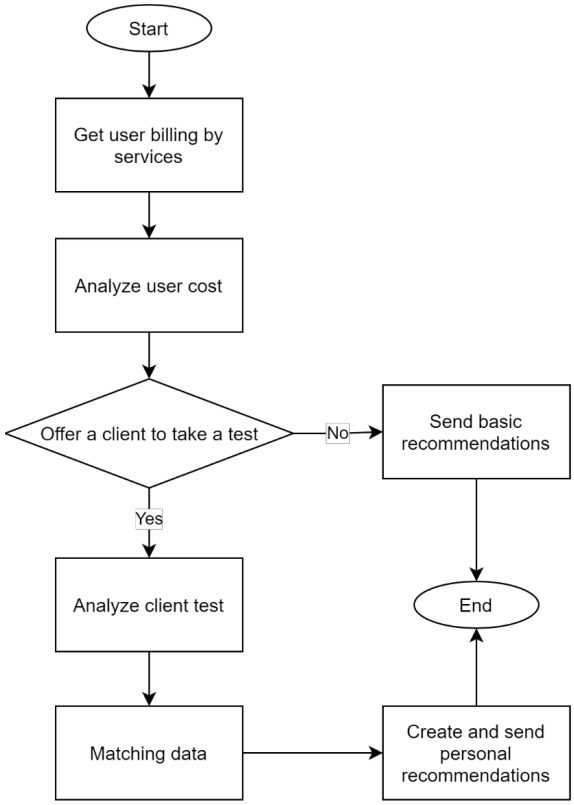


Рисунок 3.9. Алгоритм роботи рекомендацій

В даний час система працює з моделю для виявлення аномалій без опитування та рекомендацій. На меті є провести статистичні дослідження та дотренувати ML модель для видачі точних персоналізованих рекомендацій по оптимізації інфраструктури користувача в хмарі AWS.

РОЗДІЛ 4

Апробація програмного забезпечення

Програмне забезпечення попередньо покрито unit-тестами на 75%, відсоток знизився з впровадженням аналітичної моделі (Рисунок 4.1). Проведено стрес тестування системи, результатом є можливість одночасно оброблювати запити від 4 клієтів за секунду.

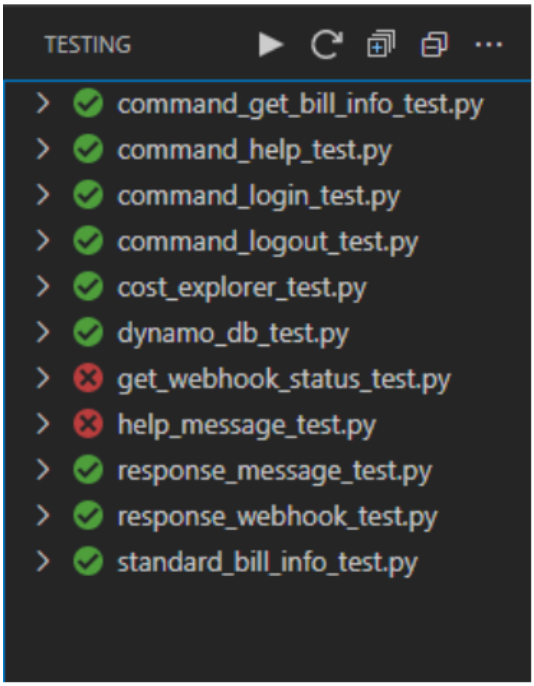


Рисунок 4.1. Виконання unit-тестів

Програмне забезпечення розповсюджується за допомогою AWS Serverless Application Repository. Впроваджене на використання в Elcore Cloud UA для внутрішніх клієнтів, в Unit Sity для одного з проектів, частково впроваджений функціонал в Zeller.

ВИСНОВКИ

Аналіз альнернативних рішень показав, що ідея створити віртуального помічника має шанс на успіх. Серед головних конкурентів можна виділити офіційні додатки від хмарних провайдерів AWS та Azure.

В ході дослідження платформи для реалізації, проаналізовано безліч архітектурних рішень та обраний найоптимальніший. Перевагами безсерверного підходу були: автомасштабованість за замовчуванням, новітність технології, оптимізація витрат на обслуговування інфраструктури. З мінусів можна виділити відсутність вже існуючих реалізацій конкретно під нашу задачу. Також не зовсім зрозумілу документацію офіційного Telegram API в розрізі налаштування комунікацій та обміну повідомленнями.

Було розроблено та впроваджено віртуального помічника для аналізу витрат в публічній хмарі AWS на базі месенджера Telegram та, частково, Slack. Виведено етапи розробки та тестування нових функцій системи. Етапи випуску нових версій заплановані не частіше двох разів на місяць.

Виконано дослідження при розробці аналітичного модулю додатку. Намічені наступні кроки розвитку. Результатом роботи став робочий модуль з аналізу та виявлення аномальної поведінки сервісів. Трішки статистики: найчастішим сервісом, котрий використовував найбільше ресурсів серед користувачів став CloudWatch.

Провести статистичні дослідження та натренувати ML модель для аналізу аномалій використання ресурсів та повідомлення що до оптимізації інфраструктури користувача в хмарі AWS є викликом з наукової точки зору. Автоматизація процесів, швидке отримання результатів аналізу, звільнення ресурсів — задачі розробки продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram Bot API — [Електронний ресурс]. — Режим доступу: https://core.telegram.org/bots/api
2. Безсерверні обчислення AWS — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/serverless/
3. Amazon EC2 — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/ec2/
4. Amazon RDS — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/rds/
5. AWS Serverless Application Model — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/serverless/sam/
6. Amazon Route 53 – Amazon Web Services — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/route53/
7. AWS Certificate Manager – Amazon Web Services (AWS) — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/certificate-manager/
8. Amazon API Gateway | Amazon Web Services — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/api-gateway/
9. AWS Lambda — [Електронний ресурс]. — Режим доступу: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
10. Amazon CloudWatch — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/cloudwatch/
11. Amazon EventBridge — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/eventbridge/
12. Amazon DynamoDB | Amazon Web Services — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/dynamodb/
13. Infrastructure as Code — [Електронний ресурс]. — Режим доступу: https://containersonaws.com/introduction/infrastructure-as-code/
14. AWS CloudFormation – інфраструктура як код — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/cloudformation/

15. AWS CodePipeline — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/codepipeline/
16. AWS CodeBuild — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/codebuild/
17. AWS CodeDeploy — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/codedeploy/
18. AWS Serverless Application Repository – Amazon Web Services — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/serverless/serverlessrepo/
19. AWS Well-Architected Framework — [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/architecture/well-architected/?wa-lens-whitepapers.sort-by=item.additionalFields.sortDate&wa-lens-whitepapers.sort-order=desc
20. Рибалочка М.С., Serverless, як наступний етап розвитку хмарних обчислень. —-Київ // Об’єднані наукою, 2020. — 216 с.
21. Рибалочка М.С., Вплив хмарних технологій на розвиток BigData — Київ // 9-та Східно-Европейська конференція Математичні та програмні технології Internet of Everything, 2021.

ДОДАТКИ

Додаток А

Лістинг програми

За неможливістю прикласти весь вихідний код чат-боту, до додатку прикріплено одну із головних функцій програми та код обробки однієї з команд.

```
"""
Processing Telegram WebHook Updates.

This function is main function for process update from Telegram bot.

Function have local module with Telegram commands.

Function use external libraries.
    * json - for parse incoming updates and package response.
    * logging - for enable logging.

This file can't be imported as a module.
"""

import json
import logging
from commands import get_bill_info, logout, start_and_help, login

# Enable logging level
logging.basicConfig()
logging.getLogger().setLevel(logging.INFO)

commands = {
    # personal chat
    '/start': lambda chat_id, message_text: start_and_help.start_and_help(chat_id, ""),
    '/help': lambda chat_id, message_text: start_and_help.start_and_help(chat_id, ""),
    '/getbillinfo': lambda chat_id, message_text: get_bill_info.get_bill_info(chat_id, message_text),
    '/login': lambda chat_id, message_text: login.login(chat_id, message_text),
    '/logout': lambda chat_id, message_text: logout.logout(chat_id, ""),
    # group
    '/start@AWSManagementConsole_bot': lambda chat_id, message_text: start_and_help.start_and_help(chat_id, ""),
    '/help@AWSManagementConsole_bot': lambda chat_id, message_text: start_and_help.start_and_help(chat_id, ""),
    '/getbillinfo@AWSManagementConsole_bot': lambda chat_id, message_text: get_bill_info.get_bill_info(chat_id, message_text),
    '/login@AWSManagementConsole_bot': lambda chat_id, message_text: login.login(chat_id, message_text),
    '/logout@AWSManagementConsole_bot': lambda chat_id, message_text: logout.logout(chat_id, "")
}

```

Рисунок А.1. Код головної функції

```python
"""       Rybalochka-MS, 4 months ago via PR #6 • Add documentation for all command functions. Refa…
Login new user to AWS Console Management Bot DataBase.

Function use external libraries.
    * get_response_message - for package response.
    * dynamoDb - use this module for adding users.
    * logging - for enable logging.

This file can be imported as a module.
"""
import logging
from amcb.aws_api import dynamo_db
from amcb.amcb_libs.response_message import get_response_message


Add Debug Configuration | Edit Debug Configuration (Beta)
def login(chat_id: str, message_text: str) -> dict:
    """
    Function register new user credentials for future use.

    :param chat_id: unique chat id from Telegram.
    :param message_text: incoming text message from Telegram Message object.
    :return: response json to Telegram API.
    """
    try:
        # Package user credentials data to dictionary.
        data = {
            'user_id': chat_id,
            'access_key': message_text.split(" ")[1],
            'secret_access_key': message_text.split(" ")[2]
        }
        logging.getLogger().info("User data: %s", data)
        # Adding user to DataBase.
        dynamo_db.add_user(data)
        # Creating response text.
        text = '*Login successful!*'
    except IndexError:
        logging.getLogger().info("Incorrect incoming user data!")
        # Creating response text with error and help info.
        text = 'Not found credentials data.\nUse unless pattern: /login AccessKey SecretAccessKey'

    # Return response message to Telegram API
    return get_response_message(chat_id, text)

```

Рисунок А.2. Функція обробки команди реєстрації

Додаток Б

Приклад опитування користувача та видачі рекомендацій

Припустимо, що користувач має розгорнутий web-додаток в хмарі AWS. Наприкінці поточного місяця він отримує рахунок та хоче оптимізувати свої витрати. Чат-бот попередньо проаналізував сервіси користувача. Серед них:

- Amazon EC2
- Amazon RDS
- Amazon CloudWatch

Через це, у опитуванні повинні бути включені питання:

1. Чи має додаток працювати 24/7 (позначений як критичний)?
2. Чи є трансакційна частина в додатку, котру не можна завершувати достроково?
3. Яка операційна система використовується в роботі додатку?

Припустимо користувач відповів наступним чином:

«Додаток має працювати цілодобово. Трансакційної частини у додатку немає. Всі дані зберігаються в базі даних. Операційна система для роботи додатку Linux.»

Після надання відповідей та аналізу, чат-бот повинен надати наступний результат: «Рекомендації. Створіть групу автомасштабування для обчислювальних ресурсів додатку, як мінімум в двох зонах присутності. Також можна використовувати менші конфігурації віртуальних машин, а також використовувати тип машин spot замість on-demand. Це наддасть змогу скоротити витрати на обчислювальні ресурси до 60%. Також можна зарезервувати використання БД в сервісі Amazon RDS, що наддасть змогу скоротити витрати на БД до 20%. Також для полегшення адміністрування та подальших змін в інфраструктурі рекомендовано використовувати сервіс Elastic Beanstalk.»

AWS Console Management Bot

Software Architecture Document (SAD)

CONTENT OWNER: Mykhailo Rybalochka

DOCUMENT NUMI	RELEASE/REVISION:	RELEASE/REVISION DA
• 1.0	• 0.0.9	• 22.04.2022

1. INTRODUCTION

Many product and retail companies use clouds. Sometimes we have a question: “how do we can monitor our cloud cost without login into the cloud console?”. This document describes the high-level application architecture for these tasks. But creating stuff for notification only was so boring. That’s why this application must have a new module to help analyse cloud infrastructure and send recommendations for improving it and saving your costs.

1. PURPOSE

This application must help them control and analyse cloud costs.

We need to create an application on a serverless framework with an ML model for cost notification and analysing architecture problems in the customer cloud if needed.

The application must connect as a bot to the next messengers:

- Telegram
- Microsoft Teams
- Slack
- Discord

Require messenger is Telegram. Support other messengers should be created on additional prise as next steps in project life.

Firstly application must send customise (time, date, account) cost notifications and detect anomalies.

ML model must analyze these anomalies and propose ways to fix them.

2. SCOPE

The project team and developers are working intently on making this project a fully operational technology system for the end-user.

Goals:

- Create a serverless application for notifications. All application setups must be written on IAC templates and stored in the serverless repo.
- Create an ML model to analyse cost anomalies.

- Create subscription plans for separate functional features:
  - Notification only – free for first account
  - Unlimited account notification
  - ML analysing

In addition to that, the author hasn't had the chance to talk to the other in-house developer to discuss general information about the original component of the system and any decisions made.

Thus, it is highly encouraged to at least one future revision of the document to correct inaccuracies, extend it and insert additional feedback and decisions that were made in the past.

3. GLOSSARY AND ACRONYM LIST

API	Application Programming Interface; Application Program Inter Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common object request broker architecture
COTS	Commercial-Off-The-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
OO	Object-Oriented
ORB	Object Request Broker
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE	Software Engineering Environment

SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
SW-CMM	Capability Maturity Model for Software
CMMI-SW	Capability Maturity Model Integrated - includes Software Engineering
UML	Unified Modeling Language

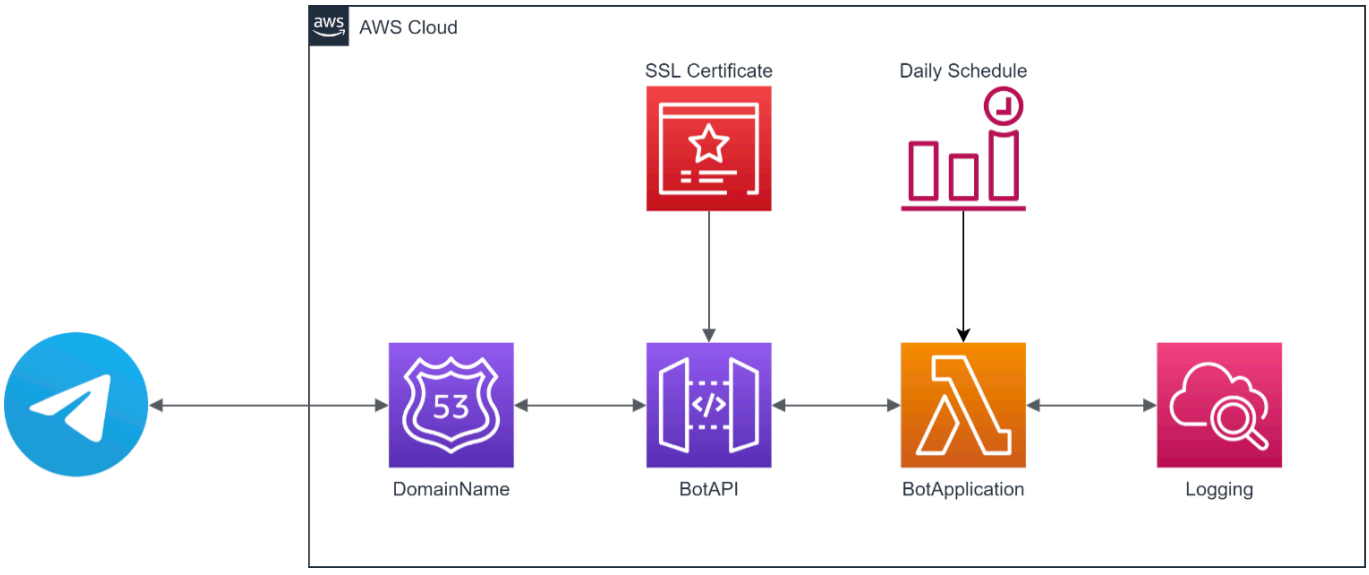
4. NON-FUNCTIONAL REQUIREMENTS

- High availability and durability. The system must grant access to user data 365/24/7 or 99,9 %.
- Scalable. The system must support more than 50 users per second.
- Security. All user data must be encrypted and access to it must be logged.
- Fault tolerance. The system must support users if it has problems – print highlight/ “how to” when errors have happened.

2. ARCHITECTURE OVERVIEW

This document is the first approach to present the information of this project in a structured fashion and discuss its architecture. It also provides guidelines for the upcoming half-to-a-year development.

1. ARCHITECTURE SUMMARY



System realises messenger chat-bot for billing notification. The main messenger is Telegram, another way clients for work with a bot. All requests to bot process on AWS platform.

2. BACKGROUND CONSTRAINS

For realising stable connection we must support web-hooks and HTTPS protocol. All modules must be realised as a function (framework constraint).

All application components are related to the cloud. The current application version uses the AWS cloud.

The programming language should be Python 3.8 or higher.

The main framework is AWS serverless application model. This framework helps us optimise costs and compute processing. All modules in AWS must be realised as a function and isolated.

Realise all API in API Gateway in AWS. All system logic must be downloaded to AWS Lambda. REST API calls must be delivered requests to the functions and send responses to subscribers in synchronous mode.

3. COMPONENTS

1. CLIENT

Telegram application is the client part of the system. This application support chatting and creating groups/channels/bots and s.o.

1. RESPONSIBILITIES

Creating an interface between user and system.

2. FEATURES

All messenger features.

3. SETUP

Install the Telegram application from App Store/Google Store/Microsoft Store or the official site.

2. BACKEND

Private Serverless application repository on AWS cloud. Support quick setup via a link for external companies and store version history.

1. RESPONSIBILITIES

Store backend application.

2. FEATURES

Easy and quick backend setup. Support versions history.

3. SETUP

Automated setup from IAC (infrastructure as a code) template from the repo.

3. DATABASE

NoSql database for storing all application data.

1. *RESPONSIBILITIES*

Store user data like credentials and cost history. This data uses for analytics.

2. *FEATURES*

Scalable, high availability and secure database. Data can be migrated from a backup or .csv file.

3. *SETUP*

Automated setup from IAC (infrastructure as a code) template from the repo.

4. ML DRIVER

Application or service for running ML model.

1. *RESPONSIBILITIES*

Store, update and run ML model.

2. *FEATURES*

Scalable, high availability and secure space.

3. *SETUP*

Automated setup from IAC (infrastructure as a code) template.

ADDITIONS

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	Viewpoint(s) that apply to that class of stakeholder's concerns
D2	The system must send messages every day to my personal Telegram account. The message must indicate how much money has been used for the last day and for the current month.
Elcore Cloud	There should be a feature that will allow you to adjust the frequency of automatic messages in the chatbot. For example, send an account balance once a day or week.
Elcore	It would be great to get cost statistics over a period of time. For example, show as a percentage how much the account has increased compared to the previous month. This will greatly help you analyze and optimize costs faster.
D2	Ability to add a chatbot to a group chat of a team or company. This will help system administrators to see a clear picture of costs and respond quickly to various changes in the system.
AWS	The ability to implement the bot in different messengers will greatly help its distribution among end customers