Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра

на тему: «Проект підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ.

Виконав студент II-го курсу, групи КБм-21-1
спеціальності 125 «Кібербезпека»
Лещенко Б.С.

Керівник
завідувач кафедри комп'ютерної інженерії та кібербезпеки, кандидат технічних наук, доцент
Єфіменко А.А.

Рецензент
професор кафедри комп'ютерної інженерії та кібербезпеки, доктор технічних наук, доцент
Воротніков В.В.

Житомир – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА КІБЕРБЕЗПЕКИ
СПЕЦІАЛЬНІСТЬ 125 «КІБЕРБЕЗПЕКА»

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерної інженерії
та кібербезпеки
Єфіменко А. А.
«25» 12 2022 р.

ЗАВДАННЯ
на випускну кваліфікаційну роботу

Студента Лещенка Богдана Сергійовича

Тема роботи: Проект підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ.

Затверджена Наказом університету від «25» жовтня 2022 р. № 501/с
Термін здачі студентом закінченої роботи «19» грудня 2022р.
Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

1. Розроблені технічні вимоги до проекту
2. Розроблена функціональна модель проекту
3. Розроблений застосунок для парсингу даних
4. Розроблені скрипти для початкового налаштування систем
5. Контейнеризований застосунок
6. Здійснено автоматичне розгортання застосунку

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Керівник _____
(підпис)

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання		Виконано
2	Розробка вимог до проекту підсистеми захисту		Виконано
3	Аналіз програмних продуктів для відправки пошти		Виконано
4	Вибір інфраструктури та хмарного провайдеру		Виконано
5	Дослідження механізмів захисту електронної пошти		Виконано
6	Побудова принципових схем роботи додатків		Виконано
7	Розробка детальних схем роботи додатків.		Виконано
8	Побудова діаграми послідовності дій		Виконано
9	Розробка парсеру		Виконано
10	Контейнеризація та розробка скриптів для початкового запуску застосунку.		Виконано
11	Розробка шаблонів для автоматичного розгортання рішення		Виконано
12	Розгортання систем агрегування та моніторингу на базі контейнеризованих технологій		Виконано
13	Аналіз результатів		Виконано
14	Написання пояснювальної записки		Виконано
15	Підготовка презентації та виступу		Виконано
16	Отримання відгуку керівника та рецензії на кваліфікаційну роботу		Виконано
17	Захист		Виконано

Студент _____
(підпис)

Керівник _____
(підпис)

РЕФЕРАТ

Кваліфікаційна робота складається з проекту підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ та пояснювальної записки до нього. Пояснювальна записка до роботи містить 60 сторінок, 41 ілюстрацій та 3 таблиць.

Метою роботи є розробка проекту підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ.

В роботі було досліджено теоретичні основи проектування підсистем захисту систем електронної пошти, розроблено вимоги до проекту підсистеми та виконано аналіз програмних продуктів для відправки пошти.

Було виконано проектування підсистеми захисту системи електронної пошти підприємства, виконано опис інфраструктури підприємства та визначено недоліки в наявній системі, було обрано інфраструктуру, хмарний провайдер послуг та сервер авторизації пошти.

Було виконано дослідження механізмів захисту електронної пошти, а саме: протоколи SPF, DKIM, DMARC, TLS.

Було виконано розробку функціональної моделі та застосунку підсистеми. Було побудовано принципові схеми роботи додатків, побудовано діаграми послідовностей дій. Розроблено парсер, виконано контейнеризацію та розроблено скрипти початкового запуску застосунку, розроблено шаблони для автоматизації розгортання. Розгорнуто систему моніторингу рішення на базі контейнеризованих технологій.

КЛЮЧОВІ СЛОВА: ЗАХИСТ, KUBERNETES, POSTFIX, MTA.

Зм.	Арк.	№ докум.	Підпис	Дата	Проект підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ.	Літ.	Арк.	Аркушів
Розроб.		Лещенко Б.С.					3	60
Керівник		Єфіменко А. А.		13.12				
Рецензент.		Безрєшнов		19.12		Державний університет «Житомирська політехніка», група КБ-1		
Зав. каф.		Єфіменко А. А.		19.12				

ЗМІСТ

ВСТУП

Актуальність теми. Про переваги електронної пошти є багато інформації, але про ризики, які стосуються безпеки електронної пошти, набагато менше. Найбільші ризики включають в себе, в тому числі, зловмисників, які намагаються скористатися потенційними вразливостями систем електронної пошти, щоб проникнути в них за допомогою фішингових атак, шкідливих програм, програм-вимагачів тощо.

Ось чому засоби захисту електронної пошти постійно стають сильнішими: щоб захистити як одержувачів, так і відправників.

Безпека електронної пошти має вирішальне значення для будь-якої організації, але це особливо важливо для компаній, які надсилають електронні листи великим спискам одержувачів. І як відправник, ви повинні вжити заходів, щоб гарантувати безпеку всіх електронних листів, які ви надсилаєте.

Електронні листи є ефективним початковим вектором зараження, оскільки майже кожна компанія використовує електронну пошту.

Сьогодні більшість Інтернет-сайтів використовують поштовий протокол SMTP для доставки та отримання поштових повідомлень. Sendmail все ще залишається одним із найпоширеніших серверів SMTP, але з ним виникали проблеми. Монолітна архітектура Sendmail є основною причиною численних проблем безпеки, і її може бути складно налаштувати та підтримувати. Спочатку Postfix був задуманий як заміна поширеного Sendmail. Його конструкція усуває багато можливостей для проблем безпеки. Postfix також усуває велику частину складності, пов’язаної з керуванням установкою Sendmail. Адміністрування Postfix здійснюється за допомогою двох простих конфігураційних файлів, і Postfix був розроблений із самого початку, щоб витончено справлятися з неочікуваними проблемами апаратного чи програмного забезпечення.

Саме тому обрана тема є актуальною зараз та буде актуальною ще довгий час.

Метою кваліфікаційної роботи є розробка проекту підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ.

Встановлена мета створює наступні завдання:

- аналіз програмних продуктів для відправки пошти;
- вибір інфраструктури та хмарного провайдеру;
- вибір серверу авторизації пошти;
- дослідження механізмів захисту електронної пошти;
- розробка парсеру для роботи з логуванням;
- койтернеризація застосунку Postfix та парсеру;
- розробка шаблону для розгортання в кластерних середовищах;
- розгортання та впровадження моніторингу для системи;

Об’єктом дослідження є процес розробки автоматизації розгортання захищеної системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ.

Предметом дослідження є методи і засоби забезпечення захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ.

Апробація результатів:

1. Лещенко Б.С., Єфіменко А.А., Вакалюк Т.А. Загрози безпеки доменної системи імен // Тези доповідей V Всеукраїнської науково-технічної конференції «Комп’ютерні технології: інновації, проблеми, рішення», м. Житомир, 01-02 грудня 2022 р. – Житомир: Житомирська політехніка, 2022.

2. Лещенко Б.С., Єфіменко А.А., Вакалюк Т.А. Проблематика захисту сервісів DNS у сучасному світі // Тези доповідей V Всеукраїнської науково-технічної конференції «Комп’ютерні технології: інновації, проблеми, рішення», м. Житомир, 01-02 грудня 2022 р. – Житомир: Житомирська політехніка, 2022.

1. ТЕОРЕТИЧНІ ОСНОВИ ПРОЕКТУВАННЯ ПІДСИСТЕМИ ЗАХИСТУ СИСТЕМИ ЕЛЕКТРОННОЇ ПОШТИ.

Електронні листи є ефективним початковим вектором зараження, оскільки майже кожна компанія використовує електронну пошту, і середньостатистичний працівник, як ми всі можемо собі уявити, отримує багато електронних листів. Величезна кількість електронних листів означає, що працівник має лише короткий проміжок часу, щоб присвятити кожному з них, і викликає у них помилкове відчуття безпеки. Кіберзлочинці користуються цим у фішингових атаках, які стали ще більш поширеними та ефективними з розвитком хмарної електронної пошти [4,6,44].

Електронну пошту можна назвати однією з найдавніших та найважливіших функції Інтернету. Майже кожна людина має персональну електронну адресу та кожна організація має свій домен, а деякі, навіть власні поштові сервери.

Історія електронної пошти в Інтернеті починається з початку 1970-х років, коли перші повідомлення були надіслані через Arpanet, попередницю сучасного Інтернету. З того часу електронна пошта була і продовжує залишатися найпоширенішим додатком в Інтернеті. У минулі часи доставка електронної пошти була відносно простою і зазвичай складалася з переміщення поштових файлів з одного великого хосту на інший великий хост, який обслуговував багато користувачів. Оскільки Інтернет розвивався, а сама мережа ставала складнішою, потрібні були більш гнучкі інструменти для переміщення пошти між різними мережами та різними типами мереж. Пакет Sendmail, випущений на початку 1980-х років, був розроблений для роботи з багатьма варіаціями поштових систем. Він швидко зайняв домінуючу роль у доставці пошти в Інтернеті [23,51].

Сьогодні більшість Інтернет-сайтів використовують поштовий протокол SMTP для доставки та отримання поштових повідомлень. Sendmail все ще залишається одним із найпоширеніших серверів SMTP, але з ним виникали проблеми. Монолітна архітектура Sendmail є основною причиною численних проблем безпеки, і її може бути складно налаштувати та підтримувати. Спочатку

Postfix був задуманий як заміна поширеного Sendmail. Його конструкція усуває багато можливостей для проблем безпеки. Postfix також усуває велику частину складності, пов’язаної з керуванням установкою Sendmail. Адміністрування Postfix здійснюється за допомогою двох простих конфігураційних файлів, і Postfix був розроблений із самого початку, щоб витончено справлятися з неочікуваними проблемами апаратного чи програмного забезпечення [14].

Postfix показує свою реальну цінність при роботі в стресових умовах. Навіть у простих середовищах програмне забезпечення може зіткнутися з несподіваними умовами. Наприклад, багато програмних систем поводяться непередбачувано, коли їм не вистачає пам’яті або місця на диску. Postfix виявляє такі умови та замість того, щоб посилювати проблему, дає системі шанс відновитися. Не дивлячись на небезпеки, Postfix вживає всіх заходів, щоб працювати стабільно та надійно [22-25].

Звичайно, як і будь-яка система, яка починала своє існування дуже давно, система електронної пошти еволюціонувала впродовж і паралельно розвитку всього Інтернету.

Отже, так само як і інфраструктура підприємств змінювалась спочатку в сторону гідридної, а, згодом, і до повністю хмарної, інфраструктура також потребує розвитку. Станом на сьогодні, все більше і більше додатків мігрують до Kubernetes кластерів, які розміщені в різних хмарних середовищах – це означає, що не лише безпосередньо додатки повинні бути перепроектовані під таке контейнеризоване середовище, але ще й все супровідне програмне забезпечення [36].

1.1 Розробка вимог до проекту підсистеми захисту

1. Загальні відомості.

1.1 Найменування проектної документації: «Проект підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ»

1.2 Призначення проекту підсистеми. Проект підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ, призначена для надійної та захищеної відправки електронних повідомлень з контейнерів, які знаходяться в кластерному середовищі.

1.3 Об’єкт середовища. Об’єктом середовища є Kubernetes кластер в хмарній платформі Microsoft Azure.

7. Основні функції системи.

2.1 Отримання. Система повинна здійснювати отримання електронної пошти, яка згенерована додатком, який розроблено підприємством.

2.2 Обробка. Система повинна здійснювати обробку електронних листів: переробка логів додатку в зручний формат для подальшої роботи.

2.3 Шифрування. Система повинна здійснювати шифрування електронних листів.

2.4 Логування. Система повинна здійснювати логування всіх відправлених листів: надсилання інформації про відправників та кількість відправлених листів в різні види баз даних.

2.5 Зберігання. У разі, коли сервер посередник недоступний або блокує доставку листів, система повинна зберігати листи, до моменту, коли сервер авторизації листів стане знов доступний.

2.6 Обмеження доступу. Система повинна передбачати можливість входу по паролю для запобігання несанкціонованому доступу до її ресурсів і налаштувань.

3. Характеристика середовища.

3.1 Хмарне середовище. Система повинна бути розміщена в хмарному середовищі Microsoft Azure.

3.2 Кластер. Кластерне середовище повинне використовувати кластерне середовище Azure Kubernetes Service. Параметри серверів, які мають бути включені в кластер:

Кількість серверів – 2

Розмір – Standard_B2s (vCPUs – 2, Memory – 4 GiB, Hyper-V Generations –

V1,V2, Proccessor – Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz, OS disk size – 1023 GiB)

3.3 База даних. В якості бази даних для довгострокового зберігання інформації про кількість відправлених листів повинна бути використана MSSQL база даних розміром 1024GiB.

3.4 Сервер моніторингу. Розмір серверу моніторингу – Standard_B2s (vCPUs – 2, Memory – 4 GiB, Hyper-V Generations – V1,V2, Proccessor – Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz, OS disk size – 1023 GiB)

4. Документація

4.1 Після розробки підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ, повинна бути розроблена відповідна документація, яка описує процеси в роботі даної підсистеми.

4.2 Повинні бути розроблені схеми взаємозв’язків між елементами системи.

4.3 Після розробки підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ, повинна бути розроблена відповідна документація, яка описує процеси відлагодження помилок в разі збоїв роботи елементів системи.

4.4 Повинна бути розроблена документація, яка описує процес розгортання підсистеми в кластері.

5. Тестування

5.1 Після розробки підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ, повинні бути виконані тести, а саме: перевірка відправки 1000 листів в період 1 хвилини, 1 години, та одного тижня.

1.2 Принципи організації обміну електронною поштою

Метою електронної пошти є надсилання повідомлень через Інтернет. Повідомлення — це записана частина інформації, яка асинхронно доставляється від відправника одному або кільком одержувачам. Асинхронна комунікація

означає, що повідомлення може бути спожито в довільній точці після його створення, замість того, щоб взаємодіяти з відправником [19].

Основні елементи обміну електронними листами представлено в таблиці 1.1 та на рисунку 1.1.

Таблиця 1.1 – Термінологія елементів електронної пошти

Акронім	Назва	Опис
MUA	Mail user agent	Клієнт для створення, надсилання, отримання та читання електронних листів, наприклад Microsoft Outlook, Apple Mail і Mozilla Thunderbird.
MSA	Mail submission agent	Сервер для отримання вихідних електронних листів від автентифікованих користувачів і постановки їх у чергу для доставки агентом передачі пошти (MTA).
MSS	Mail submission server	
MTA	Mail transfer agent	Сервер для доставки електронних листів у черзі та отримання їх на іншому кінці. Потім він пересилає отримані електронні листи агенту доставки пошти (MDA).
MDA	Mail delivery agent	Сервер для отримання електронних листів від локального агента передачі пошти (MTA) і зберігання їх у сховищі повідомлень (MS) одержувача.
MS	Message store	Сервер для зберігання електронних листів, отриманих від агента доставки пошти (MDA), і доставки їх до агента користувача пошти (MUA) одержувача.
MAS	Mail access server	

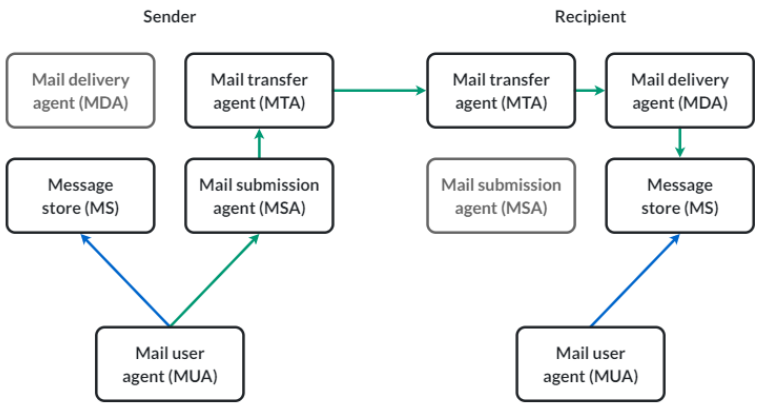


Рисунок 1.1 – Офіційна архітектура електронної пошти з підключеннями SMTP виділеними зеленим кольором та IMAP виділеними синім.

Жоден із серверів не повинен бути єдиною машиною. Крім того, вхідний MTA та вихідний MTA не мають бути однаковими.

Поштовий клієнт — це комп'ютерна програма для створення, надсилання, отримання та читання електронних листів. Він забезпечує інтерфейс, через який користувачі обробляють електронну пошту. Поштовий клієнт працює локально на пристрої користувача або віддалено на веб-сервері. Прикладами першого виду є Microsoft Outlook, Apple Mail і Mozilla Thunderbird. Прикладами останнього є Google Gmail і Yahoo! Mail під час доступу через веб-браузер [20].

Поштовий клієнт підключається до сервера вихідної пошти, щоб надсилати повідомлення для доставки іншим користувачам, і до сервера вхідної пошти, щоб отримувати нові повідомлення з поштової скриньки користувача. Обидва сервери автентифікують користувача, як правило, за допомогою імені користувача та пароля. Поштовий клієнт підключається до сервера вхідної пошти через інші інтерфейси, ніж сервери вихідної пошти[23].

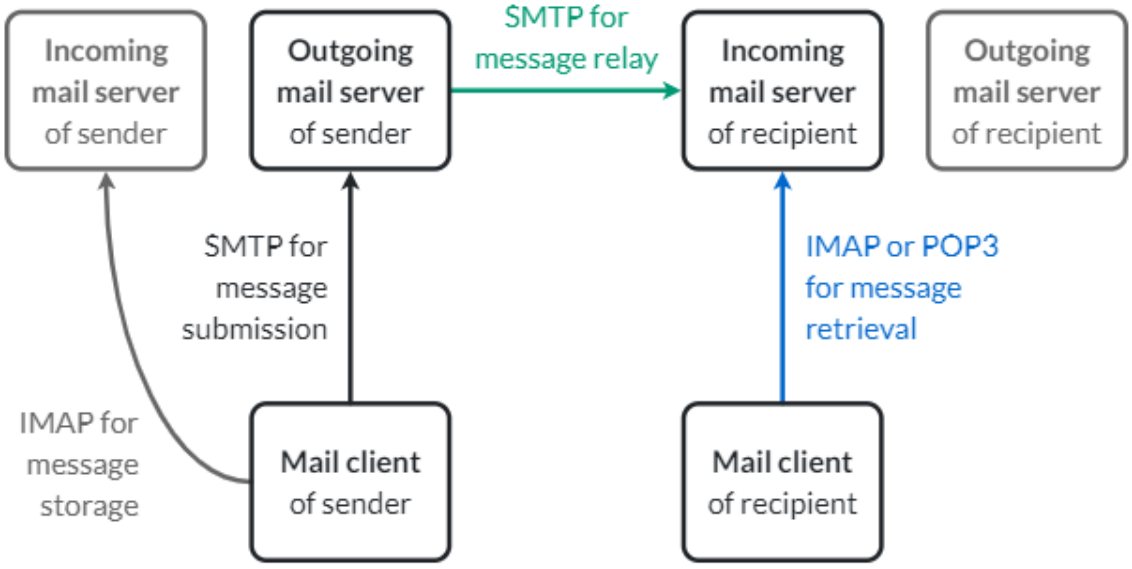


Рисунок 1.2 – Поштовий клієнт одержувача підключається до сервера вхідної пошти, використовуючи інший порт і протокол, ніж сервери вихідної пошти.

Сервер вихідної пошти приймає повідомлення від поштових клієнтів і ставить їх у чергу для доставки. Потім він визначає сервер вхідної пошти кожного одержувача та доставляє йому повідомлення. Сервер вихідної пошти діє як сервер

у взаємодії з поштовими клієнтами, але бере на себе роль клієнта під час ретрансляції повідомлення на сервери вхідної пошти. (З'єднання завжди ініціюються клієнтами.) Якщо сервер вихідної пошти не може доставити повідомлення, він надсилає повідомлення про відмову користувачеві, який надіслав повідомлення. Хоча сервер вихідної пошти не повинен змінювати вміст повідомлення, він додає інформацію про відправника. Перш ніж прийняти повідомлення, сервер вихідної пошти автентифікує користувача, як правило, на основі імені користувача та пароля[43].

Сервер вхідної пошти очікує підключення від серверів вихідної пошти інших користувачів. Коли сервер вихідної пошти підключається для передачі повідомлення, сервер вхідної пошти записує повідомлення разом з іншою інформацією сеансу, такою як IP-адреса відправника. Сервер вхідної пошти може відхилити вхідне повідомлення з кількох причин: можливо, одержувача електронної пошти не існує, його поштова скринька може бути заповнена, повідомлення може бути задовгим або відправнику не можна довіряти. Якщо повідомлення відхилено, сервер вихідної пошти може або спробувати повторно передати його пізніше, або повідомити користувача про невдалу доставку. З іншого боку, якщо сервер вхідної пошти приймає повідомлення, він також бере на себе відповідальність за доставку повідомлення. Якщо це не вдається, наприклад, коли повідомлення потрібно переслати, тоді сервер вхідної пошти повинен повідомити автора повідомлення[44].

Після завершення сеансу з сервером вихідної пошти сервер вхідної пошти додає додаткову інформацію, зібрану під час сеансу, до прийнятого повідомлення. Потім він оцінює, чи ймовірно повідомлення є спамом. Залежно цієї оцінки, повідомлення доставляється до папки «Вхідні» одержувача, поміщається в папку спаму одержувача або видаляється без повідомлення автора. Ось чому стандарт явно дозволяє серверам вхідної пошти мовчки видаляти отримані повідомлення. Якщо адреса отримання є псевдонімом, сервер вхідної пошти пересилає повідомлення на налаштовану адресу електронної пошти, а не доставляє його до

папки "Вхідні". Якщо адреса позначає список розсилки, сервер вхідної пошти надсилає повідомлення всім підписникам списку. Сервер вхідної пошти також застосовує фільтри та генерує автоматичні відповіді[32].

Сервер вхідної пошти очікує підключення від поштових клієнтів на іншому інтерфейсі. Щоб отримати доступ до поштової скриньки свого користувача, поштовий клієнт має надати відповідні облікові дані. Адреса електронної пошти та пароль користувача часто використовуються для автентифікації клієнта, якому в разі успіху надається необмежений доступ до поштової скриньки. Якщо сервер вхідної пошти підтримує OAuth, поштовий клієнт може надати маркер доступу, щоб отримати потенційно обмежений доступ до поштової скриньки користувача. Області дії, які пропонує Gmail, є прикладом того, як може виглядати обмежений доступ. Хоча обмежена авторизація є звичайною для інших служб, це ще не норма для електронної пошти. Після автентифікації клієнт може отримувати, зберігати та видаляти повідомлення. Він також може позначити їх як прочитані або позначити їх для подальшої обробки[32].

Зазначені вище об’єкти спілкуються за допомогою двох типів протоколів: вони використовують протоколи доставки для доставки повідомлень і протоколи доступу для доступу до поштової скриньки користувача. Як з’ясовано раніше, лише SMTP для передачі повідомлень є обов’язковим. Усі інші протоколи можна замінити у власній архітектурі. Існують спроби поєднати надсилання повідомлень і доступ до поштової скриньки стандартизованим способом[50].

Історично SMTP, POP3 і IMAP запускалися безпосередньо поверх транспортного рівня за допомогою протоколу керування передачею (TCP), що означає, що зв’язок не був ні зашифрований, ні автентифікований. Таким чином, будь-хто, хто мав доступ до однієї з мереж, через які проходив зв’язок, міг прочитати та потенційно змінити повідомлення.

Simple Mail Transfer Protocol (SMTP) вперше був визначений у RFC 821 у 1982 році.

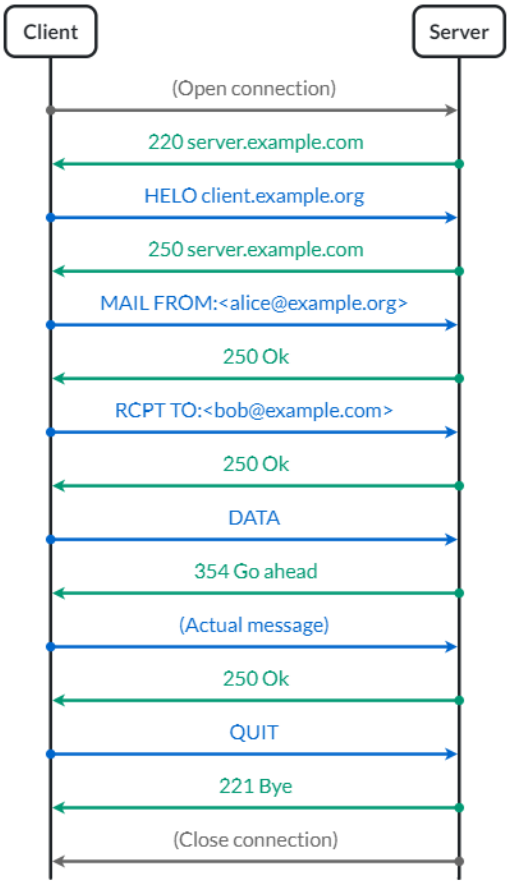


Рисунок 1.3 – Схема роботи протоколу SMTP.

Детальніше дослідження протоколів передачі електронних листів буде розглядатися в наступних розділах.

1.3 Дослідження проблем безпечної відправки пошти

Позначення повідомлень як спаму на основі їхнього вмісту та походження є проблемою класифікації. Імовірнісні класифікатори обчислюють, наскільки ймовірно, що вхідне повідомлення є спамом, на основі статистичних моделей. Популярним методом є наївна байєсівська фільтрація спаму. Оскільки повідомлення або зберігаються у папці «Вхідні», або відкидаються як спам, безперервну ймовірність потрібно перетворити на двійкове рішення. Якщо ймовірність того, що повідомлення є спамом, перевищує певне порогове значення, воно відхиляється. Рівень помилкових спрацьовувань має бути близьким до нуля, тоді як рівень помилкових негативів може бути вищим. З цієї причини поріг відхилення повідомлення зазвичай досить високий. Повідомлення з оцінкою вище

нижнього порогу зазвичай переміщуються до папки спаму, щоб користувач міг вирішити, що з ними робити [13].

Походження повідомлення відіграє вирішальну роль при оцінці його достовірності. За відсутності автентифікації домену репутація IP-адреси відправника визначає, чи буде доставлено повідомлення. Коли електронні листи більше не можна буде підробити, репутація домену відправника також стане важливою. Коли відбувається доставка електронних листів з нової IP-адреси, листи, з високою ймовірністю, потраплять у папку зі спамом своїх одержувачів, навіть якщо дотримуватися усіх найкращих практик. Щоб створити гарну репутацію, потрібен час, тому самостійно керувати сервером вихідної пошти є складним процесом. Для того, щоб досягти високого рівня доставки, зазвичай купується репутаційний капітал іншої компанії. Навколо цієї ціннісної пропозиції виникла ціла індустрія. Такі компанії відомі як постачальники послуг електронної пошти або постачальники послуг електронної пошти, і вони пропонують послугу транзакційної електронної пошти. Недоліком цієї системи репутації є те, що електронна пошта більше не є відкритою службою, якщо є необхідність придбання репутації для надсилання повідомлень від іншої компанії. Перевага цієї системи полягає в тому, що компанії мають стимули захищати свою репутацію: вони радше хочуть, щоб читачам було якомога легше скасувати підписку на їхню розсилку, ніж ризикувати бути позначеними як спам [29].

Експлуатація довірливості людей відома як соціальна інженерія. Окрім видавання себе за надійну організацію для фішингу, поширеною атакою є надсилання жертві електронного листа, який начебто надходить з її власної адреси.

Крім соціальної інженерії, підроблені адреси відправників можуть бути зловживані, коли електронні листи використовуються для автентифікації. Наприклад, люди часто можуть скасувати підписку на списки розсилки електронною поштою. Навіть якщо це не так, багато списків розсилки видаляють підписників, яким не вдалося автоматично доставити кілька повідомлень поспіль. Якщо список розсилки не використовує непередбачувані шляхи повернення

конверта зі змінними (VERP), повідомлення про відмову можна легко підробити, що означає, що ви можете скасувати підписку інших людей на список розсилки. Подібним чином, лише схвалені відправники можуть надсилати повідомлення всім передплатникам списку розсилки. Будь-хто, хто знає, як підробити електронні листи, може легко обійти це обмеження та спамити список розсилки [45].

За останнє десятиліття відсоток електронних листів, які під час пересилання зашифровуються та автентифікуються, значно зріс. Однак, коли ви надсилаєте електронний лист, немає гарантії, що конфіденційність і цілісність повідомлення буде захищено, коли воно передається з сервера вихідної пошти на сервер вхідної пошти одержувача. Це особливо проблематично, коли електронна пошта використовується для виконання важливих для безпеки операцій, наприклад скидання пароля. Завдяки зворотній сумісності протоколи електронної пошти захищені лише від пасивних зловмисників.

Gmail надає простий спосіб перевірити, чи було отримане повідомлення автентифіковано та зашифровано під час передавання, що дозволяє користувачам оцінити автентичність.

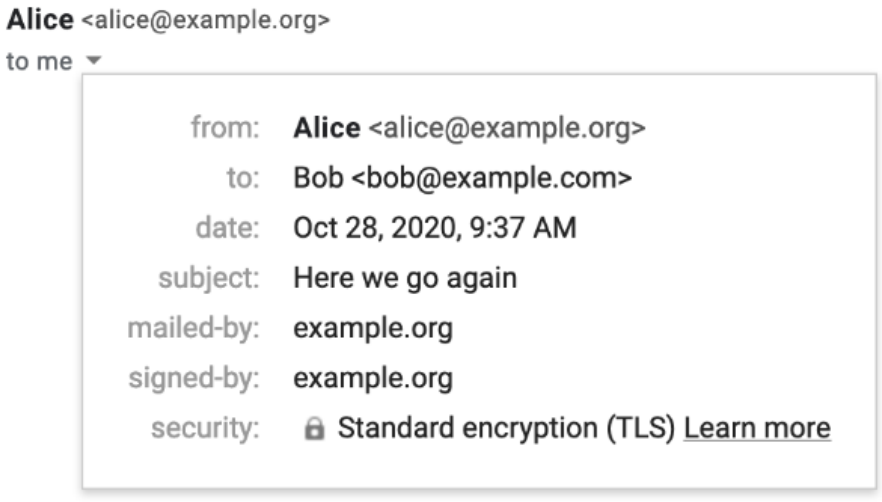


Рисунок 1.4 – mailed-by вказує на перевірку SPF, signed-by вказує на дійсність підпису DKIM. security вказує на те, що сервер вихідної пошти відправника використовував STARTTLS.

Окрім конфіденційності та цілісності, безпека інформації також пов’язана з доступністю сервісів. Оскільки повідомлення може бути відкинуто як спам або потрапити в папку спаму одержувача Більшість людей мінімізують цей ризик, не використовуючи власні системи електронної пошти.

Видання за іншого відправника називається спуфінгом. Хоча запобігання спуфінгу не усуне спам і фішинг сам по собі, оскільки спамери також можуть застосовувати такі стандарти, а фішинг залишається можливим із подібними доменами, це важлива передумова для інших методів, таких як позначення невідомих відправників. Як було з’ясовано раніше, спуфінг електронної пошти вирішується у два етапи: сервер вхідної пошти одержувача перевіряє, що інша сторона авторизована надсилати електронні листи від імені домену відправника, а сервер вихідної пошти цього домену гарантує, що локальна частина адреси «Від» належить користувачеві, який надіслав повідомлення.

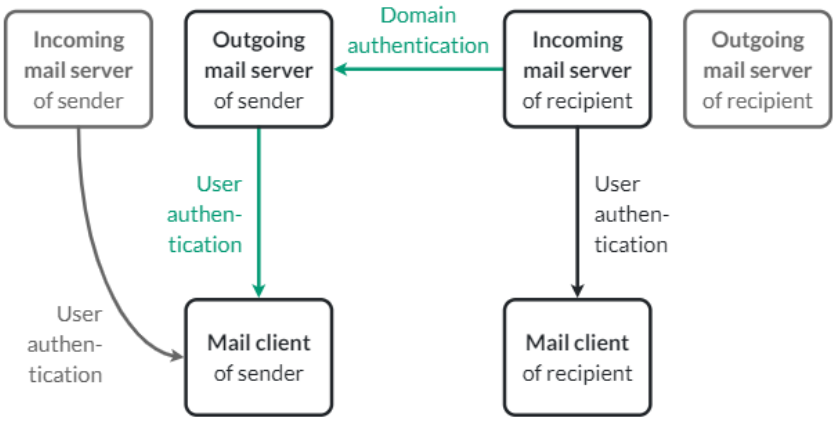


Рисунок 1.5 – Алгоритм аутентифікації домену

1.4 Аналіз програмних продуктів для відправки пошти

Запускати власний поштовий сервер є доволі непростим завданням. Тому потрібно аналізувати кожну деталь, щоб зробити правильний вибір. Перше, на що слід звернути увагу, це тип MTA: тільки посередник відправки або повноцінний поштовий сервер [11].

MTA посередники (Relay-only or send-only MTAs) можуть пересилати електронні листи на інший сервер, яким зазвичай є SMTP-сервер постачальника

послуг Інтернету (ISP). Найбільш використовувані MTA посередники – nullmailer, msmtp і sSMTP. Цей тип агента передачі пошти є хорошим вибором, якщо є необхідність простого надсилання електронних листів на електронну адресу, наприклад Gmail [38].

Однак слід враховувати такі фактори:

– Чи може MTA зберегти електронні листи в чергу для подальшої доставки в разі збою?

– Чи може MTA замінити системного агента доставки пошти (MDA)? Якщо він може, то він оброблятиме всю пошту, що надходить із системи.

– Чи підтримує MTA вимоги для підключення до SMTP-сервера ISP? Ці вимоги можуть включати певну автентифікацію або TLS.

Повноцінні (Full-fledged) MTA, також відомі як поштові центри, можуть виконувати всі функції доставки пошти в Інтернеті. Як правило, цей тип агента пересилання пошти отримує електронні листи від MTA лише для ретрансляції, а потім пересилає їх. Postfix, Sendmail і Exim належать до цієї категорії. Щоб вибрати найкращий варіант, слід враховувати кілька критеріїв вибору: безпека, продуктивність, документація та життєздатність. Список може бути ще довшим залежно від цілей. На основі цих критеріїв можна глибше зануритися в деякі технічні аспекти, як-от:

– Чи використовується один або кілька файлів для конфігурацігурації
– Чи є синтаксис зрозумілим
– Яка вбудована скриптова мова використовується
– Стабільність коду
– Регулярність оновлень програмного забезпечення
– Мінімальний набір функцій
– Та інше…

Postfix є MTA, орієнтованим на безпеку, тоді як Sendmail є стандартним MTA для систем Unix, а Exim є настроюваним і одним із найбільш гнучких агентів передачі пошти з точки зору конфігурації [46].

Таблиця 1.2 – Порівняльна таблиця агентів передачі пошти

MTA	Postfix	Sendmail	Exim
Підтримка серверних ОС	Cross-platform	Cross-platform	Unix-like
Методи авторизації	PLAIN / LOGIN / CRAM-MD5 / DIGEST-MD5 / ANONYMOUS / EXTERNAL / G2 / GSSAPI / GSS-SPNEGO / KERBEROS_V4 / NTLM / OTP / PASSDSS / SCRAM / SRP	PLAIN / LOGIN / CRAM-MD5 / DIGEST-MD5 / ANONYMOUS / EXTERNAL / G2 / GSSAPI / GSS-SPNEGO / KERBEROS_V4 / NTLM / OTP / PASSDSS / SCRAM / SRP / X.509 PKI auth via STARTTLS and EXTERNAL	PLAIN / LOGIN / CRAM-MD5 / DIGEST-MD5 / ANONYMOUS / EXTERNAL / G2 / GSSAPI / GSS-SPNEGO / KERBEROS_V4 / NTLM / OTP / PASSDSS / SCRAM / SRP / SPA / Dovecot SASL / GNU SAS / Heimdal GSSAPI
Сірий список	+	+	+
SPF	Опціонально	Опціонально	Опціонально
DKIM	Опціонально	Опціонально	Опціонально
DMARC	Опціонально	Опціонально	Опціонально
Запобіжники спаму	+	+	+

Переваги програмного забезпечення Postfix:

- Спрямованість на безпеку
- Відмінна документація
- Висока швидкість роботи з чергування листів
- Сумісність з Sendmail
- Активний розвиток
- Просте налаштування (конфігурація відповідно до параметрів (правил), які мають бути визначені у конфігураційному файлі)

Основним недоліком Postfix є його невелика гнучкість в кастомізації. Postfix надає безліч параметрів і налаштувань, а також має орієнтовану на безпеку

архітектуру. Хоча багато мілтерів (поштових фільтрів) можна використовувати з Postfix із коробки, потрібно вміти налаштовувати логіку за їх допомогою.

Перевагами Sendmail є його гнучкість та портативність. Основним недоліком є його слабка орієнтованість на безпеку. SendMail вже перестав активно розвиватися, та його доля ринку з 1996 року зменшилась від 80% до 4% [10].

В свою чергу, Exim схожий на фреймворк з власною мовою застосунків для виконання складних конфігурацій. Він має вдосконалену багатоетапну логіку обробки пошти, яка підходить для складних випадків.

З усіх MTA, які ми розглядаємо, Postfix найбільше приділяє увагу безпеці. Ймовірна причина полягає в тому, що він був створений Wietse Zweitze Venema, фахівцем з безпеки вільного програмного забезпечення. Sendmail не можна вважати безпечним MTA. Крім того, Postfix спочатку був розроблений для пом'якшення вразливостей, пов'язаних із Sendmail. Exim досить безпечний для більшості випадків, але програє Postfix. Правильна конфігурація вашого MTA Postfix забезпечує покращений захист від спаму, зловживань і витоку конфіденційних даних [29].

На відміну від Exim, Postfix має центральний менеджер черги та краще та швидше обробляє черги. Деякі системні адміністратори стверджують, що він більш ефективний при дуже високому навантаженні, ніж Exim, але не до помітного ступеня [22].

Sendmail відомий своєю неефективністю порівняно з конкурентами. Системні адміністратори часто скаржаться на випадкові проблеми і зазвичай встановлюють Postfix або Exim. Обидва є заміною Sendmail і майже однакові з точки зору надійності. Однак Postfix на крок попереду завдяки своїй модульній архітектурі. Він складається з незалежних частин системи, які можуть бути взаємозамінними в разі поломки, що забезпечує більш високий рівень надійності.

Кожний з трійки має монолітний головний конфігураційний файл. У той же час Exim є найбільш конфігурованим. Postfix, незважаючи на свою орієнтацію на безпеку, забезпечує високий рівень гнучкості та простоти адміністрування. Для

адміністраторів-початківців Postfix буде легше налаштувати, ніж будь-який інший MTA. Крім того, він реалізує Sendmail CLI і сумісний із поштовими фільтрами Sendmail [20].

Однак Postfix є менш універсальнішим, ніж Exim. Це означає, що Exim має чудові засоби інтеграції та може дати системним адміністраторам те, що вони просять. Exim так чи інакше підтримує більшість можливостей MTA. Exim також поєднується з cPanel: однією з найпоширеніших панелей керування для власників доменів.

Sendmail програє Postfix і Exim у всіх аспектах, включаючи зручність використання. На практиці немає вагомих причин для встановлення цього MTA. У той же час Sendmail попередньо встановлено на більшості комерційних операційних систем Unix. Це також портативне рішення, яке можна використовувати в системах, відмінних від Unix, наприклад Windows [53].

Висновки до розділу 1

Отже, у цьому розділі було обґрунтовано актуальність теми роботи по розробці проекту підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ.

Було досліджено завдання та обрано вектор для продовження дослідження процесів повязаних з відправкою електронної пошти за допомогою сучасних протоколів та додатків.

2. ПРОЕКТУВАННЯ ПІДСИСТЕМИ ЗАХИСТУ СИСТЕМИ ЕЛЕКТРОННОЇ ПОШТИ ПІДПРИЄМТСВА.

2.1 Опис інфраструктури підприємства та визначення недоліків у наявній системі

Підприємство, на базі якого ведеться дослідження, з самого початку розвитку свого додатку, мало розподілену інфраструктуру, та додатки розміщалися на фізичних серверах в різноманітних датацентрах. З розвитком хмарних технологій, технологій оркестрації додатків та контейнеризації, підприємство почало переміщення своєї інфраструктури до контейнерів, які знаходяться на потужностях хмарних провайдерів.

Наразі, в рамках підприємства, не було впроваджено механізм для зберігання та обробки електронної пошти, що, в свою чергу, приводить до її втрати та потенційного перехоплення третьою стороною.

В даному випадку, втрата або передача інформації або вмісту листів є критичною, тому що в листах міститься інформація про здійснені замовлення та інша чутлива інформація.

Маємо ситуацію, при якій основні додатки вже перенесено до хмарного середовища, але листи відправляються, безпосередньо, отримувачам без належного захисту та мінімальних операцій з приховання вмісту листів. Основна проблема, звісно, це втрата листів в ланцюжку між застосунком та отримувачем. Основною задачею є розробка вбудованого в кластер рішення, яке поєднає в собі функції шифрування, затримці в черзі листів, та логування кількості відправлених та невідправлених листів.

Так як, додатків в інфраструктурі є велика кількість, аудит, скільки користувачі відправили листів, є також критичним фактором. В даний момент кінцевий користувач, який відправляє лише кілька листів, та користувач, який відправляє сотні тисяч листів, платять однаково, тому що, наразі, немає механізму контролю.

2.2 Вибір інфраструктури та хмарного провайдеру

Контейнер — це стандартна одиниця програмного забезпечення, яка упаковує код і всі його залежності. Вони беруть ваш монолітний код і перетворюють його на кілька легких модулів, якими ви можете легше керувати та з'єднувати їх між собою, не турбуючись про те, що один невеликий модуль зламає всю вашу програму. Це дає вам більш детальний контроль над вашим кодом, але це також означає, що у вас є кілька рухомих частин як частина вашої платформи.

Проблема такої кількості рухомих частин полягає в тому, що за ними важко стежити. Якщо у вас є один контейнер, який підключається до іншого, ви повинні пам'ятати про оновлення кожного з них, щоб забезпечити стабільність на вашій платформі. Помножте це на десятки контейнерів, і тепер у вас виникне проблема керування кодом [27].

Kubernetes вирішує велику частину накладних витрат на розгортання контейнерів. Періоди розгортання можуть тривати цілий день між компіляцією коду, його тестуванням і перевіркою оновлення всіх служб. Кілька інструментів автоматизації з'явилися протягом багатьох років, щоб допомогти розробити кращий спосіб розгортання, але ці інструменти в основному призначені для монолітних баз коду. Оркестрація та автоматизація вирішують багато проблем, які виникають під час ручних розгортань інфраструктури — ви не забуваєте файли, ви оновлюєте всі сервери, а зміни можна скасувати.

Ще одна відмінність Kubernetes від інших варіантів розгортання полягає в тому, що розгортання контейнерів відбувається безперервно. Немає необхідності чекати компіляції та подальшого розгортання двійкових файлів один за одним. Натомість Kubernetes завжди вносить нові зміни у контейнери та розгортає їх у фоновому режимі. Це спосіб швидкого розгортання коду без зупинки продуктивності в певний час місяця. Якщо існує служба, яка постійно потребує змін, то Kubernetes і контейнери можуть впоратися з навантаженням без втручання розробників [4].

Підсумок, Kubernetes і контейнери загалом природно підходять для хмарних середовищ, оскільки контейнери набагато більш портативні та легкі – і контейнери можуть працювати в більшості хмарних і локальних середовищ. Контейнери, зокрема, є способом реалізації багатохмарної стратегії, оскільки їх можна використовувати як локально, так і серед основних хмарних провайдерів. Ця перехресна сумісність робить контейнери привабливим варіантом для зниження ризику впровадження мікросервісів у хмарі [9].

Проведемо аналіз переваг та недоліків Amazon Web Services (AWS). З трьох найкращих сервісів (AWS, Azure, GCP) AWS є лідером ринку хмарного хостингу, тому його розгляд є першочерговим. AWS має три контейнерні середовища: ECS, EKS і Fargate. ECS — найкращий варіант, якщо у розробників мало досвіду роботи з контейнерами та вже є використання AWS для розміщення своїх служб. Розгортання в ECS називається «контейнери як послуга» (containers-as-a-service), і це вважається хорошою відправною точкою для визначення того, чи підходять контейнери для використання в межах організації. Головним є дозвіл службі автоматизувати необхіднірозгортання безпосередньо в хмарі за допомогою Amazon AWS CloudFormation.

Для більш повного використання Kubernetes і контейнерів можливій вибір сервісу Amazon EKS. Якщо розміщення Kubernetes вже є локальним, EKS перемістить існуюче середовище в хмару. Аналіз переваг та недоліків показав, що EKS має хороші можливості, щоб стати найпопулярнішим способом керування контейнерами. Шістдесят три відсотки користувачів контейнерів, опитаних Kubernetes, використовують AWS.

AWS Fargate — новий продукт від Amazon. Це останній випуск від Amazon для користувачів контейнерів. Fargate дозволяє розгортати контейнери без керування серверами чи кластерами. AWS підтвердила, що Fargate також працюватиме з AWS EKS, тому буде кілька варіантів і комбінацій на основі індивідуальних потреб [48].

Розміщений Kubernetes з AWS — це те, що робить його привабливим для розробників, які тільки вивчають контейнерне середовище. Якщо є необхідність експериментальних розробок з контейнерами, і немає впевненості, що вони підходять у середовище розробки, використання AWS і їхнього сервісу Kubernetes стане хорошою відправною точкою. Недоліком є те, що EKS вважається складним у налаштуванні та вимагає певного технічного досвіду роботи з контейнерами, але це також повністю масштабоване та конфігуроване рішення, яке дозволяє компанії контролювати Kubernetes і те, як він працює з локальною розробкою [49].

Проведемо аналіз переваг та недоліків Microsoft Azure. Для системних адміністраторів, які переважно працюють з програмним середовищем Windows, розгортання в Azure є найбільш інтуєтивним та зрозумілим. Це найновіший контейнерний сервіс на хмарному ринку, доступний лише з 2015 року, тому Microsoft продовжує вдосконалювати свій сервіс.

Microsoft є компанією розробником операційної системи Windows, але на хмарній платформі Azure є також можливість роботи з різними дистрибутивами операційної системи Linux та Unix.

Це означає, що немає обмеження лише Windows, але існує є обмеження щодо гібридних контейнерів. Там, де AWS підтримуватиме гібридне розгортання, Azure обмежується лише Linux або Windows.

У жовтні 2017 року Azure випустила AKS (Azure Kubernetes Service), схожу на AWS EKS. AKS легко розгортається на віртуальній машині Azure, але найбільшою перевагою є те, що AKS є безкоштовним. Плата відбувається лише за ресурси віртуальних машин, які будуть використовуватися в Azure.

Недоліком Azure є те, що, хоча служба AKS фактично з’явилася перед AWS EKS, Kubernetes більш адаптований на AWS і GCP, і Azure зазвичай відстає від як AWS, так і Google Kubernetes Engine (GKE), в оновленні до останніх версій Kubernetes. Однак, якщо у використанні в наявній архітектурі наявний Azure і є необхідність працювати над впровадженням контейнерів, Azure спрощує розгортання та надає детальну аналітику, за допомогою якої є можливість

визначити, чи ця платформа задовольняє всі потреби. Однак слід зазначити, що Azure також має конкурентну послугу для AWS ECS під назвою Service Fabric, на яку також варто звернути увагу [41].

Проведемо аналіз переваг та недоліків Google Cloud Platform (GCP). Google є оригінальним творцем стандартів Kubernetes, тому робота з цією платформою дає перевагу майже в усіх аспектах. Будь-які нові версії та розгортання одразу доступні, тоді як інші платформи мають часове відставання. Google надає чудові можливості для роботи з технологіями великих даних, машинного навчання та штучного інтелекту (ШІ).

Основна проблема GCP полягає в тому, що він не є популярним рішенням для IaaS. Він не має хмарних пропозицій для малого бізнесу, які пропонують AWS і Azure, тому його платформа в цілому не приваблива для корпорацій, які хочуть інтегрувати хмару у свою внутрішню мережу. Немає інтеграції Active Directory, як наприклад в Azure або IAM в AWS.

Google просуває свою платформу як кращий спосіб для роботи за методологією DevOps. Відділи DevOps — це відділи, які складаються з спеціалістів, які працюють одночасно з задачами автоматизації розгортання та розробки застосунків. Розробники витрачають свій час на пошук кращих способів керування операціями. Автоматизування розгортань в команді DevOps, GCP має переваги [40].

Порівняння вартості є складним через цінову політику «оплата по мірі використання». Окремий розробник, який експериментує з Kubernetes і контейнерами, не заплатить стільки ж, скільки підприємство, якому потрібні потужні обчислювальні ресурси. Іншим фактором є те, що витрати залежать від ресурсів, які використовуються, і кожна платформа має мінімальну суму, виділену для кластера.

Отже, немає чіткої відповіді який сервіс найкраще використовувати з Kubernetes, тому що зрештою все зводиться до того, яка платформа підходить конкретному проекту. Якщо наявний досвід роботи з Azure або AWS, краще

використовувати ту саму платформу. Однак, якщо є потреба в роботі з штучним інтелектом та машинним навчанням, Google має привабливу пропозицію, яка коштує дешевше [39].

Таблиця 2.1 – Порівняльна таблиця переваг та недоліків хмарних провайдерів послуг

Провайдер	Плюси	Мінуси
AWS	Безпека, надійність і масштабованість	Потенційно дорожче, ніж інші варіанти
	Легковісні служби, щоб тестування Kubernetes і контейнерів, перш ніж перемістити локальне середовище в хмару	Важкий в використання новим розробникам, які не знайомі з контейнерними службами та Kubernetes
	Fargate дає розробникам можливість розгортати контейнери без розуміння серверної інфраструктури	
Azure	Більш інтуїтивно зрозумілий для розробників Windows	Повільніше працює під час розгортань
	Підтримує контейнери Windows і Linux	Не підтримує гібридні контейнери
	AKS є безкоштовним. Ви платите лише за ресурси віртуальних машин.	
GCP	Компанія розробник Kubernetes, тому впровадження нових функцій відбувається швидше	Не інтегрується з IaaS
	Ідеально підходить для розробників, які хочуть працювати зі ШІ та великими обсягами даних	

2.3 Вибір серверу авторизації пошти

Кожна організація має свої унікальні вимоги до розсилання електронної пошти. Хоча деяким компаніям потрібне рішення для надсилання простих

маркетингових інформаційних бюлетенів, іншим потрібна розширена інфраструктура електронної пошти для надсилання великих обсягів і транзакцій.

Далі буде розглянуто технічну сторону двох найбільших платформ на ринку електронної пошти: SendGrid і Mailgun. Обидва інструменти пропонують чудове рішення для надсилання великих обсягів, транзакцій і найкращої на ринку якості доставки.

Mailgun позиціонує себе як «Сервіс електронної пошти для розробників», пропонуючи потужні API, які дають змогу легко надсилати, отримувати та відстежувати електронні листи. Mailgun має статистику безперебійної роботи на 99,99% за угодою про рівень обслуговування та надає свої послуги електронної пошти понад 225 000 компаніям, пропонуючи потужний API, Mailjet для маркетингу, перевірки електронної пошти, гуртового надсилання, прогнозування проблем із доставкою тощо [19].

Проведемо аналіз переваг та недоліків SendGrid. SendGrid є одним із найкращих інструментів електронної пошти для транзакцій на ринку, SendGrid також пропонує функції електронного маркетингу, включаючи прості у використанні шаблони дизайну та форми реєстрації. SendGrid є лідером у надійній доставці електронної пошти для понад 80 000 клієнтів, які використовують їхню службу для надсилання понад 70 мільярдів електронних листів щомісяця. Комунікаційна платформа Twilio придбала SendGrid у 2018 році.

Коли відбувається надсилання транзакційних електронних листів, особливо в масштабі, дуже важливо мати інструмент електронної пошти з високою швидкістю доставки. Неможливо дозволяти, щоб електронні листи про покупки потрапляли у спам або були доставлені через значні проміжки часу [21].

SendGrid проклав шлях до транзакційної електронної пошти з моменту заснування компанії в 2009 році. SendGrid вирізняється своєю доступністю, масштабованістю та глибоким досвідом роботи з електронною поштою. SendGrid надає інформацію в своїх звітах про те, що клієнти, які користуються їхніми

експертними послугами, мають середній рівень доставки 97% порівняно із середнім показником по галузі 85%.

SendGrid пропонує розширену автентифікацію за допомогою SPF і DKIM для запобігання фішингу та спуфінгу, а також функції штучного інтелекту. Це включає в себе ACE, їх адаптивний комунікаційний механізм, який стає закодованим штучним інтелектом на їхній платформі, щоб підвищити доступність і адаптуватися до мінливих правил ISP.

Величезний масштаб SendGrid (вони надсилають понад 70 мільярдів електронних листів на місяць) генерує великі дані, необхідні для машинного навчання [1].

Mailgun також пропонує послугу керованої доставки з підтримкою API, керування репутацією IP-адрес та доменів, постійних консультацій і проактивного моніторингу електронної пошти.

Компанія Mailgun надає інформацію, що їй середня успішність доставки становить 97,4%. Також згадують, що їхні електронні листи мають середній показник відмов 0,42%, що значно нижче середнього показника по галузі (2%).

SendGrid має широкий спектр партнерів на ринку, включаючи Simon для кращої обробки даних і Ongage для вдосконаленого інтерфейсу для керування операціями з електронною поштою.

Mailgun має більш обмежену кількість інтеграцій, хоча він має міцні партнерські зв’язки з такими платформами, як Customer.io для автоматизованого обміну повідомленнями клієнтів і Omnisend для автоматизованих робочих процесів електронної комерції [3].

2.4 Дослідження механізмів захисту електронної пошти

Захист електронної пошти допомагає захистити організації та одержувачів від витоку даних та інших загроз.

Використання безпечних методів автентифікації входу. Використання простої пари паролю та логіну є значно застарілою практикою, але багато людей

досі використовують цей метод. Сучасні методи автентифікації два або більше факторів для підтвердження особистості користувача.

Двофакторна автентифікація (2FA) вимагає від користувача поєднання 2 типів факторів автентифікації. Ці фактори поділяються на три категорії:

– Фактори знань (те, що ви знаєте, як-от PIN-код або пароль)

– Фактори володіння (те, що у вас є, як дебетова картка чи мобільний телефон)

– Фактори фізичної унікальності людини (щось властиве вам, як-от відбиток пальця)

Наприклад, у банкоматі вимагається пред'являти дебетову картку (фактор володіння) і вводити PIN-код (фактор знання). Цей тип автентифікації часто відбувається за допомогою SMS (тексту) або тимчасових одноразових паролів.

Система єдиного входу (SSO) дозволяє користувачам отримувати доступ до кількох програм після автентифікації своєї особи через постачальника ідентифікаційної інформації. Часто SSO поєднується з 2FA.

Цей тип централізованого доступу допомагає застосовувати безпечні паролі, спрощуючи вхід для користувача, а також зменшує ризик компрометації облікового запису. Система єдиного входу також полегшує командам безпечне зберігання та обмін логінами для таких програм, як ESP [34].

Application programming interface (API) — унікальні коди, які ідентифікують і автентифікують користувача — є більш безпечною альтернативою входу за допомогою імені користувача та пароля. Розробники зазвичай використовують цей метод для контролю доступу до API, наприклад, API електронної пошти.

При налаштування ключів API, є можливість вибору різних рівнів або дозволів для кожного користувача, що дозволяє обмежувати доступ до різних частин облікового запису.

Налаштування протоколів автентифікації відправника. Протоколи автентифікації відправника показують провайдерам послуг вхідних повідомлень (ISP), що відправник є справжнім відправником, а не спамером або спуфером. Крім

того, ці протоколи ускладнюють зловмисникам видавати себе за ваш бренд або використовувати репутацію вашого відправника для здійснення фішингових атак.

З іншого боку, існує ризик бути вразливим перед спамерами або імітаторами, якщо немає використання цих протоколів,. Це може завдати шкоди репутації компанії або навіть призвести до того, що компанія опинитесь у списку блокувань [28].

Відправники повинні використовувати такі протоколи автентифікації:

– Sender Policy Framework (SPF), яка допомагає ідентифікувати поштові сервери, яким дозволено надсилати електронні листи з певного домену.

– DomainKeys Identified Mail (DKIM), який перевіряє, що відправник електронної пошти несе відповідальність за вміст електронної пошти та домен, який її надіслав.

– Domain-based Message Authentication, Reporting & Conformance (DMARC), яка дозволяє відправникам вказувати, як обробляти електронні листи, не автентифіковані за допомогою 2 методів вище

Докладніше ці протоколи буде досліджено в розділах 2.4.1 – 2.4.3.

Використання SMTP серверу. Простий протокол передачі пошти (SMTP) — це, по суті, конвеєр, який переносить електронні листи з сервера до папки «Вхідні» одержувача. Сервер SMTP обробляє електронну пошту, вирішує, на який сервер їй потрібно перейти, і надсилає її на цей сервер. Тоді провайдер, який отримує повідомлення, завантажує це повідомлення з сервера та розміщує його в папці «Вхідні» одержувача [6].

Окрім ретрансляції вашої електронної пошти до місця призначення, SMTP-сервер перевіряє, що електронна пошта надійшла з активного облікового запису, виступаючи першою лінією захисту, захищаючи одержувачів від нелегітимних відправників.

Автентифікація SMTP допомагає відправнику, захистити вихідну електронну пошту, запросивши входу за допомогою механізму автентифікації, який підтримується сервером.

Захист даних за допомогою шифрування. Шифрування є одним із найефективніших способів захисту вмісту електронної пошти. Це особливо важливо, якщо електронні листи містять конфіденційну інформацію про одержувача.

Transport Layer Security (TLS), тип шифрування, захищає дані під час їх переміщення з вашого сервера на сервер постачальника електронної пошти. За замовчуванням Twilio SendGrid намагатється доставити електронні листи через з'єднання, зашифроване TLS, якщо сервер електронної пошти одержувача це підтримує. Це запобігає перегляду вмісту електронної пошти зловмисникам, які використовують пасивні пристрої спостереження [7].

Обмеження доступу до даних. Доступ до даних, особливо даних одержувачів, має бути обмежений працівниками, яким він дійсно потрібен. Це зменшує ймовірність того, що витік електронних листів стане більш серйозною проблемою, якщо дані будуть скомпрометовані. Для цього є кілька способів:

– Обмежування кількості користувачів лише тими співробітниками, яким регулярно потрібен доступ до вашого ESP. Регулярна перевірка кількості користувачів, щоб переконатися, що в списках немає неактивних користувачів.

– Налаштування різних рівнів дозволів залежно від ролі кожного користувача, дозволяючи кожному члену команди отримувати доступ до функцій, необхідних для виконання своїх функцій, одночасно захищаючи конфіденційні дані клієнтів [18].

2.4.1 Sender Policy Framework

SPF розшифровується як Sender Policy Framework. Це метод автентифікації електронної пошти, який допомагає ідентифікувати поштові сервери, яким дозволено надсилати електронні листи з певного домену. Використовуючи цей протокол перевірки, провайдери можуть визначити, коли спуфери та фішери намагаються підробити електронні листи з домену, який їм не належить, щоб надіслати зловмисну електронну пошту користувачам.

Завдяки SPF одержувачі можуть бути впевнені, що повідомлення електронної пошти, які вони отримують, надходять від того, кого вони очікують. І відправники можуть бути спокійні, знаючи, що фішери не підманюють електронну пошту та не фішингують свою аудиторію через свій бренд.

Запис SPF — це короткий рядок тексту, який адміністратор домену додає до свого запису txt. Запис txt зберігається в DNS (системі доменних імен) разом із записами A, PTR і MX [37].

“v=spf1 ip4:12.34.56.78 include:example.com -all”

Рисунок 2.1 – Вигляд SPF запису

Запис SPF зазвичай перевіряється на дуже ранньому етапі розмови SMTP, задовго до того, як буде передано текст повідомлення. Коли робиться спроба надіслати повідомлення, між відправником і сервером-одержувачем відкривається TCP-з'єднання.

Після встановлення з’єднання видається команда HELO, яка, по суті, повідомляє серверу-одержувачу, який домен намагається надіслати йому пошту. Далі слідує команда MAIL FROM, яка повідомляє серверу-одержувачу, з якої електронної адреси надійшло повідомлення. Домен, знайдений у команді MAIL FROM (також відомий як конверт із і шлях повернення), є доменом, який використовується для перевірки запису SPF [44].

Отже, припустімо, що було отримано повідомлення, а адреса MAIL FROM — bohdan@example.com. Сервер-одержувач перевірить загальнодоступні записи DNS для example.com і шукатиме запис TXT, який починається з v=spf1. Якщо немає запису TXT, який починається з v=spf1, автентифікація пройде. Якщо існує більше ніж один запис TXT, який починається з v=spf1, може виникнути помилка.

Тепер сервер-одержувач перевірить, чи IP-адреса клієнта SMTP, який намагається надіслати повідомлення, включена до запису SPF. Якщо IP-адреса вказана, то повідомлення пройде автентифікацію SPF.

Кожен запис SPF дещо відрізнятиметься, тому слід робити перевірку правильності налаштування SPF. Інструменти, які використовуються для перевірки SPF записів:

– Scott Kitterman’s SPF Testing Tools
– OpenSPF.org
– SPF Record Check
– SPF Wizard

2.4.2 DomainKeys Identified Mail

DKIM (DomainKeys Identified Mail) — це криптографічна технологія, створена Cisco та Yahoo, яку відправники можуть використовувати для «підпису» своїх повідомлень. DKIM дозволяє одержувачу електронного повідомлення перевірити, чи це повідомлення було авторизовано та надіслано відправником, відповідальним за домен. Якщо повідомлення не підписані за допомогою DKIM, постачальники вхідних повідомлень, такі як Gmail і Microsoft, можуть блокувати повідомлення та перешкоджати їх доставці одержувачам. [45]

DKIM є відносно простою формою автентифікації електронної пошти, оскільки її єдиною функцією є перевірка того, що відправник електронного листа несе відповідальність за домен, з якого його надіслано, і він несе відповідальність за вміст електронного листа. Два кроки роботи DKIM:

– Відправник додає закритий ключ на своїх поштових серверах і підписує повідомлення.

– Сервер-одержувач перевіряє відкритий ключ, що зберігається в текстовому записі dkimselector._domainkey.domain.com, щоб перевірити закритий ключ, доданий відправником. [8]

DKIM важливий, оскільки це один із способів, за допомогою якого постачальники вхідних повідомлень можуть перевірити ідентичність відправника. Без правильної реалізації DKIM багато постачальників папок "Вхідні" блокуватимуть електронну пошту, не даючи повідомленням дістатися за

призначенням. Хоча це може здатися не дуже важливим, якщо заблоковано лише невелику кількість повідомлень, це може мати великі наслідки для бізнесу.

В Інтернеті доступні різноманітні інструменти тестування DKIM. Використання чогось на зразок аналізатора DKIM або засобу перевірки DKIM допоможе визначити, чи правильно ви опублікували свій запис DKIM. Загалом необхідно перевіряти будь-які зміни, які вносяться до записів SPF або DKIM, перш ніж їх застосовувати.

SPF дозволяє відправникам повідомляти провайдерам, які IP-адреси можуть надсилати від їх імені. DKIM дозволяє Інтернет-провайдерам перевірити, чи надісланий вміст відповідає призначенню початкового відправника. [9]

2.4.3 Domain-based Message Authentication, Reporting & Conformance

Автентифікація повідомлень на основі домену, звітування та відповідність, або DMARC, — це протокол, який використовує структуру політики відправника (SPF) і ідентифіковану пошту DomainKeys (DKIM) для визначення автентичності повідомлення електронної пошти.

Записи DMARC спрощують для постачальників послуг Інтернету (ISP) запобігання зловмисним методам електронної пошти, таким як підробка домену з метою фішингу особистої інформації одержувачів.

По суті, це дозволяє відправникам електронної пошти вказувати, як обробляти електронні листи, які не були автентифіковані за допомогою SPF або DKIM. Відправники можуть відправити ці електронні листи в папку небажаних повідомлень або заблокувати їх усі разом. Завдяки цьому провайдери можуть краще виявляти спамерів і запобігати вторгненню зловмисної електронної пошти в скриньки споживачів, мінімізуючи помилкові спрацьовування та надаючи кращі звіти про автентифікацію для більшої прозорості на ринку. [52]

Є кілька ключових причин необхідності впровадження DMARC:

Репутація. Публікація запису DMARC захищає ваш бренд, запобігаючи неавтентифікованим сторонам надсилати пошту з домену, який їм не належить. У

деяких випадках проста публікація запису DMARC може призвести до покращення репутації.

Видимість. Звіти DMARC покращують видимість вашої програми електронної пошти, повідомляючи, хто надсилає електронну пошту з вашого домену.

Безпека. DMARC допомагає спільноті електронної пошти встановити узгоджену політику роботи з повідомленнями, які не пройшли автентифікацію. Це допомагає екосистемі електронної пошти в цілому стати безпечнішою та надійнішою.

v=DMARC1\;p=none\;rua=mailto:dmarc@sendgrid.com\;ruf=mailto:dmarc@sendgrid.com\;rf=afrf\;pct=100

Рисунок 2.2 – Вигляд DMARC запису

Записи DMARC є важливим продовженням автентифікації електронної пошти. Це просто ще один приклад спільної роботи відправників електронної пошти та провайдерів, щоб захистити канал електронної пошти. [31]

2.4.4 Transport Layer Security

Transport Layer Security (TLS) — це протокол, який забезпечує безпеку цифрового зв’язку між двома сторонами.

Коли сервер і клієнт обмінюються даними, добре налаштований TLS гарантує, що жодна третя сторона не зможе підслухати чи підробити будь-яке повідомлення. Однак налаштувати TLS може бути складно, а якщо його налаштувати неправильно, це може призвести до помилкового відчуття безпеки.

Попередником протоколу TLS був протокол Secure Sockets Layer (SSL), усі версії якого офіційно застаріли, вважаються небезпечними та не повинні використовуватися. Крім того, TLS версії 1.1 і 1.0 були офіційно визнані застарілими IETF у RFC 8996 і не повинні використовуватися. NCSC рекомендує оновити державні системи, які використовують TLS версії 1.1 або 1.0, до TLS версії 1.3 або 1.2. [30]

На момент написання, останньою версією TLS є 1.3, яка розроблена, щоб бути більш безпечною, ніж попередні ітерації.

TLS надає три основні служби, які допомагають забезпечити безпеку даних:

– Аутентифікація. Автентифікація дозволяє кожній стороні зв’язку підтвердити, що інша сторона є тим, за кого себе видає.

– Шифрування. Дані шифруються під час передачі між агентом користувача та сервером, щоб запобігти їх читанню та інтерпретації неавторизованими сторонами.

– Цілісність. TLS гарантує, що між шифруванням, передачею та дешифруванням даних жодна інформація не буде втрачена, пошкоджена, підроблена чи фальсифікована.

З’єднання TLS починається з фази рукостискання, коли клієнт і сервер узгоджують спільний секрет і обговорюють важливі параметри, такі як набори шифрів. [20]

Висновки до розділу 2

Отже, в цьому розділі було здійснено аналіз сучасного стану розвитку технологій відправки електронної пошти (проект підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ).

Було здійснено порівняння хмарної інфраструктури, серверів авторизації почтових провайдерів. Було досліджено сучасні протоколи захисту відправки електронної пошти, а саме: SPF, DKIM, DMARC, TLS.

2. РОЗРОБКА ФУНКЦІОНАЛЬНОЇ МОДЕЛІ ТА ЗАСТОСУНКУ ПІДСИСТЕМИ ЗАХИСТУ СИСТЕМИ ЕЛЕКТРОННОЇ ПОШТИ ПІДПРИЄМСТВА.

3.1 Розробка функціональної моделі

3.1.1 Побудова принципових схем роботи додатків.

Згідно з дослідженням вимог до проекту підсистеми захисту, маємо проблему з доставкою електронної пошти.

Рисунок 3.1 – Ілюстрація ситуації з місцем втрати листів

Було визначено, що листи можуть не дістатися до кінцевого отримувача через наступні причини:

– SendGrid може стати недоступним

– API ключ SendGrid може закінчитися, або бути неправильно налаштованим

– Неправильне налаштування обмежень за IP адресами.

Саме через це було вирішено, додати в цю схему поштового посередника, який зробить поштову чергу, та буде запобігати втрати пошти.

Рисунок 3.2 – Спрощена схема нової конфігурації інфраструктури

Так як, маємо багато кластерне середовище, потрібно подбати про роботу саме за таким сценарієм.

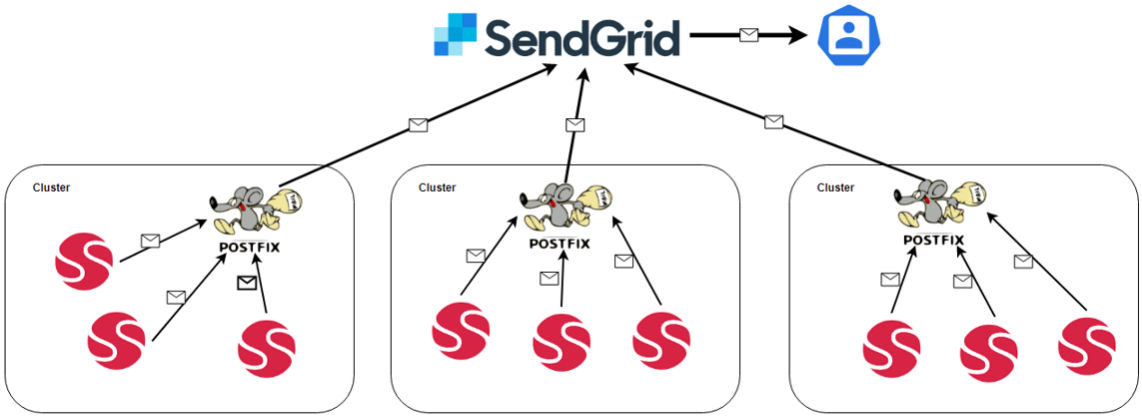


Рисунок 3.3 – Спрощена схема роботи поштової черги в багатокластерному середовищі

Отже, в кожному кластері знаходиться по два дублюючих додатки Postfix, які працюють синхронно та на різних подах. На кожному кластері Postfix матиме окремі налаштування, та свій SendGrid API ключ.

3.1.2 Розробка детальних схем роботи додатків.

Після дослідження наявних програмних засобів, було обрано наступні ресурси:

– кластер в Azure (AKS)

– postfix в ролі mail transfer agent

– fluent-bit для збору логів з інших контейнерів

– Grafana-loki-SQL база даних для обробки логування

– Azure managed disks для постійної памяті для зберігання поштової черги, навіть коли з деплойментом щось станеться.

Розробити систему, яка буде зберігати пошту, яку відправляє додаток, перевіряти чи відправка пошти можлива та лише тоді робити відправку пошти. Також потрібно розробити систему моніторингу кількості відправлених листів.

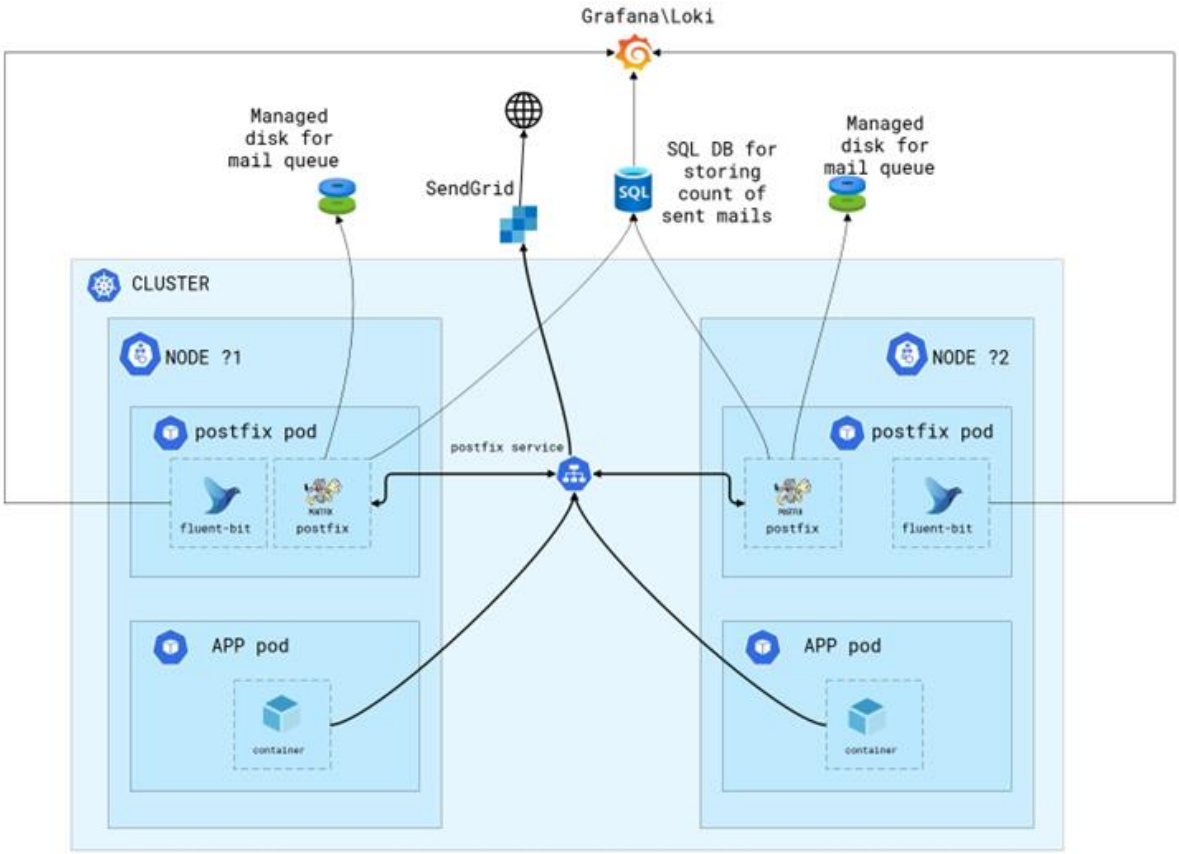


Рисунок 3.4 – Детальна схема роботи додатків у кластері за визначеними раніше вимогами

Розробляємо конфігураційні файли Helm для розгортання інфраструктури в кластерних середовищах.

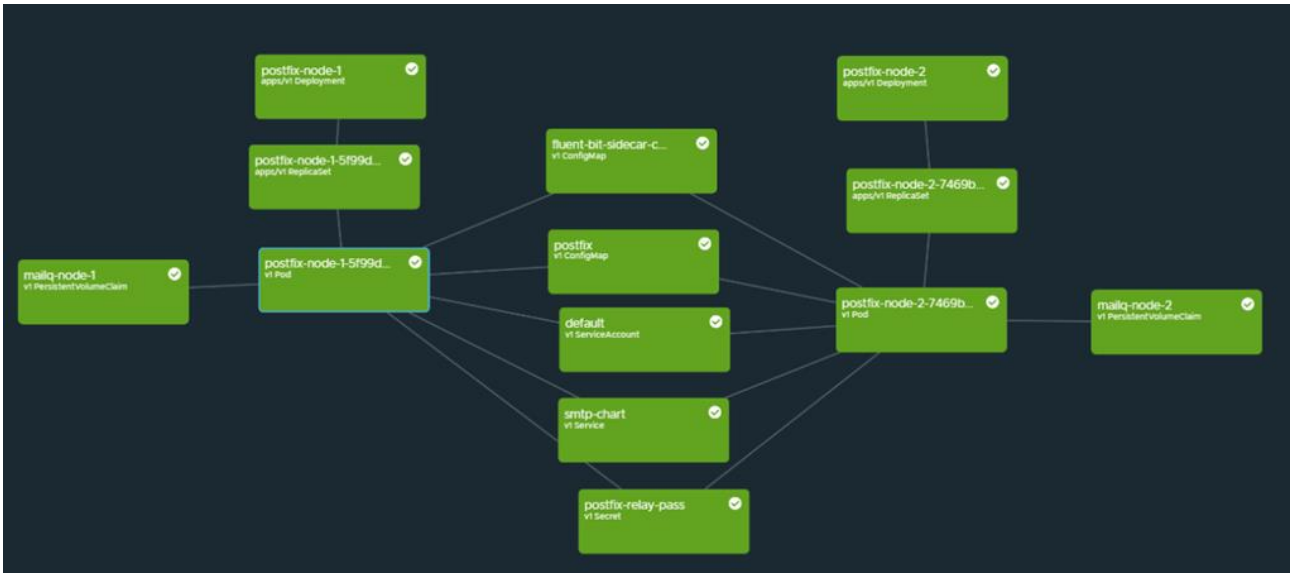


Рисунок 3.5 – Структура Helm чарту

Отже, в одному кластері маємо:

– Два поди Postfix, які включають в себе два контейнери з Fluent-bit та Postfix.
– Два PersistentVolumeClaim
– Два ReplicaSet
– Один ConfigMap для FluentBit
– Один ConfigMap для Postfix
– Один ServiceAccount
– Один Service
– Один Secret

3.1.3 Побудова діаграми послідовності дій

TLS – це протокол шифрування, призначений для захисту Інтернет-зв'язку. Рукостискання TLS – це процес, який розпочинає сеанс зв'язку з використанням шифрування TLS. Під час рукостискання TLS дві сторони, що спілкуються, обмінюються повідомленнями, щоб підтвердити одна одну, перевірити одна одну, встановити алгоритми шифрування, які вони використовуватимуть, і узгодити ключі сеансу. Рукостискання TLS є основною частиною роботи HTTPS.

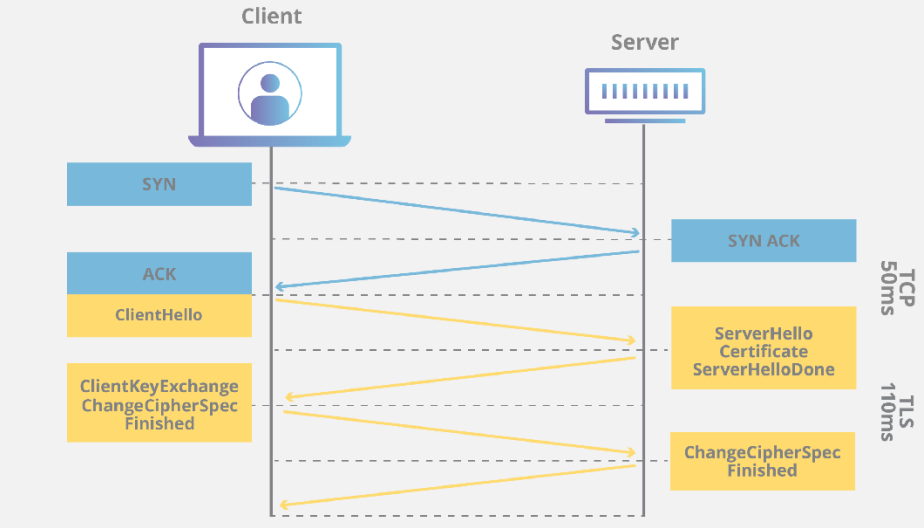


Рисунок 3.6 – Структура рукостискання TLS між Postfix та SendGrid

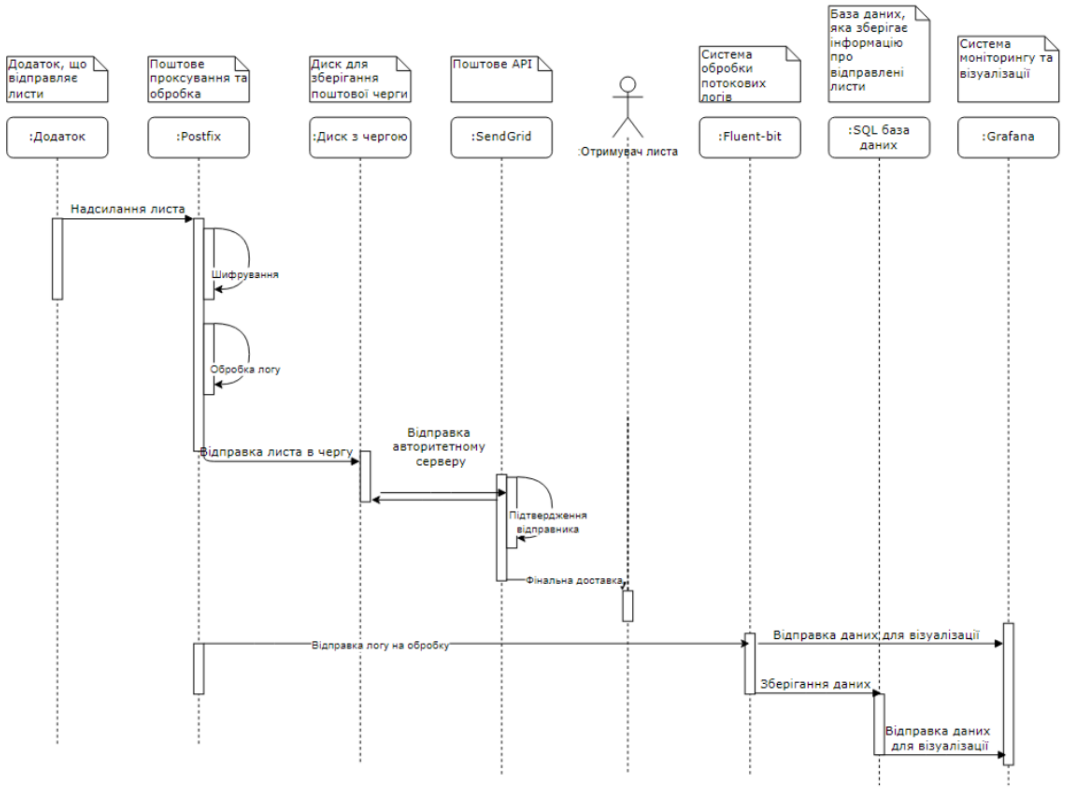


Рисунок 3.7 – Загальний вигляд діаграми послідовності проекту підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ

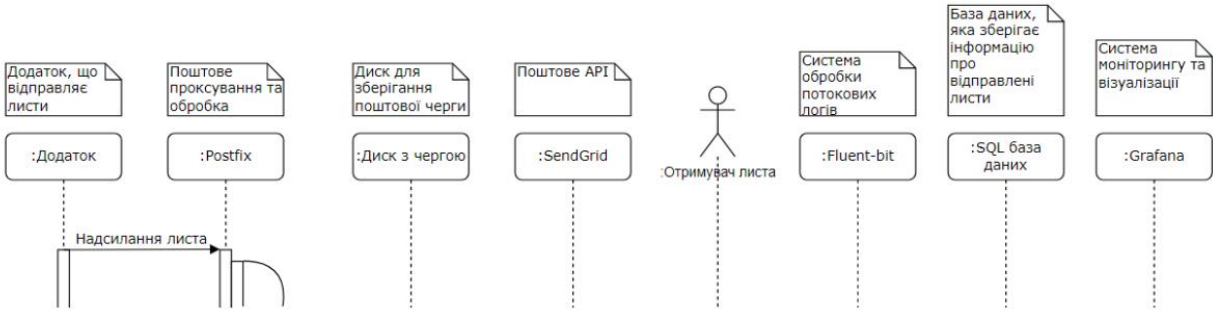


Рисунок 3.8 – Опис елементів діаграми

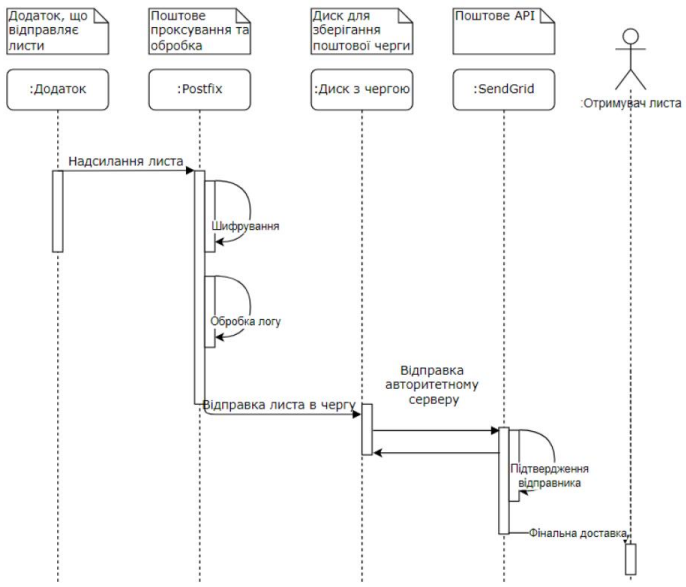


Рисунок 3.9 – Детальніший огляд процесу відправки листів

3.2 Розробка та контейнеризація застосунку

3.2.1 Розробка парсеру.

Основою поштової системи для проекту було обрано застосунок Postfix. Основною проблемою Postfix, у нашому випадку, є те, що його розробка починалась давно та не була розрахована на сучасну інфраструктуру інтернет технологій. Postfix показав себе простою та надійною системою для відправки та утворення поштової черги, але він не був розроблений для контейнерних середовищ, та має систему логів, з якою досить важко працювати. Для оптимальної роботи з записами логів та, взагалі, з моніторингом системи, було вирішено створити застосунок для парсингу та перетворення логів [2].

Postfix підтримує власну систему журналювання як альтернативу syslog (яка залишається типовою). Це доступно з Postfix версії 3.4 або новішої. Реєстрація в stdout корисна, коли Postfix працює в контейнері, оскільки це усуває залежність syslogd[6].

Розробку рішення було прийнято здійснювати за допомогою мовb програмування високого рівня загального призначення – Python. Його філософія

дизайну наголошує на зручності читання коду з використанням значних відступів[11].

За замовчуванням Postfix формує логи у важкому для парсингу вигляді:

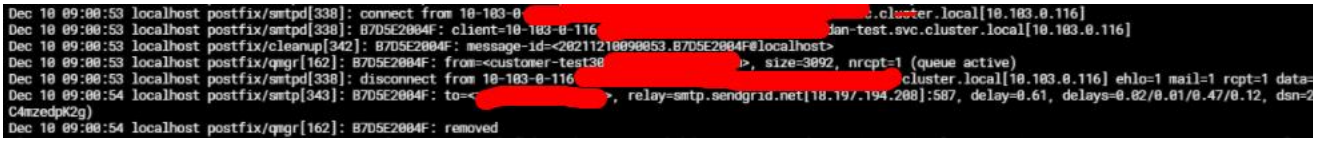


Рисунок 3.10 – Стандартний вигляд логів Postfix

На рисунку 1 результат успішної відправки листа за допомогою Postfix. Інформація про успішне надсилання листа було поділено на сім стрічок логу[22].

Основна ціль парсеру це зчитати ці 7 стрічок логу для подальшого використання та надсилання інформації до лог системи та бази даних про кількість відправленої пошти кожним користувачем[1].

Для цього необхідно, щоб парсер безперервно зчитував лог повідомлення із стандартного виводу та записував інформацію. У кожного листа є свій унікальний внутрішній ідентифікаційний номер, це означає, що за допомогою нього можливо робити з семи стрічок логів один, а далі вже працювати з отриманою інформацією[34].

Отже, скрипт з циклів while постійно вичитує вміст файлу з логами Postfix, який лежить за стандартним шляхом /var/log/maillog. Для виділення необхідної інформації використовуються регулярні вирази, які знаходяться в бібліотеці re[24].

```
82  patternLongSvc = re.compile(
83      "client=(.+)\.(?P<svc>.+)\.(.+)\.(.+)\.(.+)\.(.+)\[(?P<podIP>.+)\]")
84  patternUnknownSvc = re.compile("client=(?P<svc>unknown)\[(?P<podIP>.+)\]")
85  sessionIdPattern = re.compile("[A-F,0-9]{5,12}")
86  customerIdPattern = re.compile("\w+-(?P<customerId>\w+)-.+")
87  svcPattern = re.compile("client=.+")
88  fromPattern = re.compile("from=<.+>")
89  toPattern = re.compile("to=<.+>")
90  trashFilter=re.compile(".monitoring.svc.cluster.local")
```

Рисунок 3.11 – Використані регулярні вирази

Для зчитування файлу було використано бібліотеку subprocess, яка дозволяє в Python коді виконувати запуск системних додатків. В даному випадку

використовується стандартний Linux застосунок tail з прапорцем f, що виводить на екран останній рядок у файлі[19].

```
93    f = subprocess.Popen(['tail', '-F', filename],
94                         stdout=subprocess.PIPE, stderr=subprocess.PIPE)
```

Рисунок 3.12 – Запуск застосунку tail з коду

Як тільки парсер отримує рядок, який відповідає заданим параметрам та містить MailID у правильному форматі, парсер записує цей лог запис до файлу, який має назву, таку ж як і MailID.

Всі чутливі налаштування передаються за допомогою змінних середовища. Такий підхід вважається одним з найбезпечніших. Такий метод наразі є стандартним у контейнеризованих додатках.

```
51    server = os.environ.get('DB_SERVER')  # 'sanacore-dbs.database.windows.net'
52    database = os.environ.get('DB_NAME')  # 'devopsrnd-maillog-db'
53    username = os.environ.get('DB_USER')  # 'devopsrnd-maillog-db'
54    password = os.environ.get('DB_PASSWORD')
55    table = os.environ.get('DB_TABLE')
56    driver = '{ODBC Driver 18 for SQL Server}'
```

Рисунок 3.13 – Отримання необхідних параметрів із змінних середовища

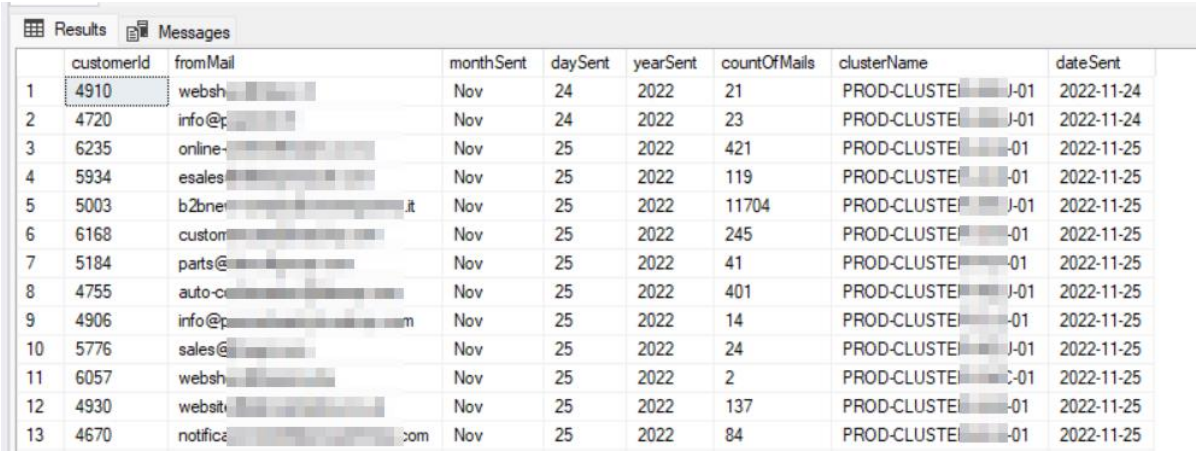
Парсер виконує дві функції перетворення логів до зручного використання Fluent-bit та відправка інформації про успішно відправлені листи до MSSQL бази даних[14].

Fluent Bit це легковісний процесор логів та метрик, передавач інформації, який працює на операційних системах сімейств Linux, OSX, Windows та BSD. Його було розроблено з фокусом на швидкодії для збору подій з різних джерел без значної складності. Fluent Bit є суб-проєктом CNCF[6].

Основними перевагами є підтримка великої різноманітності форматів, включаючи JSON, Regex, LTSV, Logfmt. Наявна сумісність з Prometheus. Перевагою є вбудована підтримка TLS/SSL та асинхронного вводу\виводу мережевого трафіку[18].

Базою даних було обрано Microsoft SQL Database, через вже наявний в архітектурі MSSQL сервер. База даних містить інформацію про користувачів та їх

використання пошти. Є відображення скільки кожен клієнт відправив кожного дня, індетифікаційний номер та в якому кластері від розміщений[39].



Results | Messages

	customerId	fromMail	monthSent	daySent	yearSent	countOfMails	clusterName	dateSent
1	4910	websh	Nov	24	2022	21	PROD-CLUSTE J-01	2022-11-24
2	4720	info@p	Nov	24	2022	23	PROD-CLUSTE J-01	2022-11-24
3	6235	online-	Nov	25	2022	421	PROD-CLUSTE -01	2022-11-25
4	5934	esales	Nov	25	2022	119	PROD-CLUSTE -01	2022-11-25
5	5003	b2bne .it	Nov	25	2022	11704	PROD-CLUSTE J-01	2022-11-25
6	6168	custom	Nov	25	2022	245	PROD-CLUSTE -01	2022-11-25
7	5184	parts@	Nov	25	2022	41	PROD-CLUSTE -01	2022-11-25
8	4755	auto-c	Nov	25	2022	401	PROD-CLUSTE J-01	2022-11-25
9	4906	info@p m	Nov	25	2022	14	PROD-CLUSTE -01	2022-11-25
10	5776	sales@	Nov	25	2022	24	PROD-CLUSTE J-01	2022-11-25
11	6057	websh	Nov	25	2022	2	PROD-CLUSTE C-01	2022-11-25
12	4930	websit	Nov	25	2022	137	PROD-CLUSTE -01	2022-11-25
13	4670	notifica com	Nov	25	2022	84	PROD-CLUSTE -01	2022-11-25

Рисунок 3.14 – База даних з інформацією про успішно надіслані листи

Для відправки даних до бази було використано бібліотеку pyodbc.

```
query = """IF EXISTS (SELECT * FROM [dbo].[{table}] WHERE
customerId = {customerId} AND fromMail = '{fromMail}' AND
daySent = {daySent} AND monthSent = '{monthSent}' AND  yearSent
= {year})

BEGIN

    UPDATE [dbo].[{table}]

        SET countOfMails = countOfMails + 1

        WHERE customerId = {customerId} AND fromMail =
'{fromMail}' AND daySent = {daySent} AND monthSent =
'{monthSent}' AND  yearSent = {year}

END

ELSE

BEGIN

   INSERT INTO {table} ( customerId, fromMail, daySent,
monthSent, yearSent, countOfMails )

   values ( {customerId} , '{fromMail}', {daySent},
'{monthSent}', {year}, 1)

END"""
```

Рисунок 3.15 – Запит для оновлення даних про електронні листи

Вихідний код парсеру знаходиться у додатку А.

3.2.2 Контейнеризація та розробка скриптів для початкового запуску застосунку.

В основу контейнеру з парсером та Postfix було вирішено використовувати базовий контейнер Alpine останньої версії. Alpine, наразі, є стандартом для використання його як основи для своїх додатків. Він постійно оновлюється, постійно отримує сучасні вирішення з безпековими проблемами, має базовий функціонал, і, незважаючи на все вище перераховане, має доволі малий розмір ~5Mb.

Для подальшого використання, скопіюємо початковий скрипт та парсер всередину образу.

```
COPY entrypoint.sh .
COPY parser.py .
```

Рисунок 3.16 – Копіювання необхідних файлів до поточного каталогу образу

Проте, для правильної роботи Postfix та всіх функцій парсеру необхідно встановлювати програмне забезпечення. Для роботи парсеру необхідно, насамперед, встановити python з репозиторію, бібліотеку pyodbc, msodbcsql, mssql-tool. Для роботи Postfix з безпечною авторизацією необхідно встановити Postfix та пакети postfix-pcre, libsasl, cyrus-sasl. Окрім всіх перелічених пакетів необхідно встановити і деякі додаткові. Весь список операцій зі встановленням пакетів зображено на рисунку 3.17.

```
RUN apk update && apk upgrade && apk add --no-cache  \
    python3-dev py3-pip  g++ unixodbc unixodbc-dev postfix postfix-pcre libsasl curl cyrus-sasl  && \
    curl -O https://download.microsoft.com/download/b/9/f/b9f3cce4-3925-46d4-9f46-da08869c6486/msodbcsql18_18.0.1.1-1_amd64.apk && \
    curl -O https://download.microsoft.com/download/b/9/f/b9f3cce4-3925-46d4-9f46-da08869c6486/mssql-tools18_18.0.1.1-1_amd64.apk && \
    apk add --allow-untrusted msodbcsql18_18.0.1.1-1_amd64.apk && \
    apk add --allow-untrusted mssql-tools18_18.0.1.1-1_amd64.apk && \
    pip install --upgrade pip && pip install pyodbc && \
    chmod +x /entrypoint.* && rm msodbcsql1* mssql-tools*
```

Рисунок 3.17 – Встановлення необхідних залежностей

Для правильної роботи авторизації в Postfix необхідно створити системного користувача та додати інформацію про нього до конфігураційних файлів Postfix.

```
RUN adduser postfixuser --system && \
    echo 'postfixuser:postfixusernewpassword' | chpasswd

RUN  saslauthd -a shadow && mkdir -p /etc/postfix/sasl && \
    echo "pwcheck_method: saslauthd"  > /etc/postfix/sasl/smtpd.conf  && \
    echo "mech_list: PLAIN LOGIN" >> /etc/postfix/sasl/smtpd.conf
```

Рисунок 3.18 – Створення та налаштування користувача

Фінальним кроком у Dockerfile є запуск скрипту для початкового налаштування та запуск парсеру.

```
ENTRYPOINT ["/entrypoint.sh"]
```

Рисунок 3.19 – Запуск скрипту entrypoint.sh

Функціями скрипту початкового запуску є налаштування параметрів Postfix та подальший запуск парсеру. В якості Shell використовується ash – це спрощена версія shell від розробників Alpine Linux. Параметри, які використовуються для налаштування, приймаються з змінних середовища, що є безпечним підходом.

```
SMTP_RELAY_HOST=${SMTP_RELAY_HOST?Missing env var SMTP_RELAY_HOST}
SMTP_RELAY_MYHOSTNAME=${SMTP_RELAY_MYHOSTNAME?Missing env var SMTP_RELAY_MYHOSTNAME}
SMTP_RELAY_USERNAME=${SMTP_RELAY_USERNAME?Missing env var SMTP_RELAY_USERNAME}
SMTP_RELAY_PASSWORD=${SMTP_RELAY_PASSWORD?Missing env var SMTP_RELAY_PASSWORD}
SMTP_RELAY_NETWORKS=${SMTP_RELAY_NETWORKS:-10.0.0.0/8,127.0.0.0/8,172.17.0.0/16,192.0.0.0/8}
```

Рисунок 3.20 – Передача параметрів у скрипт з змінних середовища

Всі подальші налаштування відбуваються за допомогою утиліти postconf. Частину цих налаштувань зображено на рисунку 3.21.

```
postconf "relayhost = ${SMTP_RELAY_HOST}"
postconf "myhostname = ${SMTP_RELAY_MYHOSTNAME}"
# Set allowed networks
postconf "mynetworks = ${SMTP_RELAY_NETWORKS}"
# Set configurations
postconf 'smtp_sasl_auth_enable = yes'
postconf 'smtp_sasl_password_maps = lmdb:/etc/postfix/sasl_passwd'
postconf 'smtp_sasl_security_options = noanonymous'
postconf 'smtp_sasl_tls_security_options = noanonymous'
postconf 'smtp_tls_security_level = none'
postconf 'header_size_limit = 4096000'
```

Рисунок 3.21 – Налаштування Postfix через postconf

Також було реалізовано функціонал, коли необхідно передавати нестандартні налаштування при розгорненні рішення. Для цього існує окрема змінна середовища POSTFIX_CUSTOM_CONFIG, в яку через символ «;» передаються нестандартні параметри.

```
# Add extra configuration directly to /etc/postfix/main.cf
if [ -n "${POSTFIX_CUSTOM_CONFIG}" ]; then
    echo "====================================="
    echo "====== Custom configuration ========"
    OLD_IFS=${IFS}
    IFS=';'
    for f in ${POSTFIX_CUSTOM_CONFIG}; do
        f=$(echo "$f" | tr -d ' ')
        echo "$f"
        postconf "$f"
    done
    IFS=${OLD_IFS}
    echo "====================================="
fi
```

Рисунок 3.22 – Парсер додаткових налаштувань через змінну POSTFIX_CUSTOM_CONFIG

3.2.3 Розробка шаблонів для автоматичного розгортання рішення

На сьогоднішній день існує безліч провайдерів хмарних послуг, які надають послуги з орендування їх обчислювальних можливостей та користування Kubernetes кластерами на базі їх обладнання. В даному випадку, архітектура рішення є побудованою на базі Azure Kubernetes Service від постачальника послуг Microsoft. Не дивлячись на це, розробка автоматизованих шаблонів на базі Helm Charts не є залежною від постачальників послуг, у більшості випадків, а є універсальною, тобто, її можна використовувати на майже будь яких середовищах, в тому числі і локальному комп’ютері[45].

Helm це менеджер пакетів для Kubernetes. Helm застосовується для побудови "чартів", які є пакетами Kubernetes ресурсів, що застосовуються для розгортання додатків у кластер. Існують різні види зберігання та розповсюдження цих пакетів: можливо зберігання архівами або просто директоріями з файлами, розповсюдження через приватні або відкриті репозиторії[51].

Helm Chart, у нашому випадку, складається з Deployment, ConfigMap, Secret, Service, PersistentVolume та PersistentVolumeClaim ресурсів.

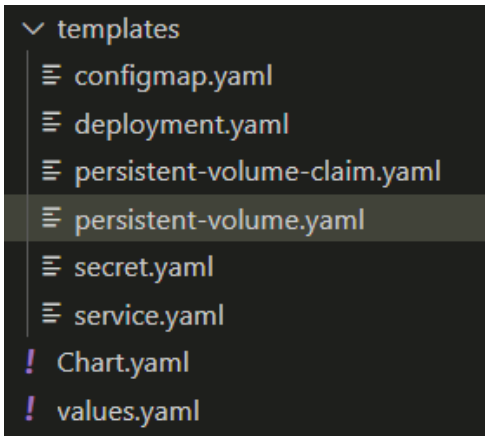


Рисунок 3.23 – Файлова структура Helm Chart

Основною частиною будь якого Helm chart є тип Deployment, в якому описується з чого буде складатися Pod, які контейнери мають бути в нього включені, які файли потрібно до нього примонтувати, та які змінні середовища повинні бути в контейнерах[37].

В даному випадку, Deployment складається з Pod в який входить два контейнери: Fluent-Bit останньої версії та, безпосередньо, контейнер з парсером та Postfix. До контейнеру з Fluent-Bit примонтовуються сховище для логів та файл з конфігурацією.

```
- name: fluent-bit
  image: fluent/fluent-bit
  volumeMounts:
    - name: log-storage
      mountPath: /mnt/log/
      readOnly: true
    - name: fluent-bit-config
      mountPath: /fluent-bit/etc/
```

Рисунок 3.24 – Приклад монтування сховищ до контейнеру

До контейнеру з Postfix примонтовується диск з постійною памятю для зберігання логів в разі перезавантаження кластеру або контейнеру. В Deployment додатково зконфігуровані перевірки чи контейнер успішно ввімкнувся, та чи він досі доступний.

В разі, якщо необхідно мати декілька копій з Postfix на різних хост машинах для більшої надійності, у Deployment налаштовано запобігання встановлення двох копій на один хост.

```
spec:
  topologySpreadConstraints:
    - maxSkew: 1
      topologyKey: kubernetes.io/hostname
      whenUnsatisfiable: DoNotSchedule
      labelSelector:
        matchLabels:
          app_selector: app_selector
```

Рисунок 3.25 – Приклад налаштування розміщення контейнеру

Також, в контейнер передаються змінні середовища, які зберігаються у ConfigMap. У файлі ConfigMap.yaml описано два ресурси ConfigMap: в першому зберігаються змінні середовища, які передаються контейнеру з Postfix та парсером, у другому – конфігураційні файли для Fluent-Bit.

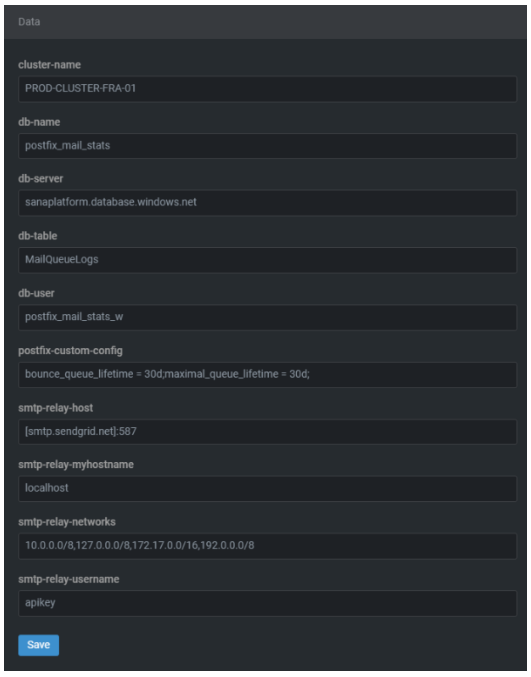


Рисунок 3.26 – ConfigMap який містить з собі змінні середовища

В налаштуваннях FluentBit описано, звідки брати інформацію, як її обробляти, як правильно відфільтрувати тільки потрібні дані, і куди далі відправити сформовану вихідну інформацію. Також міститься інформація, про те, як правильно працювати з інформацією, яку ми отримуємо з розробленого парсеру.

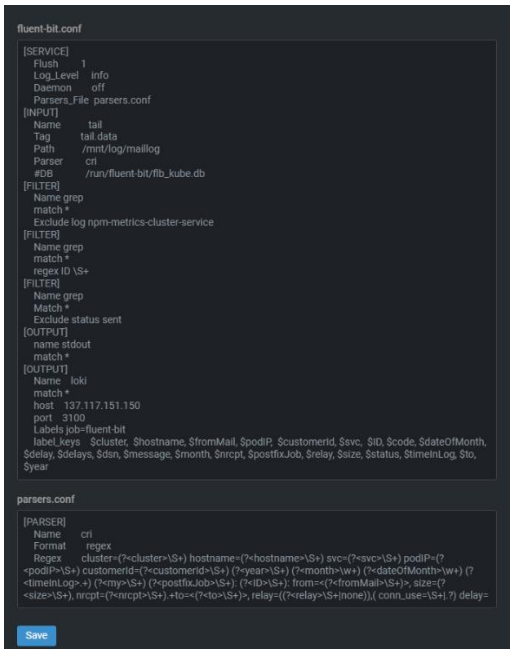


Рисунок 3.27 – ConfigMap з налаштуваннями FluentBit

Для зберігання паролів, сертифікатів, тощо використовується тип Secret, який зберігає інформацію у гешованому вигляді. Варто звернути увагу, що саме гешованому, а не шифрованому. У Secret нашого Helm Chart зберігаємо пароль до MSSQL бази даних та ключ до SMTP провайдера послуг.

Для того, щоб інші сервіси з кластеру, а саме додатки, які відсилають пошту, мали мережеве підключення до Postfix необхідно створити ресурс тиру Service. В ньому вказуються параметри port-forwarding.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: smtp-chart
  namespace: {{ .Values.name }}
spec:
  type: ClusterIP
  ports:
    - port:  {{ .Values.service.port }}
      targetPort:  {{ .Values.service.targetPort }}
      protocol: TCP
      name: smtp
  selector:
    app_selector: app_selector
```

Рисунок 3.28 – Service для роботи з протоколом SMTP

У файлі Chart.yaml міститься базова інформація про сам Helm Chart, така як версія api, яка використовується, назва, версія, версія додатку та короткий опис Helm Chart.

```yaml
apiVersion: v1
name: postfix
version: 1.0.0
appVersion: 1.0.0.0
description: Helm chart for postfix
```

Рисунок 3.29 – Chart.yaml

Змінні до Helm Chart можна передавати різними методами, але найоптимальнішим є зберігати їх у файлі формату yaml та зберігати у git репозиторії з контролем версій для запобігання дрифту конфігурацій.

Передбачено розгортання одного Postfix Deployment на один кластер, але з різною кількістю реплік Postfix, мінімально два.

```yaml
name: postfix
namespace: postfix
clusterName: DevOpsRnd-test-cluster-GWC01
service:
    port: 587
    targetPort: 587
image:
    repo: postfix
    tag: 1.1
resources:
    mem:
        request: 0.1Gi
```

Рисунок 3.30 – Фрагмент values.yaml

На цьому розробку шаблону для автоматичного розгортання рішення можна вважати успішно завершеною. Далі отримані файли буде автоматично обробляти утиліта helm, яка є частиною плагінів порталу Azure DevOps Pipelines.

3.2.4 Розгортання систем агрегування та моніторингу на базі контейнеризованих технологій.

Після того, як систему було успішно розроблено, сконструйовано працездатність декількох систем, необхідно мати повну інформацію про роботу

цих систем, візуалізацію помилок та повну статистику про успішно доставлену пошту.

Існує безліч систем для здійснення моніторингу, але вибір було зупинено на безкоштовних рішеннях з відкритим кодом – Grafana та Grafana Loki.

Grafana Loki агрегатор лог записів, в даному випадку отриманих від FluentBit.

Grafana це сучасний візуалізатор даних, який може працювати практично з будь яким джерелом даних. У нас є необхідність візуалізації даних з Grafana Loki та MSSQL.

Розгортати Grafana та Grafana Loki можна різними способами, включаючи встановлення як сервіс, запуск у контейнері, розгортання у Kubernetes. Так як найсучасніший спосіб розгортання наразі є використання Kubernetes, було вирішено використати саме його.

За допомогою IDE для Kubernetes – Lens, зробимо розгортання вище згаданих сервісів з стандартними налаштуваннями, але додамо сертифікат для безпечного підключення до веб ресурсів.

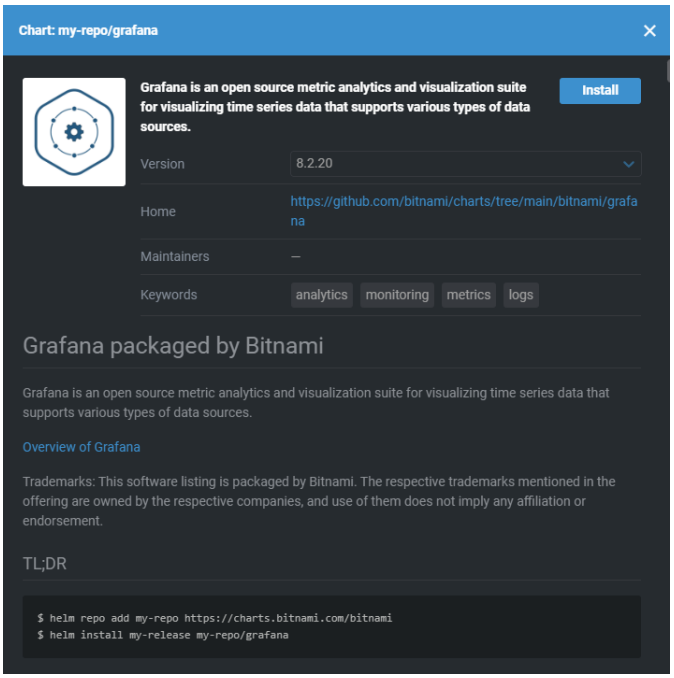


Рисунок 3.31 – Розгортання Grafana через IDE Lens

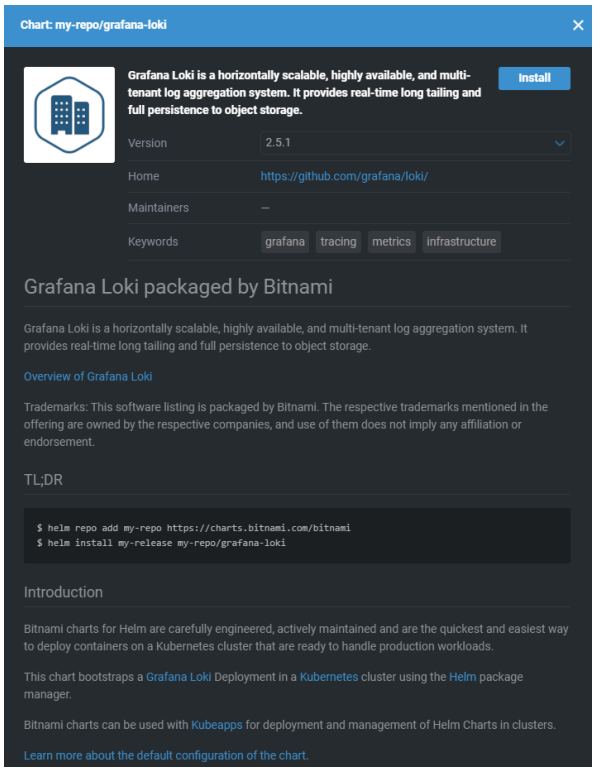


Рисунок 3.32 – Розгортання Grafana Loki через IDE Lens.

В налаштуваннях Grafana джерел даних проводимо налаштування для Grafana Loki та MSSQL.

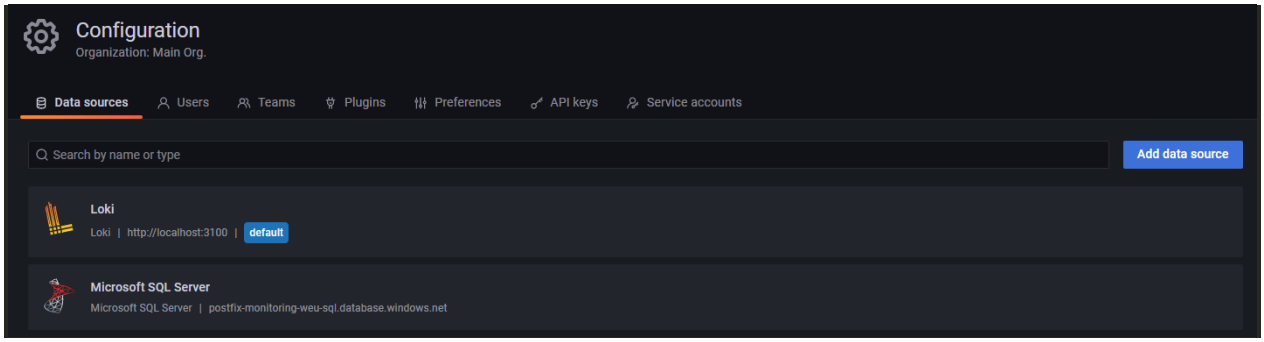


Рисунок 3.33 – Результат успішного підключення Grafana Loki та MSSQL Server для візуалізації у Grafana.

Grafana має досить широкий набір інструментів для виводу інформації, що означає, що оператор зможе всі необхідні дані у тому вигляді, у якому йому буде зручно.

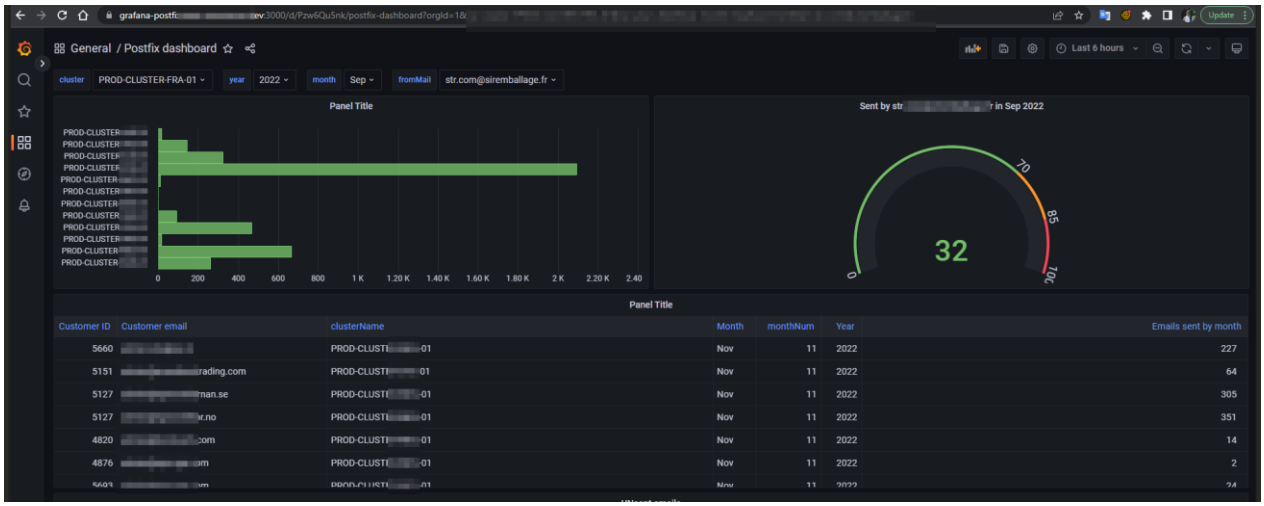


Рисунок 3.34 – Приклад виведення інформації про відправлені листи за допомогою Grafana.

Всі файли конфігурацій та налаштувань знаходяться в додатках. На цьому розробку рішення з імплементації шаблонів розгортання можна вважати завершеною. Моніторингові системи встановлено на кластерне середовище та розроблено моніторингові дошки для зручного отримання необхідної інформації. В подальшому ці дошки будуть доповнюватися та модифіковуватися, в залежності від вимог.

Висновки до розділу 3

Отже, в третьому розділі було розроблено функціональну модель проекту підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ. Побудовано принципові схеми роботи додатків. Було розроблено детальні схеми роботи додатків в кластерному середовищі. Також, було побудовано діаграми послідовності дій.

Висновок

В ході виконання кваліфікаційної роботи було виконано дослідження актуальності теми проекту підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ. Було визначено мету, об’єкт та предмет дослідження.

Було виконано аналіз сучасного стану розвитку (проект підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ), а саме: виконано аналіз програмних продуктів для відправки пошти, досліджено всі можливі вибори інфраструктури та хмарних провайдерів, досліджено різні сервери авторизації пошти.

Було досліджено механізми захисту електронної пошти, а саме: Sender Policy Framework, DomainKeys Identified Mail, Domain-based Message Authentication, Reporting & Conformance та Transport Layer Security.

За допомогою отриманої інформації та проведених досліджень було виконано розробку вимог до проекту підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ. Було визначено, що для проекту буде оптимальним використання Postfix, як агента відправки пошти, SendGrid, як сервер авторизації пошти, Microsoft Azure та Azure Kubernetes Service, як хмарного провайдера для розміщення інфраструктури застосунків.

В ході виконання кваліфікаційної роботи з було виконано розробку програмного рішення підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ. Було визначено мету, об’єкт та предмет дослідження.

Було виконано розробку та контейнеризацію застосунку для відправки пошти, парсингу та логування поштового сервісу. Було розроблено парсер на мові програмування Python, розроблено скрипт для налаштування та початкового запуску парсеру. Було розроблено Dockerfile з описом контейнера, для подальшого його побудови і розповсюдження на приватний репозиторій образів.

Було підготовлено шаблон для автоматичного розгортання рішення та було виконано розгортання систем моніторингу на базі контейнеризованих середовищ. Було розгорнено систему візуалізації даних Grafana та систему агрегування логів Grafana Loki. Для Grafana було виконано налаштування підключень, а також розроблено інтерактивні дошки для зручної візуальної обробки даних.

В результаті виконаних маніпуляцій отримали систему, яка може автоматично розгортатися на різні Kubernetes кластери різних провайдерів пошти. Ця система дозволяє швидко та надійно обробляти дані про відправлену пошту, дає змогу робити аналіз по заданим параметрам. Також за допомогою цієї системи є можливість налаштування сповіщень про непередбачувані події з системою та системою провайдерів електронної пошти.

Проект підсистеми захисту системи електронної пошти підприємства, побудованої на базі контейнеризованих середовищ дає підвищений рівень найдійності та оглядовості роботи сервісів електронної пошти у підприємстві.

Зважаючи на все вище сказане, можна вважати мету кваліфікаційної роботи успішно досягнутою.

СПИСОК ЛІТЕРАТУРИ

1. Cheshire J. Exam ref AZ-900 microsoft azure fundamentals. Pearson Education, 2020. 304 p.
2. Dent K. Postfix: the definitive guide. O'Reilly Media, Inc., 2003. 278 p.
3. Dotson C. Practical cloud security: a guide for secure design and deployment. O'Reilly Media, Incorporated, 2019. 196 p.
4. Ejsmont A. Web scalability for startup engineers: Tips & techniques for scaling your Web application. 2015. 396 p.
5. Exam ref AZ-104 microsoft azure administrator / M. Washam et al. Pearson Education, 2020. 512 p.
6. Heinlein P., Hartleben P. IMAP and POP3: building a mail server with courier and cyrus. No Starch Press, 2008. 368 p.
7. Hoffman A. Web application security: exploitation and countermeasures for modern web applications. O'Reilly Media, 2020. 330 p.
8. Ibryam B., Huβ R. Kubernetes patterns: reusable elements for designing cloud native applications. O'Reilly Media, Incorporated, 2019. 266 p.
9. Justice M. From amps to apps: how computers work. No Starch Press, Incorporated, 2020. 380 p.
10. Kleppmann M. Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media, 2017. 624 p.
11. Kozierok C. The TCP/IP guide: a comprehensive, illustrated internet protocols reference. No Starch Press, 2005. 1616 p.
12. Kubernetes best practices: blueprints for building successful applications on kubernetes / B. Burns et al. O'Reilly Media, Incorporated, 2019. 268 p.
13. Newman S. Building microservices: designing fine-grained systems. O'Reilly Media, Incorporated, 2020. 250 p.
14. Practical network security monitoring. No Starch Press,US, 2013.
15. Rice L. Container security: fundamental technology concepts that protect containerized applications. O'Reilly Media, Incorporated, 2020. 200 p.

16. Sanders C. Practical packet analysis: using wireshark to solve real-world network problems. No Starch Press, 2007. 192 p.

17. Security. *SendGrid*. URL: https://sendgrid.com/policies/security/ (date of access: 17.07.2022).

18. Thomas O., Diogenes Y. Exam ref AZ-500 microsoft azure security technologies. Pearson Education, 2020. 410 p.

19. Toroman M. Hands-On Cloud Administration in Azure: Implement, monitor, and manage important Azure services and components including IaaS and PaaS. Packt Publishing, 2018. 390 p.

20. Vehent J. Securing devops: security in the cloud. Manning Publications, 2018. 384 p.

21. Aggrawal S. A complete review on SMTP protocol. International journal of recent advancement in engineering & research. 2017. Vol. 2, no. 3. P. 15. URL: https://doi.org/10.24128/ijraer.2017.mn78ab (date of access: 17.07.2022).

22. AWS architecture overview / B. Beach et al. Pro powershell for amazon web services. Berkeley, CA, 2019. P. 1–8. URL: https://doi.org/10.1007/978-1-4842-4850-8_1 (date of access: 17.07.2022).

23. Bai H. Containers and microservices. Zen of cloud. 2019. P. 55–92. URL: https://doi.org/10.1201/9780429278594-3 (date of access: 17.07.2022).

24. Buchanan S., Rangama J., Bellavance N. Helm charts for azure kubernetes service. Introducing azure kubernetes service. Berkeley, CA, 2019. P. 151–189. URL: https://doi.org/10.1007/978-1-4842-5519-3_8 (date of access: 17.07.2022).

25. Chakraborty M., Kundan A. P. Grafana. Monitoring Cloud-Native Applications. Berkeley, CA, 2021. P. 187–240. URL: https://doi.org/10.1007/978-1-4842-6888-9_6 (date of access: 17.07.2022).

26. DomainKeys identified mail (DKIM) signatures / ed. by D. Crocker, T. Hansen, M. Kucherawy. RFC Editor, 2011. URL: https://doi.org/10.17487/rfc6376 (date of access: 17.07.2022).

27. Jain V. Analyzing layer 2 and layer 3 traffic. Wireshark fundamentals. Berkeley, CA, 2022. P. 79–134. URL: https://doi.org/10.1007/978-1-4842-8002-7_3 (date of access: 17.07.2022).

28. Jangla K. Containers. Accelerating development velocity using docker. Berkeley, CA, 2018. P. 1–8. URL: https://doi.org/10.1007/978-1-4842-3936-0_1 (date of access: 17.07.2022).

29. Martin P. Security. Kubernetes. Berkeley, CA, 2020. P. 115–156. URL: https://doi.org/10.1007/978-1-4842-6494-2_11 (date of access: 17.07.2022).

30. Perkins B. Microsoft azure architect technologies and design complete study guide exams AZ-303 and AZ-304. Wiley & Sons, Limited, John, 2021. 940 p.

31. Python Algorithms: mastering basic algorithms in the Python language. Choice reviews online. 2011. Vol. 48, no. 10. P. 48–5731–48–5731. URL: https://doi.org/10.5860/choice.48-5731 (date of access: 17.07.2022).

32. Relan K., Singhal V. Pentest ninja. The second international conference, Udaipur, India, 4–5 March 2016. New York, New York, USA, 2016. URL: https://doi.org/10.1145/2905055.2905243 (date of access: 17.07.2022).

33. Rhodes B., Goerzen J. Network data and network errors. Foundations of python network programming. Berkeley, CA, 2014. P. 75–92. URL: https://doi.org/10.1007/978-1-4302-5855-1_5 (date of access: 17.07.2022).

34. Rhoton J. Programmer's guide to internet mail (HP technologies). Digital Press, 2000. 291 p.

35. Secure key exchange over an insecure medium with public key cryptography. Implementing SSL/TLS using cryptography and PKI. Indianapolis, IN, USA, 2011. P. 91–155. URL: https://doi.org/10.1002/9781118255797.ch3 (date of access: 17.07.2022).

36. Spf. Encyclopedia of genetics, genomics, proteomics and informatics. Dordrecht, 2008. P. 1855. URL: https://doi.org/10.1007/978-1-4020-6754-9_15959 (date of access: 17.07.2022).

37. Tls. Encyclopedia of cryptography and security. Boston, MA, 2011. P. 1305. URL: https://doi.org/10.1007/978-1-4419-5906-5_1043 (date of access: 17.07.2022).

38. Wood D. Programming Internet email. Sebastopol, CA : O'Reilly, 1999. 362 p.

39. Žogots D., Ežmale S. Posibilities of using docker containers. HUMAN. ENVIRONMENT. TECHNOLOGIES. proceedings of the students international scientific and practical conference. 2019. No. 23. P. 160. URL: https://doi.org/10.17770/het2019.23.4409 (date of access: 17.07.2022).

40. Das R., de Guise P. Amazon web services. Protecting information assets and IT infrastructure in the cloud. 2019. P. 61–114. URL: https://doi.org/10.1201/9780429261602-2 (date of access: 17.07.2022).

41. Practical constructions of symmetric-key primitives. Introduction to modern cryptography. 2014. P. 213–260. URL: https://doi.org/10.1201/b17668-11 (date of access: 17.07.2022).

42. RFC1035. Domain names - implementation and specification. Effective from 1987-11-13. Official edition. 1987. URL: https://datatracker.ietf.org/doc/html/rfc1035 (date of access: 09.07.2022).

43. RFC3826. The advanced encryption standard (AES) cipher in the SNMP user-based security model. Effective from 2004-06-01. Official edition. 2004. URL: https://datatracker.ietf.org/doc/html/rfc3826 (date of access: 17.07.2022).

44. RFC5321. Simple mail transfer protocol. Effective from 2008-10-01. Official edition. 2008. URL: https://datatracker.ietf.org/doc/html/rfc5321 (date of access: 17.07.2022).

45. RFC6376. DomainKeys identified mail (DKIM) signatures. Effective from 2011-09-01. Official edition. 2011. URL: https://datatracker.ietf.org/doc/html/rfc6376 (date of access: 15.07.2022).

46. RFC7208. Sender policy framework (SPF) for authorizing use of domains in email, version 1. Effective from 2014-04-01. Official edition. 2014. URL: https://datatracker.ietf.org/doc/html/rfc7208 (date of access: 16.07.2022).

47. RFC7489. Domain-based message authentication, reporting, and conformance (DMARC). Effective from 2015-03-03. Official edition. 2015. URL: https://datatracker.ietf.org/doc/html/rfc7489 (date of access: 14.07.2022).

48. RFC793. Transmission control protocol. Effective from 1981-09-01. Official edition. California, 1981. URL: https://datatracker.ietf.org/doc/html/rfc793 (date of access: 17.07.2022).

49. RFC8446. The transport layer security (TLS) protocol version 1.3. Effective from 2018-08-01. Official edition. 2018. URL: https://datatracker.ietf.org/doc/html/rfc8446 (date of access: 17.07.2022).

50. Serious cryptography: a practical introduction to modern encryption. No Starch Press, 2017.

51. The real world. Applied cryptography, second edition. Indianapolis, Indiana, 2015. P. 559. URL: https://doi.org/10.1002/9781119183471.part4 (date of access: 17.07.2022).

ДОДАТКИ

Додаток А

Dockerfile

```
FROM alpine:latest
    EXPOSE 587/tcp
    COPY entrypoint.sh .
    COPY parser.py .
    RUN apk update && apk upgrade && \
        apk add --no-cache  \
        python3-dev py3-pip  g++ unixodbc unixodbc-dev \
        postfix postfix-pcre libsasl curl cyrus-sasl  && \
        curl -O https://download.microsoft.com/download/b/9/f/b9f3cce4-3925-46d4-9f46-da08869c6486/msodbcsql18_18.0.1.1-1_amd64.apk && \
        curl -O https://download.microsoft.com/download/b/9/f/b9f3cce4-3925-46d4-9f46-da08869c6486/mssql-tools18_18.0.1.1-1_amd64.apk && \
        apk add --allow-untrusted msodbcsql18_18.0.1.1-1_amd64.apk && \
        apk add --allow-untrusted mssql-tools18_18.0.1.1-1_amd64.apk && \
        pip install --upgrade pip && pip install pyodbc && \
        chmod +x /entrypoint.* && \
        rm msodbcsql1* mssql-tools*
    RUN adduser postfixuser --system && \
        echo 'postfixuser:postfixusernewpassword' | chpasswd
    RUN  saslauthd -a shadow && mkdir -p /etc/postfix/sasl && \
        echo "pwcheck_method: saslauthd"  > /etc/postfix/sasl/smtpd.conf  && \
        echo "mech_list: PLAIN LOGIN" >> /etc/postfix/sasl/smtpd.conf
    ENTRYPOINT ["/entrypoint.sh"]
```

Додаток Б

entrypoint.sh

```
#!/bin/ash
SMTP_RELAY_HOST=${SMTP_RELAY_HOST?Missing env var SMTP_RELAY_HOST}
SMTP_RELAY_MYHOSTNAME=${SMTP_RELAY_MYHOSTNAME?Missing env var
SMTP_RELAY_MYHOSTNAME}
SMTP_RELAY_USERNAME=${SMTP_RELAY_USERNAME?Missing env var
SMTP_RELAY_USERNAME}
SMTP_RELAY_PASSWORD=${SMTP_RELAY_PASSWORD?Missing env var
SMTP_RELAY_PASSWORD}
SMTP_RELAY_NETWORKS=${SMTP_RELAY_NETWORKS:-
10.0.0.0/8,127.0.0.0/8,172.17.0.0/16,192.0.0.0/8}
[ -d /var/spool/postfix/etc ] && rm -r /var/spool/postfix/etc

#Exposing 587 port
echo "587 inet n - n - - smtpd" >>/etc/postfix/master.cf

echo "${SMTP_RELAY_HOST}
${SMTP_RELAY_USERNAME}:${SMTP_RELAY_PASSWORD}" >/etc/postfix/sasl_passwd
postmap /etc/postfix/sasl_passwd
rm /etc/postfix/sasl_passwd

# These are required
postconf "relayhost = ${SMTP_RELAY_HOST}"
postconf "myhostname = ${SMTP_RELAY_MYHOSTNAME}"

# Set allowed networks
postconf "mynetworks = ${SMTP_RELAY_NETWORKS}"
# Set configurations
postconf 'smtp_sasl_auth_enable = yes'
postconf 'smtp_sasl_password_maps = lmdb:/etc/postfix/sasl_passwd'
postconf 'smtp_sasl_security_options = noanonymous'
postconf 'smtp_sasl_tls_security_options = noanonymous'
postconf 'smtp_tls_security_level = none'
postconf 'header_size_limit = 4096000'
postconf 'inet_protocols = all'
postconf 'disable_dns_lookups = yes'
postconf 'maillog_file = /var/log/maillog'

postconf 'smtpd_sasl_auth_enable = yes'
postconf 'smtpd_sasl_path = smtpd'
postconf 'smtpd_sasl_type = cyrus'
postconf 'cyrus_sasl_config_path = /etc/postfix/sasl'
postconf 'broken_sasl_auth_clients = yes'
postconf 'smtpd_sasl_security_options = noanonymous'
postconf 'smtputf8_enable = no'
postconf 'always_add_missing_headers = yes'
sed -i '/readme_directory =/d' /etc/postfix/main.cf
# Add extra configuration directly to /etc/postfix/main.cf
if [ -n "${POSTFIX_CUSTOM_CONFIG}" ]; then
    echo
    echo "==================================="
```

```
    echo "====== Custom configuration ========"
    OLD_IFS=${IFS}
    IFS=';'
    for f in ${POSTFIX_CUSTOM_CONFIG}; do
        f=$(echo "$f" | tr -d ' ')
        echo "$f"
        postconf "$f"
    done
    IFS=${OLD_IFS}
    echo "====================================="
fi
postfix set-permissions
python3 parser.py
```

Додаток В

parser.py

```
import subprocess
import re
from typing import cast
import pyodbc
from datetime import datetime
import time
import os
import math
def convertToNumber(s):
   return int.from_bytes(s.encode(), 'little')
def convertFromNumber(n):
   return n.to_bytes(math.ceil(n.bit_length() / 8), 'little').decode()
def customerIdGenerate(customerId):
   try:
      customerId = (int(customerId))
   except:
      customerId = convertToNumber(customerId)
   return str(customerId)
def extract_data2(string):
   pattern = re.compile(
      r"""cluster=(?P<cluster>\S+) hostname=(?P<hostname>\S+) svc=(?P<svc>\S+) podIP=(?P<podIP>\S+)
customerId=(?P<customerId>\S+) (?P<year>\S+) (?P<month>\w+) (?P<dateOfMonth>\w+) (?P<timeInLog>.+)
(?P<my>\S+) (?P<postfixJob>\S+): (?P<ID>\S+): from=<(?P<fromMail>\S+)>, size=(?P<size>\S+),
nrcpt=(?P<nrcpt>\S+).+to=<(?P<to>\S+)>, relay=((?P<relay>\S+|none)),( conn_use=\S+|.?)
delay=(?P<delay>\S+), delays=(?P<delays>\S+), dsn=(?P<dsn>\S+), status=(?P<status>\S+)
.*\((.*(?P<code>\d{3})|.*) (?P<message>.*)\)""")
   match = pattern.match(string)
   podIP = match.group("podIP")
   customerId = match.group("customerId")
   fromMail = match.group("fromMail")
   year = match.group("year")
   monthSent = match.group("month")
   dateOfMonth = match.group("dateOfMonth")
   status = match.group("status")
   return (fromMail, year, monthSent, dateOfMonth, status, podIP, customerId)
def replace_line(file_name, lineNum, text):
   with open(file_name, 'r') as file:
      lines = file.readlines()
   lines[lineNum] = text
   with open(file_name, 'w') as file:
      file.writelines(lines)
```

```
CLUSTER_NAME = os.environ.get('CLUSTER_NAME')
HOSTNAME = os.uname().nodename
filename = "/var/log/maillog"  # ?Maillog file pat
# ?Credentials for MSSQL query
server = os.environ.get('DB_SERVER')  # 'sanacore-dbs.database.windows.net'
database = os.environ.get('DB_NAME')  # 'devopsrnd-maillog-db'
username = os.environ.get('DB_USER')  # 'devopsrnd-maillog-db'
password = os.environ.get('DB_PASSWORD')
table = os.environ.get('DB_TABLE')
driver = '{ODBC Driver 18 for SQL Server}'
svc = '777'
query = """IF EXISTS (SELECT * FROM [dbo].[{table}] WHERE customerId = {customerId} AND fromMail
= '{fromMail}' AND daySent = {daySent} AND monthSent = '{monthSent}' AND  yearSent = {year})
BEGIN
   UPDATE [dbo].[{table}]
      SET countOfMails = countOfMails + 1
      WHERE customerId = {customerId} AND fromMail = '{fromMail}' AND daySent = {daySent} AND
monthSent = '{monthSent}' AND  yearSent = {year}
END
ELSE
BEGIN
  INSERT INTO {table} ( customerId, fromMail, daySent, monthSent, yearSent, countOfMails )
  values ( {customerId} , '{fromMail}', {daySent}, '{monthSent}', {year}, 1)
END"""
print("=======================================")
print("======== Starting python script ========")
print("=======================================")
while os.path.exists(filename) != True:  # ?
   subprocess.Popen(['postfix', 'start'])  # ? Check if postfix have started
   time.sleep(3)  # ?
patternLongSvc = re.compile(
   "client=(.+)\.(?P<svc>.+)\.(.+)\.(.+)\.(.+)\.(.+)\[(?P<podIP>.+)\]")
patternUnknownSvc = re.compile("client=(?P<svc>unknown)\[(?P<podIP>.+)\]")
sessionIdPattern = re.compile("[A-F,0-9]{5,12}")
customerIdPattern = re.compile("\w+-(?P<customerId>\w+)-.+")
svcPattern = re.compile("client=.+")
fromPattern = re.compile("from=<.+>")
toPattern = re.compile("to=<.+>")
trashFilter=re.compile(".monitoring.svc.cluster.local")
f = subprocess.Popen(['tail', '-F', filename],
                stdout=subprocess.PIPE, stderr=subprocess.PIPE)
while True:
   try:
      year = str(datetime.now().year)
      # ? Get new line from file and decode it from binary to UTF
      logline = f.stdout.readline().decode('UTF-8')

      if trashFilter.match(logline): pass
      else: print(logline)

      logArray = logline.split()
      sessionId = logArray[5][:-1:]

      if sessionIdPattern.match(sessionId):
         sessionidFile = '/var/spool/postfix/qstorage/' + sessionId
         logType = logArray[6]

         if svcPattern.match(logType):
            if patternLongSvc.match(" ".join(logArray[6:])):
               matchSvc = patternLongSvc.match(" ".join(logArray[6:]))
               svc = matchSvc.group("svc")
```

```
                podIP = matchSvc.group("podIP")
                try:
                    customerId = customerIdGenerate(
                        customerIdPattern.match(svc).group("customerId"))
                except:
                    customerId = "777"

            elif patternUnknownSvc.match(" ".join(logArray[6:])):
                matchSvc = patternUnknownSvc.match(" ".join(logArray[6:]))
                svc = matchSvc.group("svc")
                podIP = matchSvc.group("podIP")
                customerId = customerIdGenerate(svc)
            startLog = 'cluster=' + CLUSTER_NAME + ' hostname=' + HOSTNAME + ' svc=' + \
                svc + ' podIP=' + podIP + ' customerId=' + customerId + ' ' + year + '\n2\n3\n'
            # ? Making a file with ID in name
            with open(sessionidFile, 'w', encoding='utf-8') as file:
                file.write(startLog)
        if os.path.exists(sessionidFile):
            if fromPattern.match(logType):
                replace_line(sessionidFile, 1, logline + '\n')
            if toPattern.match(logType):
                replace_line(sessionidFile, 2,  " ".join(logArray[5:]))
                resultLine = " ".join(open(sessionidFile, 'r').readlines()).replace(
                    '\n', '').replace('  ', ' ') + '\n'
                with open("/var/log/maillog", "a", encoding='utf-8') as file:
                    file.write(resultLine)  # ?   Writing result to the maillog
                try:  # !# PAY YOUR ATTENTION ALL ERROR LOGS ARE PARSED WITH FLUENT-BIT, SO
YOU CAN FIND THEM IN THE GRAFANA
                    fromMail, year, monthSent, daySent, status,  podIP, customerId = extract_data2(
                        resultLine)  # ? Parsing
                    if status == "sent":  # Making query to the DB
                        with
pyodbc.connect('DRIVER='+driver+';SERVER=tcp:'+server+';PORT=1433;DATABASE='+database+';UID='+
username+';PWD=' + password) as conn:
                            with conn.cursor() as cursor:
                                cursor.execute(query.format(fromMail=fromMail, year=year, daySent=daySent,
                                        monthSent=monthSent,  podIP=podIP, customerId=customerId[0:8], table=table))
                                print(customerId[0:8] + " Successfull query")
                except Exception as e:
                    print(e)
            elif logType == 'removed':
                os.remove(sessionidFile)  # ? removing tmp ID file
    except Exception as e:
        print(e)
```

Додаток Г

configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Values.name }}
  labels:
    app: {{ .Values.name }}
data:
  db-server: "{{ .Values.db.server }}"
  db-name: "{{ .Values.db.name }}"
  db-user: "{{ .Values.db.user }}"
  db-table: "{{ .Values.db.table }}"
  smtp-relay-host: "{{ .Values.smtp.relayHost }}"
```

```
  smtp-relay-myhostname: "{{ .Values.smtp.relayMyhostname }}"
  smtp-relay-username: "{{ .Values.smtp.relayUsername }}"
  smtp-relay-networks: "{{ .Values.smtp.relayNetworks }}"
  postfix-custom-config: "{{ .Values.postfixCustomConfig }}"
  cluster-name: "{{ .Values.clusterName }}"
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-sidecar-config
  labels:
    app: {{ .Values.name }}
data:
  fluent-bit.conf: |
    [SERVICE]
        Flush        1
        Log_Level    info
        Daemon       off
        Parsers_File  parsers.conf
    [INPUT]
        Name          tail
        Tag           tail.data
        Path          /mnt/log/maillog
        Parser        cri
        #DB           /run/fluent-bit/flb_kube.db
    [FILTER]
        Name grep
        match *
        Exclude log npm-metrics-cluster-service
    [FILTER]
        Name grep
        match *
        regex ID \S+
    [OUTPUT]
        name stdout
        match *
    [OUTPUT]
        Name    loki
        match *
        host    20.105.188.238
        port    3100
        Labels job=fluent-bit
        label_keys       $cluster,  $hostname, $fromMail, $podIP,  $customerId, $svc,  $ID, $code, $dateOfMonth, $delay, $delays, $dsn, $message, $month, $nrcpt, $postfixJob, $relay, $size, $status, $timeInLog, $to, $year
        Line_Format key_value
  parsers.conf: |
    [PARSER]
        Name       cri
        Format       regex
        Regex                cluster=(?<cluster>\S+) hostname=(?<hostname>\S+) svc=(?<svc>\S+) podIP=(?<podIP>\S+)  customerId=(?<customerId>\S+)  (?<year>\S+)  (?<month>\w+) (?<dateOfMonth>\w+)  (?<timeInLog>.+)  (?<my>\S+)  (?<postfixJob>\S+): (?<ID>\S+): from=<(?<fromMail>\S+)>,  size=(?<size>\S+),  nrcpt=(?<nrcpt>\S+).+to=<(?<to>\S+)>, relay=((?<relay>\S+|none)),(  conn_use=\S+|.?)  delay=(?<delay>\S+),  delays=(?<delays>\S+), dsn=(?<dsn>\S+), status=(?<status>\S+) .*\((.*(?<code>\d{3})|.*) (?<message>.*)\)
```

```
    Time_Key        time
    Time_Format     %Y-%m-%dT%H:%M:%S.%L%z
```

Додаток Д

deployment.yaml

```
{{range $i := until ( $.Values.replicas | int ) }}
apiVersion: apps/v1
kind: Deployment
metadata:
  name: "{{ $.Values.name }}-{{  $i| add 1 }}"
  namespace: {{ $.Values.namespace }}
  labels:
    app: {{ $.Values.name }}-{{  $i| add 1 }}
    app_selector: app_selector
spec:
  template:
    metadata:
      name: {{ $.Values.name }}-{{  $i| add 1 }}
      labels:
        app: {{ $.Values.name }}-{{  $i| add 1 }}
        app_selector: app_selector
    spec:
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: kubernetes.io/hostname
          whenUnsatisfiable: DoNotSchedule
          labelSelector:
            matchLabels:
              app_selector: app_selector
      volumes:
        - name: mailq-storage-{{  $i| add 1 }}
          persistentVolumeClaim:
            claimName: mailq-node-{{  $i| add 1 }}
        - name: log-storage
          emptyDir: {}
        - name: fluent-bit-config
          configMap:
            name: fluent-bit-sidecar-config
      containers:
        - name: fluent-bit
          image: fluent/fluent-bit
          volumeMounts:
            - name: log-storage
              mountPath: /mnt/log/
              readOnly: true
            - name: fluent-bit-config
              mountPath: /fluent-bit/etc/
        - name: {{ $.Values.name }} ##postfix
          image: "sanaacrweu01.azurecr.io/{{ $.Values.image.repo }}:{{ $.Values.image.tag }}"
          imagePullPolicy: Always
          volumeMounts:
```

```
    - name: mailq-storage-{{  $i| add 1 }}
      mountPath: /var/spool/postfix
    - name: log-storage
      mountPath: /var/log/
  env:
    - name: DB_SERVER
      valueFrom:
        configMapKeyRef:
          name: postfix
          key: db-server
    - name: DB_NAME
      valueFrom:
        configMapKeyRef:
          name: postfix
          key: db-name
    - name: DB_USER
      valueFrom:
        configMapKeyRef:
          name: postfix
          key: db-user
    - name: DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: "{{ $.Values.name }}-relay-pass"
          key: db-password
    - name: DB_TABLE
      valueFrom:
        configMapKeyRef:
          name: postfix
          key: db-table
    - name: CLUSTER_NAME
      valueFrom:
        configMapKeyRef:
          name: postfix
          key: cluster-name
    - name: SMTP_RELAY_HOST
      valueFrom:
        configMapKeyRef:
          name: postfix
          key: smtp-relay-host
    - name: SMTP_RELAY_MYHOSTNAME
      valueFrom:
        configMapKeyRef:
          name: postfix
          key: smtp-relay-myhostname
    - name: SMTP_RELAY_NETWORKS
      valueFrom:
        configMapKeyRef:
          name: postfix
          key: smtp-relay-networks
    - name: SMTP_RELAY_USERNAME
      valueFrom:
        configMapKeyRef:
          name: postfix
          key: smtp-relay-username
    - name: SMTP_RELAY_PASSWORD
```

```
            valueFrom:
              secretKeyRef:
                name: "{{ $.Values.name }}-relay-pass"
                key: smtp-relay-password
          - name: POSTFIX_CUSTOM_CONFIG
            valueFrom:
              configMapKeyRef:
                name: postfix
                key: postfix-custom-config
        securityContext:
          privileged: true
          capabilities:
            add:
              - NET_ADMIN
        resources:
          requests:
            cpu: 0.1
            memory: {{ $.Values.resources.mem.request | default "0.1Gi" }}
        ports:
          - containerPort: {{ $.Values.service.port }}
            protocol: TCP
        startupProbe:
          tcpSocket:
            port: {{ $.Values.service.port }}
          initialDelaySeconds: 20
          periodSeconds: 5
          failureThreshold: 3
        livenessProbe:
          tcpSocket:
            port: {{ $.Values.service.port }}
          initialDelaySeconds: 0
          periodSeconds: 5
          failureThreshold: 4
        readinessProbe:
          tcpSocket:
            port: {{ $.Values.service.port }}
          initialDelaySeconds: 0
          periodSeconds: 5
          failureThreshold: 3
    selector:
      matchLabels:
        app: {{ $.Values.name }}-{{  $i| add 1 }}

---
{{end}}
```

Додаток Е

Secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: "{{ .Values.name }}-relay-pass"
  labels:
    app: {{ .Values.name }}
```

```
type: Opaque
data:
  smtp-relay-password: {{ .Values.smtp.relayPassword | b64enc }}
  db-password: {{ .Values.db.pass | b64enc }}
```

Додаток Ж

Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: smtp-chart
  namespace: {{ .Values.name }}
spec:
  type: ClusterIP
  ports:
    - port:  {{ .Values.service.port }}
      targetPort:  {{ .Values.service.targetPort }}
      protocol: TCP
      name: smtp
  selector:
    app_selector: app_selector
```

Додаток И

Chart.yaml

```
apiVersion: v1
name: postfix
version: 1.0.0
appVersion: 1.0.0.0
description: Helm chart for postfix
```

Додаток К

values.yaml

```
name: postfix
namespace: postfix
clusterName:
service:
   port:
   targetPort:
image:
   repo:
   tag:
resources:
   mem:
      request:
replicas: 1
subscription:
```

```
smtp:
  relayHost:
  relayMyhostname:
  relayUsername:
  relayPassword:
  relayNetworks:
postfixCustomConfig:
db:
  pass:
  server:
  table:
```