ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «СИСТЕМА КОНТРОЛЮ ДОСТУПУ ДО ВНУТРІШНІХ СЕРВІСІВ НА БАЗІ ТЕХНОЛОГІЙ REVERSE PROXY ТА LET'S ENCRYPT»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми «Комп'ютерна інженерія»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Дмитро МАЛІКОВ*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КІД-41
Дмитро МАЛІКОВ
Ім'я, ПРІЗВИЩЕ

Керівник: Ярослав ТОРОШАНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерна інженерія

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
«_____» _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Маліков Дмитро Вікторович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Система контролю доступу до внутрішніх сервісів на базі технологій Reverse Proxy та Let's Encrypt»
керівник кваліфікаційної роботи: Ярослав ТОРОШАНКО, кандидат технічних наук, доцент, старший науковий співробітник ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026р. №51
2. Строк подання кваліфікаційної роботи«14» травня 2026р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, стандарти та протоколи HTTP/HTTPS і TLS, технології Reverse Proxy та Nginx Proxy Manager, сервіси динамічного DNS і Let's Encrypt, засоби контейнеризації Docker, методи маршрутизації та контролю доступу до внутрішніх сервісів, тестування та порівняльний аналіз ефективності системи.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Основи маршрутизації та безпеки внутрішніх веб-ресурсів
    2. Побудова системи доступу на базі Docker та Nginx Proxy Manager
    3. Аналіз ефективності та напрямки покращення проксі-системи
5. Перелік ілюстративного матеріалу: *презентація*
    1. Мета, об'єкт, предмет дослідження
    2. Актуальність роботи
    3. Архітектура розробленої системи
    4. Використані компоненти
    5. Практична реалізація системи
    6. Налаштування домену та SSL
    7. Практичне тестування
    8. Порівняння з альтернативними рішеннями
    9. Оптимізація продуктивності

10. Напрямки модернізації
11. Висновки
12. Публікації та апробація роботи

6. Дата видачі завдання «20» лютого 2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	22.02-20.03.2026	Виконано
2	Обробка матеріалу	20.03-30.03.2026	Виконано
3	Розробка теоретичної частини	31.03-10.04.2026	Виконано
4	Побудова системи доступу на базі Docker та Nginx Proxy Manager	11.04-20.04.2026	Виконано
5	Висновки за результатами аналізу	21.04-26.04.2026	Виконано
6	Оформлення пояснювальної записки	26.04-28.04.2026	Виконано
7	Розробка презентації	29.04-01.05.2026	Виконано
8	Передзахист	08.05-11.05.2026	Виконано
9	Здача в деканат	08.05-14.05.2026	Виконано

Здобувач(ка) вищої освіти _____ Дмитро МАЛІКОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ярослав ТОРОШАНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 98 стор., 39 рис., 3 табл., 30 джерел.

Мета роботи – спрощення та підвищення безпеки доступу до внутрішніх сервісів локальної мережі за рахунок використання Reverse Proxy, автоматизації HTTPS-з'єднань та контейнеризації мережевих служб.

Об'єкт дослідження – процеси організації доступу до внутрішніх веб-сервісів у локальних мережах.

Предмет дослідження – засоби та технології побудови, налаштування й оптимізації систем контролю доступу на базі Reverse Proxy, Let's Encrypt, Dynamic DNS та Docker.

Короткий зміст роботи: У роботі розглянуто теоретичні основи побудови систем централізованого доступу до внутрішніх ресурсів, принципи роботи Reverse Proxy, особливості маршрутизації HTTP/HTTPS-запитів, використання доменних імен у закритих середовищах, сервіси Dynamic DNS, механізми шифрування TLS та принципи роботи центру сертифікації Let's Encrypt. Також виконано порівняльний аналіз традиційного прокидання портів і сучасних Reverse Proxy-рішень з позиції безпеки, масштабованості та зручності адміністрування.

Особливу увагу приділено практичній реалізації системи доступу із використанням Docker Compose, Nginx Proxy Manager та DuckDNS, автоматичній генерації SSL-сертифікатів через DNS Challenge, маршрутизації запитів до внутрішніх сервісів і тестуванню HTTPS-доступу. Додатково розглянуто методи оптимізації роботи системи, перспективи впровадження автентифікації та перенесення інфраструктури на енергоефективний мікросервер.

КЛЮЧОВІ СЛОВА: REVERSE PROXY, LET'S ENCRYPT, NGINX PROXY MANAGER, DOCKER, DUCKDNS, HTTPS, TLS, ЛОКАЛЬНА МЕРЕЖА, КОНТРОЛЬ ДОСТУПУ, DNS CHALLENGE, ВНУТРІШНІ СЕРВІСИ.

ЗМІСТ

ВСТУП 8
1. ОСНОВИ МАРШРУТИЗАЦІЇ ТА БЕЗПЕКИ ВНУТРІШНІХ ВЕБ-РЕСУРСІВ ... 9
1.1 Принципи роботи технології Reverse Proxy 9
1.2 Системи доменних імен у закритих середовищах 17
1.3 Інфраструктура відкритих ключів (PKI) та SSL/TLS 25
1.4 Контейнеризація як середовище для мережевих служб 34
1.5 Загрози локальних мереж. 42
2. ПОБУДОВА СИСТЕМИ ДОСТУПУ НА БАЗІ DOCKER ТА NGINX PROXY MANAGER 49
2.1 Архітектурне проєктування захищеного доступу 49
2.2 Покрокова реалізація інфраструктури 55
2.3 Алгоритми обробки мережевих запитів 68
2.4 Практичне тестування доступу до сервісів 77
3. АНАЛІЗ ЕФЕКТИВНОСТІ ТА НАПРЯМКИ ПОКРАЩЕННЯ ПРОКСІ-СИСТЕМИ 85
3.1 Методи оптимізації роботи Nginx Proxy Manager 85
3.2 Порівняння із традиційними підходами до маршрутизації 92
3.3 Економічний ефект та вектори модернізації 98
ВИСНОВКИ 105
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 106
Додаток А. Фрагмент файлу docker-compose.yml для розгортання Nginx Proxy Manager 108
Додаток Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 109

ВСТУП

У локальних мережах внутрішні веб-сервіси дедалі частіше використовуються для адміністрування обладнання, зберігання даних, моніторингу та організації допоміжних цифрових процесів. Разом із поширенням таких ресурсів зростає потреба у безпечному, централізованому та зручному способі доступу до них без прямого використання локальних IP-адрес і великої кількості окремих портів. Одним із найбільш доцільних підходів до розв'язання цього завдання є застосування технології Reverse Proxy, що дозволяє організувати єдину точку входу до внутрішніх сервісів.

Теоретична основа дослідження охоплює принципи роботи зворотного проксі-сервера, особливості маршрутизації HTTP-запитів, передачу службових заголовків та відмінності між Forward Proxy і Reverse Proxy. Окрема увага приділяється використанню доменних імен у закритих середовищах, застосуванню Dynamic DNS-сервісів, механізмам локального резолвінгу та прив'язці доменів до приватних адрес, що формує зрозумілу структуру доступу до внутрішніх ресурсів.

Важливою складовою сучасної мережевої інфраструктури є захист переданих даних. Саме тому розглядаються принципи роботи SSL/TLS, інфраструктура відкритих ключів, функціонування центру сертифікації Let's Encrypt та методи підтвердження володіння доменом. Додатково аналізуються загрози локальних мереж, зокрема перехоплення незашифрованого трафіку.

Практична частина роботи передбачає побудову системи контролю доступу до внутрішніх сервісів на базі Docker, Nginx Proxy Manager, DuckDNS та сертифікатів Let's Encrypt. Планується створення контейнеризованого середовища, налаштування доменного доступу до локальних ресурсів, автоматична генерація HTTPS-сертифікатів через DNS Challenge, маршрутизація запитів до внутрішніх вузлів, а також тестування доступу до панелі маршрутизатора й інших сервісів через захищені доменні адреси. Результатом має стати зручна, масштабована та безпечна система централізованого доступу до ресурсів локальної мережі.

1. ОСНОВИ МАРШРУТИЗАЦІЇ ТА БЕЗПЕКИ ВНУТРІШНІХ ВЕБ-РЕСУРСІВ

1.1 Принципи роботи технології Reverse Proxy

Комп’ютерні мережі дедалі частіше будуються за принципом централізованого доступу до сервісів, коли користувач взаємодіє не безпосередньо з кінцевим ресурсом, а через проміжний вузол керування трафіком. Однією з найбільш поширених технологій такого підходу є Reverse Proxy, або зворотний проксі-сервер, використання якого стало стандартною практикою як у корпоративних мережах, так і в локальних лабораторних середовищах, де необхідно організувати зручний, контрольований та безпечний доступ до внутрішніх ресурсів. Зворотний проксі виступає єдиною точкою входу до одного або кількох внутрішніх сервісів залежно від потреб інфраструктури, приймає HTTP або HTTPS-запити від клієнтів, аналізує їх параметри та перенаправляє до відповідного цільового ресурсу. Після отримання відповіді від внутрішнього сервера проксі повертає її клієнтському пристрою, тому користувач фактично не взаємодіє напряму з реальним сервером і бачить лише адресу самого проксі-вузла.

Такий підхід дозволяє приховати внутрішню структуру мережі, централізувати обробку запитів, впровадити шифрування трафіку, балансування навантаження, кешування контенту, а також реалізувати додаткові механізми безпеки. У межах побудови систем контролю доступу до внутрішніх веб-ресурсів використання Reverse Proxy є доцільним рішенням, оскільки забезпечує роботу сервісів через доменне ім’я та HTTPS-з’єднання без необхідності відкривати реальні внутрішні адреси серверів [1].

Роль зворотного проксі у сучасних мережах. У класичному варіанті локальних мереж доступ до ресурсів часто здійснюється напряму за IP-адресою пристрою або за локальним DNS-іменем. Наприклад, веб-інтерфейс

маршрутизатора нерідко відкривається за адресою типу 192.168.1.1, мережеве сховище може працювати за 192.168.1.20, а внутрішній сервер — за іншою локальною адресою. Подібний підхід має низку недоліків, серед яких незручність використання числових IP-адрес, відсутність централізованого контролю доступу, складність керування HTTPS-сертифікатами, необхідність окремого налаштування кожного сервісу та підвищені ризики безпеки у випадках відкриття ресурсів у зовнішню мережу.

Reverse Proxy усуває значну частину цих обмежень, оскільки користувач звертається лише до одного доменного простору, а система самостійно визначає, до якого внутрішнього ресурсу необхідно перенаправити запит. Наприклад, можуть використовуватися адреси типу "router.my-local-lab.duckdns.org", "nas.my-local-lab.duckdns.org" або "panel.my-local-lab.duckdns.org", де кожне ім’я відповідає окремому сервісу, хоча фізично всі запити спочатку потрапляють на один проксі-сервер. Після цього система прозоро виконує маршрутизацію до внутрішнього ресурсу без участі користувача (див. рис. 1.1) [1].

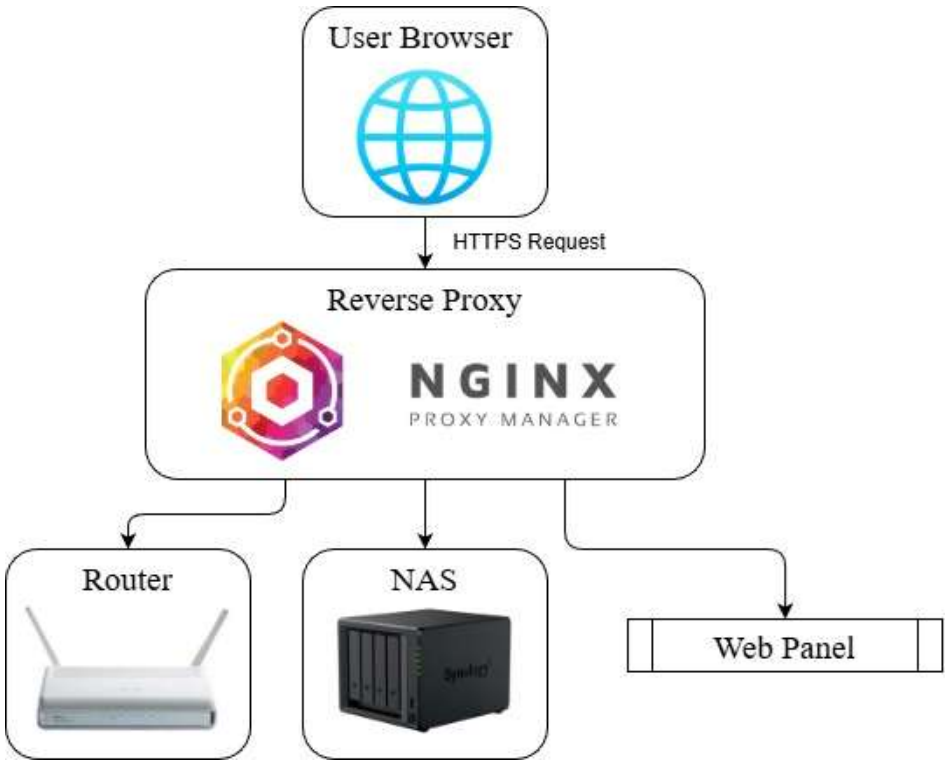


Рисунок 1.1 – Роль Reverse Proxy у централізованому доступі до сервісів

Однією з головних функцій зворотного проксі є централізована точка входу, оскільки замість десятків адрес і портів використовується один вузол доступу, що значно спрощує адміністрування мережевої інфраструктури. Іншою важливою функцією є SSL/TLS термінація, коли проксі приймає HTTPS-запити, виконує розшифрування та надалі передає трафік у внутрішню мережу через HTTP або HTTPS залежно від конфігурації. Такий підхід дозволяє зосередити керування сертифікатами в одному місці, спрощує оновлення криптографічних параметрів та зменшує навантаження на внутрішні сервіси [1].

Окреме значення має приховування внутрішньої топології мережі, оскільки користувачі не бачать реальних IP-адрес серверів, а зовнішній спостерігач не отримує інформації про структуру внутрішнього сегмента. Завдяки цьому знижується ризик сканування мережі, пошуку відкритих портів та автоматизованих спроб доступу до сервісів. Додатково Reverse Proxy дозволяє реалізувати фільтрацію доступу за [1]:

- IP-адресами;
- логінами;
- токенами;
- часовими правилами;
- іншими політиками безпеки.

Все це особливо актуально для адміністративних панелей, маршрутизаторів та службових веб-ресурсів.

Ще однією перевагою є масштабованість, оскільки до одного проксі-сервера можна поступово додавати нові сервіси без зміни клієнтських налаштувань. Для користувача залишається незмінною загальна схема доступу через доменне ім’я, а адміністратор лише додає новий маршрут або піддомен. У результаті система легко розширюється без складної реконфігурації мережі. Важливим аспектом також є журналювання та моніторинг, адже всі звернення проходять через одну контрольовану точку, де можуть зберігатися журнали доступу, статистика запитів, інформація про помилки та спроби несанкціонованого входу.

У сучасних інформаційних інфраструктурах Reverse Proxy застосовується надзвичайно широко [1]:

— дата-центрах;
— хмарних середовищах;
— контейнерних кластерах Kubernetes;
— системах відеоспостереження;
— корпоративних мережах;
— освітніх закладах;
— домашніх лабораторіях;
— малих офісах.

Популярність такого підходу пояснюється поєднанням зручності адміністрування, підвищеної безпеки та гнучкості маршрутизації, що робить зворотний проксі універсальним рішенням для організації доступу до внутрішніх веб-ресурсів.

Механізми трансляції адрес та заголовків HTTP. Одним із головних завдань зворотного проксі-сервера є коректне перетворення мережевих параметрів запиту таким чином, щоб внутрішній сервіс працював у штатному режимі та не потребував інформації про наявність проміжного вузла між клієнтом і сервером. Внутрішній ресурс у такому випадку сприймає з'єднання так, ніби взаємодія відбувається напряму, хоча фактично весь трафік проходить через Reverse Proxy. Саме завдяки цьому досягається можливість централізованого доступу до різних сервісів без зміни їх внутрішньої архітектури.

Типовий сценарій починається з того, що користувач відкриває у браузері адресу на зразок "https://router.my-local-lab.duckdns.org". Браузер виконує DNS-резолвінг і надсилає запит не до самого маршрутизатора чи іншого цільового ресурсу, а до IP-адреси проксі-сервера. Далі Reverse Proxy аналізує доменне ім'я, тип запиту, шлях URL та інші параметри, після чого виконує перенаправлення до реального внутрішнього сервера, наприклад "http://192.168.1.1". Таким чином зовнішня адреса, яку бачить користувач, та внутрішня адреса, на яку фактично передається трафік, є різними. Для клієнта реальний сервер залишається прихованим, а вся взаємодія здійснюється лише через проксі-рівень (див. рис. 1.2).



Рисунок 1.2 – Схема трансляції HTTP-заголовків через проксі

Подібний механізм є особливо корисним у випадках, коли внутрішні ресурси мають приватні IP-адреси, недоступні напряму ззовні, або коли необхідно об’єднати кілька сервісів під одним доменним простором. Користувачеві не потрібно знати, де фізично розміщений сервіс, який порт він використовує чи яка його внутрішня адресація, оскільки всі ці деталі бере на себе система маршрутизації проксі.

Окрім трансляції адрес, Reverse Proxy виконує роботу із заголовками HTTP, що є не менш важливим аспектом коректного функціонування веб-додатків. Під час передачі запиту можуть змінюватися або додаватися службові поля, які повідомляють внутрішньому серверу необхідну технічну інформацію. Одним із найважливіших є заголовок Host. Коли клієнт надсилає запит до адреси "router.my-local-lab.duckdns.org", браузер передає саме це доменне ім’я у заголовку Host. Проксі може залишити це значення незмінним або замінити його на внутрішню адресу чи інше ім’я залежно від конфігурації. Такий механізм є необхідним для веб-серверів, що використовують віртуальні хости та обслуговують декілька сайтів на одному мережевому інтерфейсі.

Широко використовується заголовок X-Forwarded-For, у якому передається реальна IP-адреса клієнта. Якщо не використовувати цей механізм, внутрішній сервер бачитиме лише адресу проксі-сервера, а не кінцевого користувача. Для журналювання, статистики, систем безпеки та обмежень доступу важливо зберігати саме реальне джерело запиту, тому Reverse Proxy додає службовий запис із фактичною адресою клієнта, наприклад "192.168.1.50" [3].

Іншим важливим параметром є заголовок X-Forwarded-Proto, який вказує оригінальний протокол з’єднання. Навіть якщо між проксі та внутрішнім сервером передача здійснюється через HTTP, система може повідомити внутрішньому застосунку, що користувач підключався саме через HTTPS. Це необхідно для коректного формування абсолютних посилань, редиректів, політик безпеки та cookie-параметрів, що вимагають захищеного каналу зв’язку [3].

Додатково часто використовується заголовок X-Real-IP, який також передає адресу користувача у більш простому форматі. Частина веб-додатків орієнтується саме на нього, тому сучасні проксі-рішення зазвичай підтримують обидва механізми одночасно [3].

Важливою функцією є також переписування URL-адрес. Наприклад, користувач відкриває адресу "https://my-local-lab.duckdns.org/admin", а Reverse Proxy перенаправляє цей запит на внутрішній ресурс "http://192.168.1.20:8080/". Такий підхід дозволяє публікувати кілька сервісів через один домен, використовуючи різні шляхи або піддомени. У результаті один сервер доступу може одночасно обслуговувати панель маршрутизатора, файлове сховище, систему моніторингу та інші веб-інтерфейси без конфлікту адрес.

Окреме значення має SSL offloading або TLS termination, коли Reverse Proxy бере на себе обробку захищеного з’єднання. Клієнт встановлює HTTPS-сеанс із проксі-сервером, а вже далі трафік передається до внутрішнього ресурсу через HTTP або HTTPS залежно від політики безпеки. Такий підхід зменшує навантаження на внутрішні сервери, оскільки їм не потрібно самостійно обробляти шифрування, а також значно спрощує централізоване керування сертифікатами. Замість налаштування TLS на кожному пристрої достатньо підтримувати актуальний сертифікат лише на проксі-вузлі [3].

У результаті механізми трансляції адрес та HTTP-заголовків дозволяють Reverse Proxy виступати інтелектуальним посередником між користувачем і сервісом, зберігаючи прозорість роботи внутрішніх систем, спрощуючи маршрутизацію та підвищуючи рівень безпеки всієї мережевої інфраструктури.

Відмінності між Forward та Reverse Proxy. Попри схожість назв, Forward Proxy та Reverse Proxy виконують принципово різні функції та працюють у протилежних напрямах мережевої взаємодії. Forward Proxy представляє інтереси клієнта, тоді як Reverse Proxy представляє інтереси сервера або групи серверів. Саме ця різниця визначає їх:

— сферу застосування;

— принцип роботи;

— місце в архітектурі мережі.

Forward Proxy встановлюється між користувачем і зовнішніми ресурсами. Клієнтський пристрій надсилає запит спочатку до проксі, а вже той звертається до потрібного сайту або сервісу від власного імені. У результаті кінцевий сервер бачить не реального користувача, а адресу проксі-сервера. Подібний підхід застосовується для [2]:

— анонімізації;

— обходу географічних обмежень;

— фільтрації доступу;

— кешування веб-сторінок;

— контролю вихідного трафіку в організаціях.

Наприклад, навчальний заклад або підприємство може забороняти доступ до певних сайтів через корпоративний проксі-сервер, який аналізує всі запити співробітників або студентів.

Reverse Proxy працює у зворотному напрямку. Користувач звертається до зовнішньо видимого сервера доступу, вважаючи його кінцевою точкою взаємодії, однак проксі перенаправляє запит до одного з внутрішніх ресурсів. Клієнт не знає реальної структури серверної частини та не бачить внутрішніх IP-адрес. Така схема широко використовується для [2]:

— захисту серверів;

— керування SSL-сертифікатами;

— балансування навантаження між кількома вузлами;

— маршрутизації до мікросервісів;

— приховування інфраструктури від зовнішнього спостереження.

Якщо умовно порівняти обидва підходи, то Forward Proxy захищає або контролює користувача під час доступу до зовнішнього світу, тоді як Reverse Proxy захищає серверну частину та контролює доступ користувачів до внутрішніх сервісів. Саме тому в задачах побудови локальної системи доступу до маршрутизатора, файлового сховища чи веб-панелі доцільно використовувати саме Reverse Proxy (див. рис. 1.3) [2].

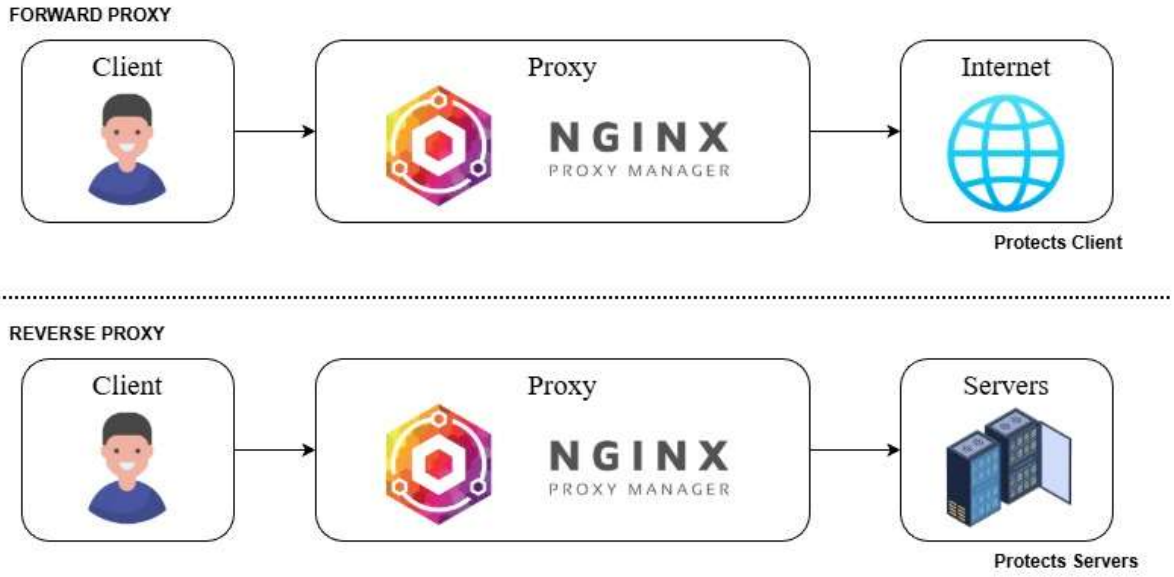


Рисунок 1.3 – Напрямки маршрутизації трафіку у різних типах проксі

У практичному сценарії користувач відкриває доменне ім’я, яке вже відоме проксі-серверу, після чого система автоматично перенаправляє запит до потрібного ресурсу всередині локальної мережі. Реальний сервер залишається прихованим, централізовано керується через SSL-сертифікацію, а нові сервіси можуть додаватися без зміни всієї мережевої структури. Достатньо створити новий маршрут або піддомен, після чого сервіс стає доступним за єдиною логікою доступу.

Reverse Proxy у сучасних мережах є фундаментальним інструментом централізованого керування сервісами, оскільки поєднує маршрутизацію, безпеку, шифрування, журналювання та зручність масштабування. На відміну від Forward

Proxy, який орієнтований переважно на клієнтську сторону, зворотний проксі орієнтований на захист і публікацію серверних ресурсів (див. табл. 1.1).

Таблиця 1.1 – Порівняльна характеристика Forward Proxy та Reverse Proxy

Параметр	Forward Proxy	Reverse Proxy
Напрямок трафіку	Від клієнта до зовнішніх ресурсів	Від клієнта до внутрішніх серверів через проксі
Призначення	Представлення клієнта у зовнішній мережі	Представлення серверів для клієнтів
Рівень приховування вузлів	Приховує клієнта	Приховує серверну інфраструктуру
Сфера застосування	Анонімізація, фільтрація доступу, кешування	Балансування навантаження, SSL, захист серверів
Адміністрування	Контроль вихідного трафіку	Централізоване управління доступом до сервісів
Безпека	Захист користувача	Захист серверів і внутрішніх ресурсів

Саме тому використання рішень на кшталт Nginx Proxy Manager є технічно обґрунтованим вибором для побудови систем контролю доступу до внутрішніх ресурсів через доменне ім’я та HTTPS-з’єднання.

1.2 Системи доменних імен у закритих середовищах

Сучасна мережева інфраструктура практично неможлива без використання систем доменних імен DNS, оскільки саме цей механізм забезпечує перетворення зрозумілих для людини назв вузлів у числові IP-адреси, необхідні для маршрутизації трафіку. Замість запам’ятовування адрес на зразок 192.168.1.1 користувач може звертатися до ресурсу за назвою "router.local", "panel.example.com" або іншими зручними ідентифікаторами. У глобальному Інтернеті DNS застосовується для:

— доступу до веб-сайтів;
— поштових сервісів;
— хмарних платформ;
— систем віддаленої роботи;
— великої кількості цифрових ресурсів.

Однак не менш важливим є використання систем доменних імен у локальних і закритих середовищах, де зручність адресації часто має не менше значення, ніж у публічних мережах.

У корпоративних мережах, домашніх лабораторіях, офісах, навчальних закладах, виробничих сегментах та тестових інфраструктурах постійно виникає потреба у зручному доступі до внутрішніх ресурсів, які фізично недоступні з глобальної мережі. До таких ресурсів можуть належати:

— панелі адміністрування маршрутизаторів;
— файлові сервери;
— NAS-сховища;
— локальні веб-додатки;
— внутрішні CRM та ERP-платформи;
— сервери розробки;
— Docker-сервіси;
— системи відеоспостереження;
— платформи моніторингу;
— контролери автоматизації будівель;
— інші служби.

Все те що працює лише в межах приватного сегмента. У подібних випадках використання публічного DNS не завжди вирішує завдання напряму, оскільки внутрішні ресурси переважно використовують приватні IP-адреси діапазонів 192.168.x.x, 10.x.x.x, 172.16.x.x – 172.31.x.x, які не маршрутизуються в Інтернеті та не можуть бути досягнуті звичайним зовнішнім шляхом.

Саме тому в закритих середовищах застосовуються спеціальні підходи до DNS-резолвінгу, серед яких [4-6]:

— локальні DNS-сервери;

— Split-Brain DNS;

— Dynamic DNS;

— гібридні схеми адресації;

— централізовані системи внутрішнього іменування.

Такі механізми дозволяють поєднати зручність роботи з доменними іменами, контроль доступу та гнучкість адміністрування без потреби розкривати внутрішню структуру мережі.

Концепція Split-Brain DNS та локального резолвінгу. Термін Split-Brain DNS, або розділений DNS, означає архітектурний підхід, за якого один і той самий домен повертає різні IP-адреси залежно від джерела запиту. Іншими словами, користувач у локальній мережі отримує внутрішню адресу ресурсу, а користувач із зовнішнього середовища — зовнішню адресу, альтернативний запис або взагалі не отримує відповіді. Наприклад, домен "example.com" для локальної мережі може вказувати на 192.168.1.1, тоді як для зовнішнього Інтернету той самий запис може бути відсутнім або вести на іншу адресу. Така схема дозволяє використовувати однакові доменні імена як усередині мережі, так і поза нею, не змінюючи клієнтські налаштування та не створюючи окремих адресних просторів для різних категорій користувачів (див. рис. 1.4) [4].

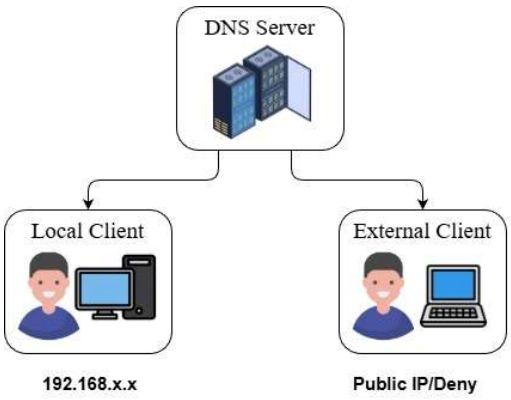


Рисунок 1.4 – Принцип роботи Split-Brain DNS

У практичному застосуванні подібна модель є особливо зручною для доступу до локальних ресурсів через зрозумілі імена на зразок "router.my-lab.duckdns.org", "nas.my-lab.duckdns.org" або "panel.my-lab.duckdns.org". Усередині домашньої чи офісної мережі такі записи можуть вказувати на приватні адреси внутрішніх серверів, тоді як із зовнішнього середовища бути недоступними або вести до захищеного шлюзу доступу. У результаті користувач працює з однаковими адресами незалежно від місця підключення, а мережеве середовище самостійно визначає, яку відповідь необхідно повернути.

Локальний резолвінг означає, що DNS-запити обробляються внутрішнім сервером у межах самої мережі. Таким сервером може виступати:

— домашній маршрутизатор;
— спеціалізований DNS-сервіс;
— шлюз фільтрації трафіку;
— окремий сервер інфраструктури.

Для цього використовуються різні програмні рішення:

— Pi-hole;
— AdGuard Home;
— Dnsmasq;
— BIND 9;
— служби DNS у Windows Server;
— вбудовані DNS-механізми мережевого обладнання.

У такій схемі клієнтський пристрій надсилає запит локальному DNS-серверу, отримує відповідь із приватною IP-адресою та встановлює з’єднання без звернення до зовнішніх резолверів.

Перевагою локального DNS є насамперед зручність користування, оскільки відпадає потреба запам’ятовувати числові адреси пристроїв. Доступ до маршрутизатора, файлового сервера чи веб-панелі здійснюється через логічне ім’я, що значно спрощує щоденну експлуатацію інфраструктури. Додатковою перевагою є незалежність від зовнішнього підключення, адже навіть за відсутності доступу до провайдера або глобального Інтернету внутрішні імена продовжують

працювати, оскільки вся система резолвінгу розташована всередині локального сегмента.

Централізоване адміністрування також є важливою властивістю такого підходу, оскільки всі записи зберігаються в одному місці. Зміна IP-адреси сервера, перенесення сервісу на інший вузол або додавання нового ресурсу не потребує переналаштування клієнтів — достатньо оновити DNS-запис. У середовищах із Reverse Proxy така модель є особливо ефективною, оскільки всі піддомени можуть спрямовуватися на єдиний проксі-сервер, а вже він виконує подальшу маршрутизацію до внутрішніх ресурсів. Наприклад, домени "router.home", "nas.home", "panel.home" можуть вести на одну адресу шлюзу доступу, де централізовано налаштовано SSL-сертифікати, політики безпеки та правила перенаправлення [4].

У практичній реалізації на маршрутизаторі або внутрішньому DNS-сервері створюються статичні записи, де:

1. "router.home" вказує на адресу маршрутизатора;
2. "proxy.home" — на проксі-сервер;
3. "nas.home" — на файлове сховище.

Після цього користувач у браузері відкриває відповідне ім’я та потрапляє до потрібного сервісу без введення IP-адреси. Такий підхід значно підвищує зручність і зменшує кількість помилок при ручному введенні адрес.

Разом із перевагами локальний резолвінг має і певні обмеження. Система працює лише всередині мережі, залежить від працездатності локального DNS-сервера, потребує ручного адміністрування записів та не завжди повноцінно підтримується бюджетними маршрутизаторами. У дешевому мережевому обладнанні функції локального DNS часто є обмеженими, що змушує використовувати окремий сервер або контейнеризоване рішення. Попри це, для домашніх лабораторій, офісів і тестових середовищ локальний DNS та Split-Brain DNS залишаються одними з найбільш практичних способів організації внутрішнього адресного простору, сумісного із сучасними системами безпечного доступу.

Використання сервісів динамічного DNS (на прикладі DuckDNS). Однією з найбільш поширених проблем домашніх та невеликих офісних мереж є використання динамічних IP-адрес, які автоматично видаються провайдером і можуть змінюватися після перезавантаження маршрутизатора, завершення оренди DHCP-сесії, технічних робіт на стороні оператора або зміни мережевої топології. На практиці це означає, що сьогодні зовнішня адреса підключення може мати один вигляд, наприклад 93.75.x.x, а через певний час уже змінитися на 178.92.x.x, де останні октети залежать від конкретної адресації провайдера. У таких умовах доступ до мережевої інфраструктури за заздалегідь відомою IP-адресою стає незручним, а в багатьох випадках взагалі втрачає сенс, оскільки адреса перестає бути сталою і її необхідно постійно перевіряти вручну. Саме для вирішення цієї проблеми використовується технологія Dynamic DNS, яка дозволяє закріпити за змінною адресою постійне доменне ім’я [6].

Dynamic DNS або DDNS являє собою механізм автоматичного оновлення DNS-записів у разі зміни поточної IP-адреси. Замість необхідності щоразу дізнаватися нову адресу користувач працює з доменним ім’ям, наприклад "my-local-lab.duckdns.org", тоді як система самостійно змінює запис у DNS при кожному оновленні зовнішнього IP. У результаті незалежно від того, яку адресу видав провайдер у конкретний момент часу, звернення до домену буде вести до актуальної точки підключення. Такий підхід значно спрощує доступ до домашніх серверів, лабораторних середовищ, систем відеоспостереження, веб-панелей маршрутизаторів, VPN-шлюзів та інших сервісів, що розташовані за динамічним підключенням.

Одним із найбільш відомих безкоштовних сервісів цього типу є DuckDNS, який побудований за максимально спрощеною моделлю роботи. Користувач реєструє власний піддомен виду "my-local-lab.duckdns.org", після чого отримує токен доступу до API та можливість автоматично оновлювати DNS-запис через:

— HTTP-запити;

— скрипти;

— клієнтські застосунки;

— контейнеризовані сервіси.

Перевагою такого рішення є:

— повна безкоштовність;

— відсутність необхідності купувати окремий домен;

— простота початкового налаштування;

— зручність інтеграції з локальною інфраструктурою.

Додатково сервіс добре поєднується з Docker-середовищами, де окремий контейнер може періодично перевіряти поточну адресу мережі та автоматично виконувати оновлення DNS без участі адміністратора [5].

Типовий алгоритм роботи виглядає наступним чином: маршрутизатор або провайдер змінює зовнішню IP-адресу підключення, після чого локальний DDNS-клієнт визначає нове значення та передає його через API до сервісу динамічного DNS. Далі DNS-запис оновлюється, і доменне ім’я починає посилатися вже на нову адресу. Для кінцевого користувача зміни відбуваються майже непомітно, оскільки використовується одна й та сама назва ресурсу. У межах домашньої лабораторії або невеликої серверної інфраструктури така модель дозволяє отримати стабільну точку входу без придбання статичної IP-адреси у провайдера.

Окремою перевагою використання DuckDNS є сумісність із механізмами SSL-сертифікації, зокрема з DNS Challenge. Завдяки цьому доменне ім’я може застосовуватися не лише як спосіб звернення до ресурсу, а і як база для отримання валідного TLS-сертифіката від центрів сертифікації. У практичному середовищі це дозволяє використовувати HTTPS навіть для внутрішніх сервісів, що особливо актуально для систем керування інфраструктурою, панелей адміністрування та веб-додатків. Користувач відкриває адресу на зразок "https://router.my-local-lab.duckdns.org", а браузер встановлює захищене з’єднання з коректним сертифікатом, навіть якщо сам ресурс фізично розташований у локальній мережі [5-6].

Прив'язка публічних доменів до приватних IP-адрес. Окремий практичний інтерес становить прив’язка публічних доменів до приватних IP-адрес. У стандартній моделі DNS-записи зазвичай ведуть на публічні маршрутизовані

адреси, однак технічно можливо створити конфігурацію, за якої доменне ім’я в локальному середовищі повертає приватну адресу внутрішнього ресурсу. Наприклад, запис "router.example.com" може вказувати на 192.168.1.1, а "nas.example.com" — на 192.168.1.20. Такий підхід працює лише для клієнтів, що знаходяться всередині локальної мережі або використовують внутрішній DNS-сервер, проте саме в цьому полягає його практична цінність. Зовнішній користувач не отримає доступу до приватної адреси, а внутрішній сегмент працюватиме через звичні доменні імена.

Потреба в такій схемі виникає тоді, коли необхідно використовувати валідне доменне ім’я, отримати SSL-сертифікат, централізувати доступ через Reverse Proxy та не відкривати порти в Інтернет. У результаті вся логіка роботи зосереджується всередині локальної інфраструктури. Не потрібно публікувати веб-інтерфейс маршрутизатора назовні, відкривати адміністративні панелі або розкривати структуру мережі. Доступ залишається внутрішнім, але здійснюється за професійною моделлю — через доменне ім’я, HTTPS та централізований шлюз маршрутизації.

Типова схема взаємодії виглядає так: клієнтський пристрій у локальній мережі надсилає DNS-запит до внутрішнього резолвера, той повертає адресу проксі-сервера, наприклад 192.168.1.100, після чого запит потрапляє на Reverse Proxy. Уже проксі аналізує доменне ім’я та перенаправляє звернення до потрібного ресурсу — маршрутизатора, файлового сховища, внутрішнього веб-сервісу чи панелі керування контейнерами. Якщо користувач вводить адресу "router.my-local-lab.duckdns.org", вона може вести не напряму до маршрутизатора, а до проксі-вузла, який уже самостійно визначає цільовий внутрішній сервер. Такий підхід значно гнучкіший за пряме зв’язування домену з кожним окремим пристроєм.

Переваги цієї моделі очевидні: локальні сервіси отримують HTTPS-захист, користувач працює з одним доменним простором, можна створювати багато піддоменів для різних ресурсів, відсутня необхідність відкривати зовнішні порти, а структура мережі прихована від сторонніх осіб. Додавання нового сервісу зводиться до створення ще одного DNS-запису та правила у проксі-сервері.

Наприклад, "router.domain", "nas.domain", "panel.domain", "camera.domain" можуть працювати через єдину точку входу без зміни архітектури мережі [4-6].

Разом із перевагами існують і певні обмеження. Подібна схема працює лише всередині локального сегмента або через VPN-підключення до нього. Користувач із відкритого Інтернету не зможе напряму досягти приватної адреси, що одночасно є і технічним обмеженням, і перевагою з точки зору безпеки. Необхідно підтримувати працездатність локального DNS, коректно налаштовувати маршрутизацію та контролювати внутрішні правила доступу. За правильної реалізації система функціонує стабільно та забезпечує значно вищий рівень зручності порівняно з використанням лише IP-адрес.

У підсумку системи доменних імен у закритих середовищах формують важливу основу сучасної локальної інфраструктури. Split-Brain DNS дозволяє розділяти зовнішній і внутрішній простір імен, локальний резолвінг забезпечує швидкий доступ до ресурсів усередині мережі, а Dynamic DNS-сервіси на зразок DuckDNS дають змогу використовувати стабільне доменне ім’я навіть за динамічної адресації провайдера. У поєднанні з Reverse Proxy створюється зручна, масштабована та безпечна система доступу до внутрішніх ресурсів без необхідності відкривання сервісів у зовнішній мережі.

1.3 Інфраструктура відкритих ключів (PKI) та SSL/TLS

В комп’ютерних мережах питання захисту передаваних даних належить до базових вимог функціонування будь-якої інформаційної системи, причому така потреба існує не лише в глобальному Інтернеті, а й у межах локального середовища. Дані, що передаються між клієнтом і сервером, можуть бути перехоплені, змінені або використані сторонніми особами, якщо канал зв’язку не має криптографічного захисту. Особливо актуальною ця проблема є для:

— веб-інтерфейсів маршрутизаторів;

— систем керування інфраструктурою;

— файлових серверів;

— контейнерних панелей адміністрування;

— внутрішніх порталів підприємств;

— інших сервісів.

У всіх них використовуються облікові записи, службові параметри або конфіденційна інформація.

Якщо обмін відбувається у відкритому вигляді через протокол HTTP, зловмисник у межах тієї ж мережі потенційно може перехопити:

— логіни;

— паролі;

— cookie-файли;

— токени сесій;

— конфігураційні дані;

— іншу службову інформацію.

Навіть у домашньому середовищі ризик не зникає повністю, оскільки джерелом загрози можуть бути:

— заражені пристрої;

— гостьові підключення до безпровідної мережі;

— помилково довірені вузли;

— навмисне втручання в локальний трафік.

Саме тому сучасним стандартом мережевої взаємодії стало використання шифрованих з’єднань на основі SSL/TLS та цифрових сертифікатів.

Основою довіри в мережевих середовищах виступає PKI — Public Key Infrastructure, тобто інфраструктура відкритих ключів. Вона забезпечує [7]:

1. Механізми автентифікації серверів.
2. Перевірки справжності цифрових сертифікатів.
3. Побудови довірчих відносин між користувачем і веб-ресурсом.

4. Безпечне використання криптографії з відкритими та закритими ключами.

Завдяки PKI браузер може перевірити, чи дійсно користувач підключається до потрібного сервера, а не до підробленого вузла, який імітує справжній ресурс.

У межах системи контролю доступу на базі Reverse Proxy тема PKI має особливе значення, оскільки доступ до локальних сервісів повинен здійснюватися через HTTPS із валідним сертифікатом без попереджень браузера про небезпечне з'єднання, прострочений сертифікат або самопідписаний підпис. Для практичної реалізації локальної інфраструктури цього типу важливо, щоб користувач отримував звичний сучасний рівень безпеки незалежно від того, чи йдеться про публічний сайт, чи про внутрішню панель маршрутизатора або домашній сервер.

Протоколи захисту транспортного рівня. SSL та його сучасна заміна TLS являють собою транспортні криптографічні протоколи, призначені для безпечної передачі даних між клієнтом і сервером. Історично використовувався SSL, однак сьогодні він вважається застарілим через наявність відомих вразливостей та практично не застосовується в актуальних системах. Натомість використовується TLS у сучасних версіях, які забезпечують вищий рівень захисту, продуктивності та сумісності з новими криптографічними алгоритмами.

Для користувача різниця між незахищеним і захищеним з'єднанням найчастіше проявляється у вигляді HTTP та HTTPS. Адреса "http://local" означає передавання даних без шифрування, тоді як "https://local" використовує TLS-з'єднання з перевіркою сертифіката та захистом трафіку. У браузері така взаємодія зазвичай відображається значком замка або відповідним індикатором захищеного каналу [8].

TLS виконує три фундаментальні функції. Першою є конфіденційність, оскільки дані шифруються і сторонній спостерігач не може прочитати зміст переданої інформації навіть у разі перехоплення пакетів. Другою є цілісність, адже застосовуються криптографічні механізми перевірки, які унеможливлюють непомітну зміну пакетів під час передавання. Третьою є автентичність, тобто клієнт

отримує підтвердження, що з’єднання встановлено саме з потрібним сервером, а не з вузлом-підробкою.

Робота TLS-з’єднання починається з процедури "handshake" — узгодження параметрів безпечної сесії між клієнтом і сервером. У спрощеному вигляді процес виглядає так: браузер надсилає запит "Client Hello" із підтримуваними алгоритмами шифрування, сервер відповідає "Server Hello", передає цифровий сертифікат та параметри сесії, клієнт перевіряє дійсність сертифіката, після чого сторони погоджують спільний сесійний ключ і переходять до захищеного обміну даними. Усе це відбувається автоматично за долі секунди та зазвичай непомітно для користувача (див. рис. 1.5) [8].

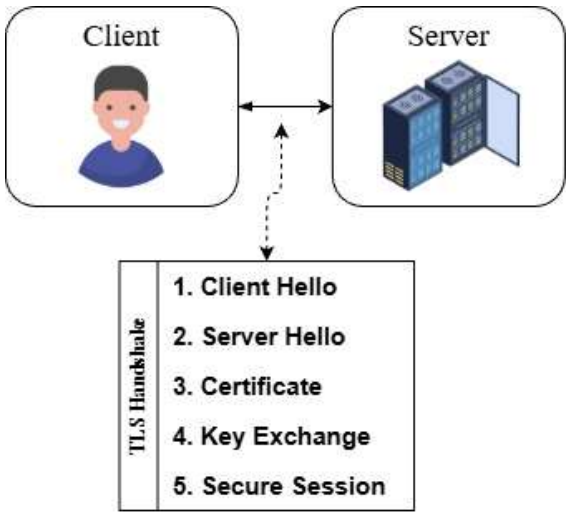


Рисунок 1.5 – Етапи встановлення TLS-з’єднання

У межах TLS застосовуються два криптографічні підходи. Перший — асиметрична криптографія, де використовується пара ключів [8]:

— відкритий;

— приватний.

Такий механізм потрібний для перевірки:

— сертифіката;

— цифрового підпису;

— безпечного погодження параметрів сеансу.

Відкритий ключ може бути доступний будь-кому, тоді як приватний зберігається лише на сервері. Другий підхід — симетрична криптографія, де використовується один спільний сесійний ключ для швидкого шифрування трафіку після встановлення з'єднання. Саме комбінація цих двох методів дозволяє поєднати безпеку та високу швидкодію.

Помилковим є уявлення, що локальна мережа автоматично є безпечною лише тому, що ізольована від Інтернету. Насправді ризики існують і в приватному сегменті. Серед типових прикладів можна назвати:

1. підключення стороннього пристрою до Wi-Fi;
2. заражений ноутбук у мережі;
3. перехоплення ARP-трафіку;
4. атаки типу Man-in-the-Middle;
5. зчитування паролів із незахищених HTTP-сесій;
6. експлуатацію слабких налаштувань маршрутизатора.

Саме тому навіть панель адміністрування домашнього роутера доцільно відкривати через HTTPS, оскільки вона містить усі критичні параметри мережевої інфраструктури.

У системах на базі Reverse Proxy, зокрема при використанні Nginx Proxy Manager, TLS зазвичай завершується на проксі-сервері. Користувач встановлює HTTPS-з'єднання з проксі-вузлом, браузер бачить валідний сертифікат, а вже далі проксі передає трафік до внутрішнього сервісу через HTTP або HTTPS залежно від архітектури мережі. Така схема спрощує адміністрування, оскільки сертифікати потрібно підтримувати лише в одній точці, а внутрішні сервіси не зобов'язані мати власні окремі налаштування SSL.

У результаті використання TLS у локальному середовищі забезпечується сучасний рівень захисту без ускладнення інфраструктури. Користувач отримує довірене з'єднання без попереджень браузера, паролі та службові дані передаються у зашифрованому вигляді, а адміністрування сервісів стає централізованим і значно зручнішим. Саме тому впровадження PKI та TLS у внутрішніх мережах сьогодні

розглядається не як додаткова опція, а як базова вимога безпечної роботи цифрових сервісів.

Принципи роботи центру сертифікації Let's Encrypt. Центр сертифікації CA, або Certificate Authority, являє собою довірену організацію, яка випускає цифрові сертифікати для доменів і серверів, підтверджуючи справжність ресурсу та забезпечуючи основу довіри в сучасних мережах. Сертифікат містить інформацію про:

— доменне ім’я;
— відкритий ключ сервера;
— строк дії;
— технічні параметри шифрування;
— цифровий підпис центру сертифікації.

Операційні системи, браузери та мобільні платформи попередньо містять перелік довірених кореневих сертифікатів, тому під час відкриття захищеного сайту клієнтський пристрій може автоматично перевірити, чи дійсно сертифікат виданий легітимною організацією, чи не був відкликаний та чи відповідає доменному імені, до якого здійснюється підключення. Завдяки такому механізму користувач отримує підтвердження, що з’єднання встановлюється саме з потрібним сервером, а не з підробленим вузлом.

Особливе місце серед сучасних центрів сертифікації займає Let's Encrypt, створений із метою масового впровадження HTTPS та спрощення доступу до криптографічного захисту для власників сайтів, серверів і внутрішніх сервісів. Основна ідея цього проєкту полягає у тому, що захищене з’єднання не повинно бути дорогим або складним у впровадженні. Саме тому видача сертифікатів здійснюється безкоштовно, автоматизовано та без необхідності проходити складні бюрократичні процедури. Такий підхід суттєво змінив практику адміністрування веб-ресурсів, оскільки раніше сертифікати часто вимагали оплати, ручного замовлення та складного налаштування [9].

Серед основних переваг Let's Encrypt можна виділити [9]:

1. безкоштовну видачу сертифікатів;

2. автоматичне продовження строку дії;
3. підтримку wildcard-сертифікатів;
4. широку сумісність із браузерами;
5. високу популярність у світі.

Завдяки довірі з боку сучасних клієнтських систем сертифікати цього центру коректно розпізнаються більшістю браузерів без додаткових дій з боку користувача. У результаті навіть невелика домашня лабораторія або локальний сервер можуть використовувати той самий рівень захисту, що й великі комерційні ресурси.

Процес видачі сертифіката в спрощеному вигляді складається з кількох етапів. Спочатку сервер або програмний клієнт генерує криптографічну пару ключів — приватний та відкритий. Далі формується запит на сертифікат, який надсилається до центру сертифікації. Після цього система перевіряє, чи заявник дійсно контролює домен, на який запитується сертифікат. Якщо перевірка проходить успішно, формується та видається готовий сертифікат, який встановлюється на сервері. У разі невдалої перевірки випуск блокується, що захищає домени від несанкціонованого використання сторонніми особами (див. рис. 1.6) [9].



Рисунок 1.6 – Схема роботи центру сертифікації Let’s Encrypt

Для автоматизації всіх цих процедур використовується стандарт ACME — Automatic Certificate Management Environment. Саме він дозволяє програмним клієнтам без ручного втручання створювати запити, проходити перевірку домену, отримувати сертифікати та продовжувати їх до завершення строку дії. Через ACME працює велика кількість сучасних інструментів, серед яких:

— Certbot;
— Nginx Proxy Manager;
— Caddy;
— Traefik.

У практичному середовищі це означає, що адміністратору не потрібно вручну перевипускати сертифікати або виконувати регулярні технічні дії, оскільки весь життєвий цикл документа може обслуговуватися автоматично.

Сертифікати Let's Encrypt видаються на 90 днів, і такий відносно короткий строк обраний навмисно. По-перше, це зменшує ризики у випадку компрометації ключів або неправильного використання сертифіката. По-друге, стимулюється автоматизація, оскільки ручне продовження кожні три місяці є незручним, а отже більшість систем переходять на сучасні автоматичні механізми керування. По-третє, це дозволяє підтримувати актуальні криптографічні параметри та швидше відмовлятися від застарілих алгоритмів [9].

Для локальної інфраструктури така модель є особливо корисною. У системі контролю доступу користувач очікує, що адреса на зразок "https://router.my-local-lab.duckdns.org" відкриється без попереджень браузера, без повідомлень про самопідписаний сертифікат або ризики безпеки. Самопідписані сертифікати технічно можуть працювати, однак викликають недовіру з боку клієнтських систем і створюють додаткові труднощі. Використання валідного сертифіката від загальновизнаного центру сертифікації повністю усуває ці проблеми та наближає локальний сервіс до стандартів публічних веб-ресурсів.

Методи валідації доменів (HTTP-01 проти DNS-01). Перед видачею сертифіката центр сертифікації повинен переконатися, що заявник дійсно контролює відповідний домен. Без такого механізму будь-яка особа могла б

отримати сертифікат на чужий сайт, що зробило б всю модель довіри непридатною. Для цього використовуються challenge-перевірки, серед яких найбільш поширеними є:

— HTTP-01;

— DNS-01.

У варіанті HTTP-01 система просить розмістити спеціальний файл за службовою адресою на веб-сервері, після чого центр сертифікації намагається отримати його через Інтернет. Якщо файл доступний і містить правильний маркер, домен вважається підтвердженим [10].

Перевагою HTTP-01 є:

— простота реалізації;

— відсутність потреби змінювати DNS;

— широка підтримка різними сервісами.

Разом із тим цей спосіб вимагає доступності сервера з Інтернету по порту 80 або можливості зовнішнього звернення до ресурсу. Для повністю закритих локальних сервісів така вимога часто є неприйнятною, оскільки суперечить самій ідеї прихованої внутрішньої інфраструктури.

Альтернативою виступає DNS-01 validation, де підтвердження права на домен здійснюється через створення TXT-запису в DNS-зоні. Наприклад, у зоні домену формується запис "_acme-challenge.example.com", після чого центр сертифікації перевіряє його наявність та вміст. Якщо запис коректний, домен підтверджується і сертифікат може бути виданий. Такий спосіб не потребує відкриття портів, сервер може бути повністю закритим від зовнішнього доступу, а сама перевірка працює навіть у мережах із NAT або CGNAT, що особливо важливо для домашніх провайдерів [10].

Додатковою перевагою DNS-01 є підтримка wildcard-сертифікатів, наприклад "*.example.com", що дозволяє одним сертифікатом захистити велику кількість піддоменів. Для інфраструктури з Reverse Proxy це особливо зручно, оскільки адреси типу "router.domain", "nas.domain", "panel.domain", "cloud.domain" можуть працювати в межах одного доменного простору та одного сертифіката.

Недоліками такого підходу є потреба доступу до DNS-провайдера, дещо складніша автоматизація та залежність від API конкретного сервісу [10].

У межах розглядуваної системи контролю доступу внутрішні сервіси не повинні відкриватися назовні, тому HTTP-01 суперечить базовій концепції безпеки. DNS-01, навпаки, дозволяє не виконувати port forwarding, не відкривати порт 80, залишити ресурс повністю внутрішнім, отримати валідний HTTPS-сертифікат і автоматизувати весь процес через API сервісів на кшталт DuckDNS. Типова схема виглядає так [10]:

1. Nginx Proxy Manager створює запит на сертифікат.
2. Через API формує TXT-запис у DNS.
3. Let's Encrypt перевіряє запис.
4. Після успішної перевірки видає сертифікат.

У підсумку інфраструктура відкритих ключів PKI формує фундамент довіри в сучасних мережах, забезпечує автентичність серверів і безпечне шифрування трафіку через TLS. Let's Encrypt зробив використання HTTPS масовим і безкоштовним, а автоматизація через ACME значно спростила адміністрування сертифікатів. Для локальних рішень, де сервіси не повинні бути доступними з Інтернету, найбільш доцільним методом перевірки домену є саме DNS-01, тому поєднання Nginx Proxy Manager, DuckDNS та Let's Encrypt є технічно обґрунтованим рішенням для побудови сучасної системи захищеного доступу до внутрішніх ресурсів [7-10].

1.4 Контейнеризація як середовище для мережевих служб

В інформаційних системах дедалі частіше застосовується підхід, за якого прикладні сервіси запускаються не безпосередньо в основній операційній системі, а в окремих ізольованих програмних середовищах, що отримали назву контейнери. Така модель розгортання стала логічним продовженням класичної віртуалізації,

однак на відміну від повноцінних віртуальних машин контейнери використовують спільне ядро операційної системи, завдяки чому потребують значно менше ресурсів, швидше запускаються та простіше адмініструються. Саме тому контейнеризація активно застосовується у веб-розробці, DevOps-практиках, хмарних платформах, тестових середовищах і локальних мережевих лабораторіях.

Для задач побудови системи контролю доступу на базі Reverse Proxy контейнерний підхід є особливо доцільним, оскільки дозволяє швидко розгорнути окремі сервіси, серед яких:

— Nginx Proxy Manager;
— база даних конфігурацій;
— допоміжний DNS-клієнт;
— тестові веб-ресурси;
— інші компоненти.

Кожен із них функціонує незалежно від інших, має власні параметри запуску, власне мережеве середовище та може оновлюватися без впливу на суміжні елементи системи. У результаті формується гнучка архітектура, придатна як для навчальних лабораторій, так і для практичного використання у малих офісах або домашній інфраструктурі.

Контейнеризація особливо корисна саме для мережевих служб, оскільки дозволяє чітко контролювати:

— відкриті порти;
— DNS-налаштування;
— правила взаємодії між застосунками;
— маршрути трафіку;
— розмежування доступу.

Наприклад, один контейнер може обробляти HTTPS-запити, другий зберігати конфігурацію, третій виконувати функції динамічного оновлення DNS, а четвертий бути внутрішнім веб-сервісом, доступ до якого надається лише через проксі-сервер. Такий підхід значно спрощує масштабування, резервування та подальшу модернізацію інфраструктури.

Базові концепції ізоляції за допомогою Docker. Контейнер являє собою ізольоване середовище виконання застосунку, що містить:

— саму програму;

— необхідні бібліотеки;

— залежності;

— конфігураційні файли;

— власні параметри мережевої взаємодії.

На відміну від традиційного встановлення програм у систему, де всі служби використовують спільні бібліотеки та можуть конфліктувати між собою, контейнер запускає застосунок у відокремленому середовищі, що значно підвищує стабільність роботи [11].

Типова архітектура виглядає таким чином: фізичний комп’ютер або сервер забезпечує апаратну основу, поверх нього працює операційна система, далі встановлюється Docker Engine як середовище керування контейнерами, після чого запускаються окремі ізольовані служби. На практиці такими контейнерами можуть бути [11]:

— Nginx Proxy Manager;

— MariaDB;

— веб-сервер Nginx;

— Portainer;

— внутрішні тестові сервіси;

— інші допоміжні компоненти.

Механізм ізоляції Docker базується на функціональності ядра Linux. Одним із основних інструментів є "namespaces", що розділяють системні ресурси між контейнерами. Завдяки цьому кожен контейнер має:

— власний список процесів;

— власний стек мережевих інтерфейсів;

— власну файлову структуру;

— власне ім’я вузла.

Контейнер бачить лише свою внутрішню логіку та не сприймає інші сервіси як частину власного середовища (див. рис. 1.7) [11].

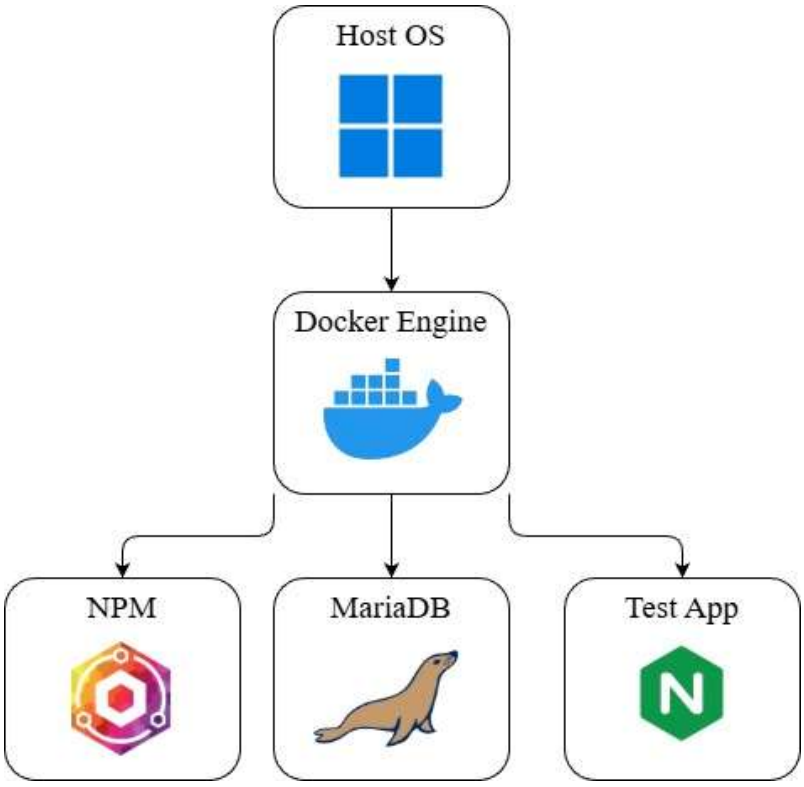


Рисунок 1.7 – Ізоляція мережевих служб у контейнерах

Іншим важливим механізмом є "control groups" або "cgroups", які відповідають за обмеження використання ресурсів. За їх допомогою можна задати, скільки оперативної пам’яті дозволено контейнеру, яку частину процесорного часу він може використовувати, як обмежується мережева пропускна здатність або дискові операції. Такий підхід особливо корисний у випадках, коли на одному пристрої працює кілька сервісів одночасно [11].

Файлова структура контейнерів часто базується на "overlay filesystem", де образ формується з декількох шарів. Завдяки цьому завантаження образів, оновлення та дублювання конфігурацій відбуваються швидко, а повторне використання спільних шарів економить дисковий простір. Якщо необхідно створити новий контейнер на базі вже існуючого образу, система не дублює весь вміст повністю, а використовує наявні шари [11].

Практична користь ізоляції полягає в тому, що збій одного контейнера не обов'язково впливає на роботу інших. Якщо перестав працювати DNS-клієнт або база даних, сам Reverse Proxy може продовжувати обробку запитів до моменту втручання адміністратора. Також контейнер легко зупинити, видалити та запустити повторно без змін у базовій операційній системі. Така модель істотно спрощує тестування нових рішень та повернення до стабільної конфігурації.

У системі контролю доступу можуть використовуватися окремі контейнери для:

— проксі-сервера;

— бази даних конфігурацій;

— автоматичного оновлення DNS;

— тестового веб-сервісу.

Логічне розділення підвищує керованість і дозволяє замінювати окремі частини інфраструктури без повної перебудови всієї системи.

Управління мережами в Docker (Bridge, Host, Macvlan). Якщо розглядати мережевий стек Docker, то саме він є одним із найважливіших компонентів під час запуску мережевих служб, оскільки контейнер повинен отримати доступ до мережі, портів, DNS-резолвінгу, інших контейнерів та зовнішніх ресурсів. Будь-який сервіс, що працює у контейнері, потребує коректної взаємодії з іншими елементами інфраструктури, а тому спосіб підключення контейнера до мережі безпосередньо впливає на доступність, безпеку, продуктивність і простоту адміністрування. Docker підтримує декілька режимів мережевої роботи, серед яких найбільш поширеними є Bridge, Host та Macvlan, і кожен із них має власні особливості використання.

Bridge-мережа використовується як стандартний режим за замовчуванням. Після встановлення Docker створює внутрішній віртуальний інтерфейс docker0, що фактично є програмним комутатором, до якого підключаються контейнери. У такій схемі один контейнер може отримати адресу на зразок 172.18.0.2, інший 172.18.0.3, після чого між ними забезпечується внутрішній обмін даними без прямої участі зовнішньої локальної мережі. Сам хост виступає шлюзом між контейнерним

сегментом та фізичною мережею. Для доступу до сервісу ззовні застосовується проброс портів, коли порт комп'ютера зв'язується з портом контейнера, наприклад "81:81", "80:80" або "443:443" [12].

Перевагою Bridge-режиму є добра ізоляція контейнерів, оскільки вони працюють у власному сегменті та не з'являються як окремі вузли локальної мережі. Також забезпечується зручна взаємодія між контейнерами через внутрішні DNS-імена, безпечна модель роботи за замовчуванням і повна підтримка Docker Compose, де мережа часто створюється автоматично без додаткового втручання. Саме тому для системи на базі Nginx Proxy Manager найчастіше використовується саме Bridge-мережа, коли проксі-сервер працює у контейнері, а назовні відкриваються лише потрібні порти. Недоліками такого режиму є необхідність ручного пробросу портів, подвійна маршрутизація через хостову систему та незначний додатковий мережевий оверхед, хоча для більшості локальних задач це не створює критичних обмежень [12].

Режим Host працює за іншим принципом, оскільки контейнер у такому випадку використовує мережевий стек хостової операційної системи напряму. Фактично застосунок у контейнері слухає реальні мережеві інтерфейси комп'ютера, а не віртуальний сегмент Docker. Якщо всередині контейнера запускається веб-сервер на порту 80, то він буде доступний як служба самої операційної системи. Такий підхід усуває проміжний рівень NAT та зменшує затримки, що корисно для високонавантажених або низькорівневих мережевих сервісів [12].

Перевагами Host-режиму є максимальна продуктивність, відсутність додаткової трансляції адрес, менші затримки та простота роботи із сервісами, які повинні напряму бачити інтерфейси системи. Саме тому такий режим доцільно використовувати для DNS-серверів, VPN-шлюзів, мережевих сканерів, IDS/IPS-систем або проксі-рішень з великим потоком трафіку. Водночас знижується рівень ізоляції, виникає ризик конфлікту портів з іншими службами хоста та загалом зменшується контроль над мережевим розмежуванням контейнерів. Якщо на

комп’ютері вже використовується порт 443, другий контейнер із тим самим портом запустити не вдасться [12].

Ще одним режимом є Macvlan, який вважається більш спеціалізованим варіантом. У цьому випадку контейнер отримує власну MAC-адресу та окрему IP-адресу з локальної фізичної мережі, тому для маршрутизатора, комутатора та інших пристроїв він виглядає як самостійний окремий вузол. Наприклад, контейнер із проксі-сервером може мати адресу 192.168.1.150, а контейнер із DNS-службою 192.168.1.151, при тому що сам хост працює з іншою адресою. Для користувача або мережевого обладнання такі контейнери сприймаються як повноцінні сервери [12].

Перевагами Macvlan є наявність окремої IP-адреси, повна інтеграція у LAN-сегмент, зручність для застарілих сервісів, які очікують окремий мережевий вузол, а також можливість використовувати контейнер як незалежний сервер без додаткового пробросу портів. Для системи на базі Nginx Proxy Manager такий підхід корисний у випадках, коли потрібно виділити проксі окремою адресою, наприклад 192.168.1.100, і звертатися до нього напряму як до автономного пристрою. Недоліками є складніше налаштування, вища вимогливість до знань адміністратора, залежність від підтримки мережевого обладнання та додаткові обмеження деяких платформ віртуалізації [12].

Вибір між Bridge, Host та Macvlan залежить від конкретного сценарію використання. Якщо пріоритетом є простота, безпечна ізоляція та сумісність із Docker Compose, найчастіше обирається Bridge. Якщо необхідна максимальна продуктивність і прямий доступ до мережі, доцільним є Host. Якщо контейнер повинен існувати як окремий вузол локальної мережі з власною адресою, найбільш логічним рішенням стає Macvlan. Саме гнучкість цих режимів робить Docker зручним середовищем для розгортання сучасних мережевих служб, оскільки одна й та сама система може бути адаптована під різні технічні вимоги без зміни самої прикладної логіки сервісів.

Використання Docker Desktop для локальної розробки. Docker Desktop є програмною платформою для Windows та macOS, що дозволяє запускати Docker-контейнери на локальному комп’ютері з використанням графічного інтерфейсу та

вбудованих інструментів адміністрування. Для навчального, домашнього або лабораторного середовища такий варіант є одним із найзручніших способів роботи з контейнерами без потреби окремого Linux-сервера, хоча за необхідності ту саму конфігурацію легко перенести на іншу систему, де вона буде працювати аналогічним чином (див. рис. 1.8).

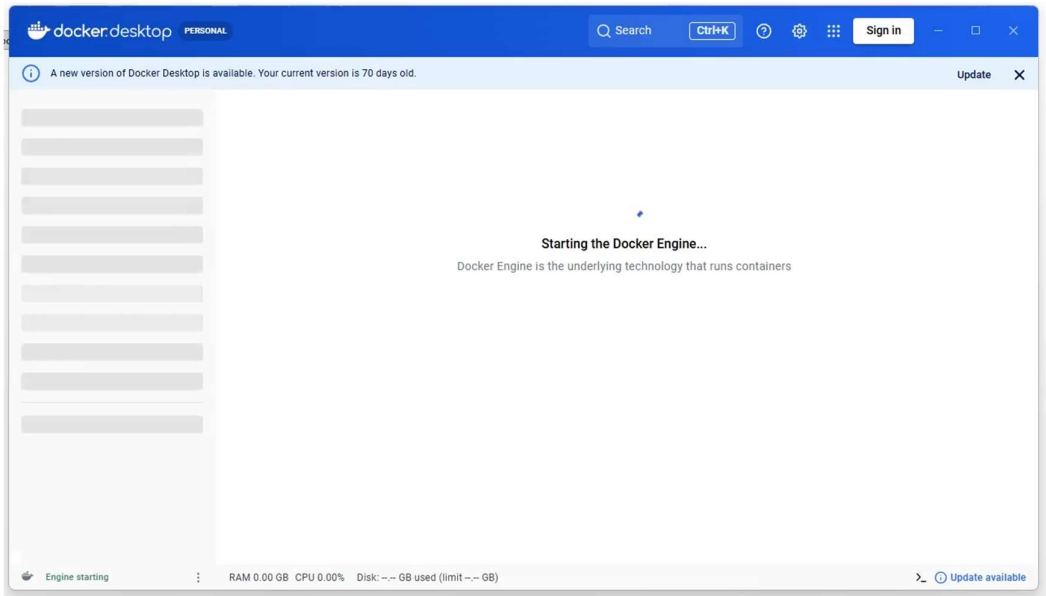


Рисунок 1.8 – Зовнішній вигляд Docker Desktop

Серед основних переваг Docker Desktop варто виділити спрощене встановлення Docker Engine без ручного налаштування служб Linux, наявність графічного інтерфейсу для перегляду контейнерів, журналів подій, використання ресурсів, образів і томів зберігання даних. Окремою перевагою є підтримка Docker Compose, що дозволяє запускати багатокомпонентні системи одним конфігураційним файлом та однією командою. Для середовища Windows додатково використовується інтеграція з WSL2, завдяки чому Linux-контейнери працюють значно ефективніше [13].

З практичної точки зору для реалізації системи контролю доступу достатньо встановити Docker Desktop, після чого створити файл docker-compose.yml, у якому буде описано всі потрібні сервіси. Наприклад, один контейнер може містити Nginx Proxy Manager, другий MariaDB, третій DuckDNS-клієнт, четвертий тестовий веб-

сервіс. У межах цього ж файлу визначаються залежності між сервісами, мережі, томи збереження даних, змінні середовища та проброс портів. Після запуску вся архітектура формується автоматично в межах єдиної конфігурації [13].

Особливо важливою перевагою є повторюваність середовища. Один і той самий compose-файл дозволяє відтворити інфраструктуру практично на будь-якому сумісному комп'ютері. Якщо необхідно перенести систему на інший ПК або сервер, достатньо скопіювати конфігурацію та, за потреби, перенести томи з даними. У результаті вся структура сервісів буде відновлена майже без ручного налаштування [13].

Корисним є і безпечне тестування, оскільки помилки всередині контейнера не пошкоджують основну операційну систему хоста. Якщо контейнер налаштовано некоректно, його можна швидко видалити та створити повторно. Такий підхід зручний для:

— експериментів;

— навчання;

— демонстраційних стендів;

— поетапного розвитку локальної мережевої інфраструктури.

Серед можливих обмежень слід враховувати:

1. споживання оперативної пам'яті;
2. необхідність увімкненої апаратної віртуалізації в BIOS/UEFI;
3. дещо нижчу продуктивність на слабких комп'ютерах.

Водночас для задач Reverse Proxy, локального DNS, тестових веб-сервісів чи невеликих лабораторних середовищ таких ресурсів зазвичай більш ніж достатньо.

1.5 Загрози локальних мереж

Локальні комп'ютерні мережі тривалий час сприймалися як відносно безпечне середовище, у якому внутрішні сервіси могли працювати без шифрування

трафіку, без багаторівневої автентифікації та без детального контролю доступу. Такий підхід базувався на припущенні, що внутрішній сегмент мережі нібито ізольований від зовнішніх загроз, а всі пристрої всередині є довіреними. У сучасних умовах подібна модель безпеки вже не відповідає реальності, оскільки до локальної мережі одночасно можуть підключатися персональні ноутбуки, смартфони, IoT-пристрої, гостьові клієнти Wi-Fi, техніка з застарілими прошивками, а також заражені системи, які стали носіями шкідливого програмного забезпечення. Навіть один компрометований пристрій усередині сегмента здатний створити серйозну загрозу для всієї інфраструктури.

Основні ризики локальних мереж пов’язані з:

1. відсутністю шифрування трафіку;
2. використанням застарілих протоколів;
3. типовими або слабкими паролями;
4. надмірною довірою до будь-якого вузла в межах сегмента;
5. відсутністю журналювання подій;
6. можливістю перехоплення пакетів;
7. несвоєчасним оновленням мережевого обладнання;
8. використанням відкритих або слабо захищених безпровідних мереж.

Усе це створює умови, за яких внутрішні веб-ресурси можуть бути атаковані навіть без прямого доступу з Інтернету. Якщо панель керування маршрутизатором, система моніторингу або локальний веб-сервіс доступні напряму через HTTP, виникає потенційна точка компрометації, що потребує усунення через сучасні механізми захисту.

Аналіз вразливостей сервісів без шифрування трафіку. Однією з найбільш поширених проблем є використання незахищеного протоколу HTTP, який передає дані у відкритому вигляді. Це означає, що будь-яка інформація між клієнтом і сервером може бути перехоплена пристроєм, який має доступ до мережевого трафіку. До такої інформації належать:

— логіни;

— паролі;

— cookie-файли сесій;
— службові заголовки;
— структура запитів;
— конфігураційні параметри;
— інші критичні дані.

Якщо у локальній мережі присутній зловмисник або заражений пристрій, отримання таких даних не становить значної складності.

Типовими прикладами незахищених локальних сервісів часто залишаються:

— веб-інтерфейси маршрутизаторів;
— IP-камери;
— DVR та NVR-системи;
— мережеві принтери;
— старі системи автоматизації будинку;
— тестові панелі адміністрування;
— внутрішні веб-додатки.

У багатьох випадках такі рішення створювалися з орієнтацією на зручність, а не на безпеку, тому використовують стандартні облікові записи або застарілі протоколи доступу.

Серед ризиків відкритого трафіку перш за все виділяється перехоплення облікових даних. Якщо пароль передавався через HTTP, його можна зчитати з мережевого дампу у відкритому вигляді. Не менш небезпечним є викрадення сесії, коли пароль може не бути видимим напряму, але зловмисник перехоплює cookie-файл авторизації та отримує доступ до вже активної сесії. У такому випадку навіть двофакторна автентифікація не завжди допомагає, оскільки атака відбувається вже після успішного входу користувача. Також можливе підмінювання вмісту сторінки, коли сторонній вузол змінює HTML або JavaScript-код у відповіді сервера, що відкриває шлях до фішингу або зараження браузера. Додатковою проблемою є відсутність перевірки сервера, оскільки клієнт не має гарантії, що підключення відбувається саме до справжнього пристрою (див. рис. 1.9).

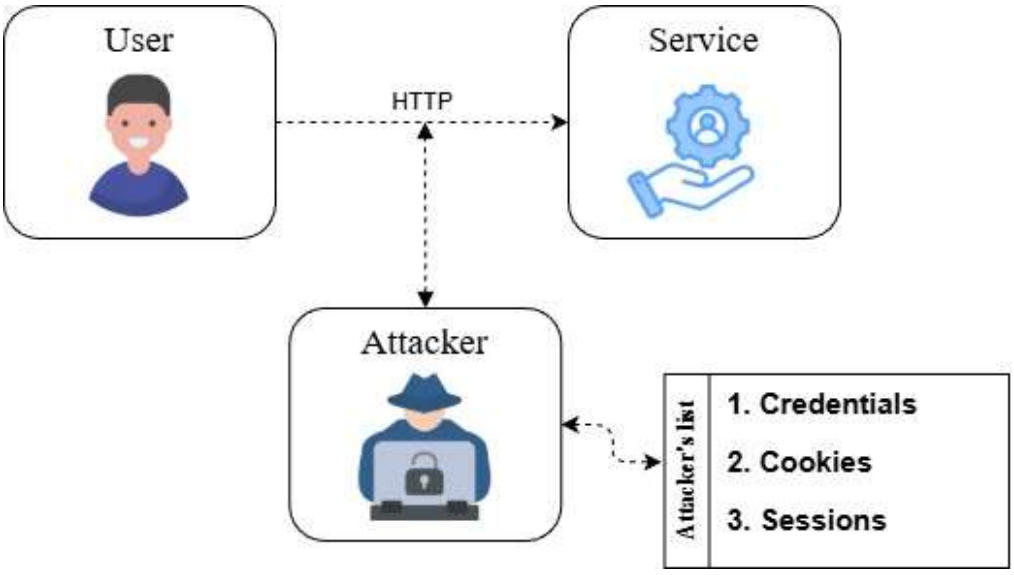


Рисунок 1.9 – Основні загрози локальних мереж без шифрування

Поширеною помилкою залишається думка, що домашня мережа автоматично є безпечною. Насправді джерелом загроз можуть бути гостьові клієнти Wi-Fi, заражені ноутбуки, смартфони зі шкідливими застосунками, сторонні пристрої, які отримали доступ до пароля мережі, або компрометоване обладнання інтернету речей. Саме тому навіть панель адміністрування маршрутизатора доцільно відкривати лише через захищене HTTPS-з'єднання. Використання реверс-проксі дозволяє замість прямого звернення до IP-адреси застосовувати доменне ім'я та TLS-захист, унаслідок чого пароль не передається відкрито, браузер перевіряє сертифікат, а сесія захищається криптографічними механізмами.

Атаки типу Man-in-the-Middle у внутрішніх сегментах. Однією з найнебезпечніших атак у внутрішніх сегментах є Man-in-the-Middle, за якої зловмисник стає посередником між клієнтом і сервером, перехоплюючи або змінюючи дані під час передачі. У локальних мережах така атака є особливо реальною, оскільки пристрої часто довіряють базовим мережевим механізмам без додаткової перевірки. До таких механізмів належать [14]:

— ARP;

— DHCP;

— DNS;

— широкомовні пакети.

Всі це історично створювалося для зручності, а не для захисту.

Одним із найпоширеніших сценаріїв є ARP spoofing, коли атакуючий вузол надсилає неправдиві ARP-відповіді та переконує клієнта, що шлюз мережі має MAC-адресу зловмисника. Після цього трафік починає проходити через атакуючий пристрій. Іншим варіантом є Rogue DHCP, коли запускається фальшивий DHCP-сервер, що видає клієнтам власний gateway або DNS-сервер. Унаслідок цього весь мережевий трафік спрямовується через атакуючу сторону. Також поширеним є DNS spoofing, коли користувач вводить знайому адресу, але отримує підроблену IP-адресу та переходить на фальшивий ресурс. У безпровідних мережах часто використовується Evil Twin — створення підробленої точки доступу з такою ж назвою мережі, як у справжнього Wi-Fi.

Наслідками подібних атак стають:

1. перехоплення логінів і паролів;
2. читання трафіку;
3. зміна веб-сторінок;
4. перенаправлення на фішингові сайти;
5. зараження браузера скриптами;
6. компрометація облікових записів;
7. подальше поширення атаки всередині мережі.

Використання HTTPS суттєво знижує ефективність таких дій, оскільки навіть у разі перехоплення трафіку його вміст залишається зашифрованим. Якщо атакуючий вузол намагається підмінити сервер, браузер виявляє відсутність валідного сертифіката та показує попередження про недовіру.

У системі на базі reverse proxy усі звернення проходять через одну контрольовану точку доступу. Це спрощує:

1. централізоване керування сертифікатами;
2. журналювання подій;
3. фільтрацію доступу;
4. додавання автентифікації;

5. контроль маршрутизації.

Додатково можуть використовуватися:

1. сучасні режими захисту Wi-Fi;
2. сегментація VLAN;
3. контроль DHCP;
4. регулярне оновлення прошивки маршрутизатора;
5. DNS-over-HTTPS або DNSSEC (там, де це технічно можливо).

Концепція нульової довіри (Zero Trust) на базовому рівні. Сучасною відповіддю на проблему надмірної довіри до внутрішнього сегмента стала концепція Zero Trust. Її основна ідея полягає в тому, що жоден користувач, пристрій або мережевий вузол не повинен отримувати автоматичну довіру лише тому, що знаходиться всередині локальної мережі. Безпека більше не базується на периметрі, а переноситься на рівень кожного запиту, кожного користувача та кожного сервісу.

Базові принципи Zero Trust включають:

— перевірку кожного доступу;
— мінімальні привілеї;
— безперервний контроль;
— припущення про можливу компрометацію середовища.

Кожен запит має проходити автентифікацію, а користувач отримує лише той рівень доступу, який потрібний для виконання конкретного завдання. Довіра не надається назавжди, а постійно переоцінюється залежно від контексту підключення. Інфраструктура будується так, ніби частина мережі вже потенційно скомпрометована (див. рис. 1.10) [15].

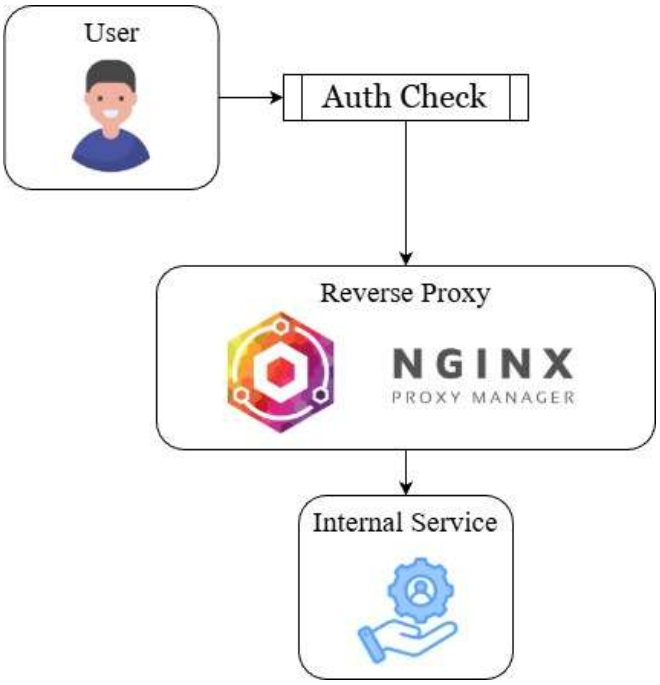


Рисунок 1.10 – Принципи Zero Trust

Навіть у домашній або лабораторній мережі такі принципи можуть бути реалізовані на базовому рівні. Першим кроком є повна відмова від HTTP на користь HTTPS. Далі всі сервіси доцільно публікувати лише через централізований reverse proxy, використовуючи зрозумілі доменні імена замість прямого доступу до внутрішніх IP-адрес. Додатково можуть застосовуватися Basic Auth, зовнішні системи автентифікації, обмеження доступу за IP або часовими правилами. Внутрішні адреси серверів залишаються прихованими, а користувач взаємодіє лише з контрольованою точкою входу.

Замість прямого звернення до адреси маршрутизатора через локальний IP формується доступ через доменне ім’я із перевіркою сертифіката та можливістю додаткового захисту. Рішення на базі Nginx Proxy Manager, DuckDNS і Let's Encrypt уже частково реалізує логіку Zero Trust, оскільки усуває потребу довіряти локальному сегменту за замовчуванням, забезпечує шифрування трафіку, приховує внутрішні сервіси та централізує контроль доступу. Надалі система може бути розширена за рахунок багатофакторної автентифікації, LDAP, SSO, журналювання входів, географічних обмежень, VPN або додаткових механізмів доступу поверх проксі-рівня.

2. ПОБУДОВА СИСТЕМИ ДОСТУПУ НА БАЗІ DOCKER ТА NGINX PROXY MANAGER

2.1 Архітектурне проєктування захищеного доступу

Для реалізації будь-якої системи доступу до внутрішніх ресурсів локальної мережі зазвичай необхідний коректний підбір програмного забезпечення та продумана архітектура взаємодії компонентів, оскільки саме від цього залежить стабільність роботи, зручність адміністрування, можливість подальшого масштабування й загальний рівень безпеки. У локальних мережах типовою проблемою часто виступає використання прямих IP-адрес, стандартних портів та відсутність централізованої точки входу до сервісів, через що доступ до ресурсів поступово ускладнюється зі збільшенням кількості пристроїв або веб-застосунків. Якщо внутрішня мережа містить маршрутизатор, файлове сховище, панель моніторингу, веб-інтерфейси керування або додаткові службові ресурси, окремий доступ до кожного вузла через IP-адресу та порт призводить до надмірної фрагментації структури адміністрування, а також створює додаткове навантаження під час супроводу такої інфраструктури.

Для усунення зазначених недоліків доцільно використовувати архітектуру, у якій центральним компонентом виступає Nginx Proxy Manager, розгорнутий у контейнерному середовищі Docker. Такий підхід формує єдину точку доступу до внутрішніх веб-сервісів, забезпечує використання доменних імен замість числових IP-адрес, надає централізоване керування HTTPS-з’єднаннями та спрощує подальше додавання нових сервісів. Додатковою перевагою є логічне розмежування компонентів системи, де мережевий рівень, прикладний рівень і рівень збереження конфігурацій функціонують окремо, але взаємопов’язано. Запропонована модель орієнтована насамперед на локальне використання в домашніх лабораторіях, навчальних середовищах, невеликих офісах або тестових

інфраструктурах, де необхідно забезпечити впорядкований доступ до внутрішніх ресурсів без використання дорогої серверної платформи.

Структурна схема локальної мережі та проксі-сервера. Якщо розглядати фізичну структуру мережі, побудова здійснюється за класичним принципом, де центральним вузлом виступає маршрутизатор, до якого підключаються всі клієнтські пристрої та серверний вузол. У межах реалізації використовується маршрутизатор Asus RT-N12. Незважаючи на застарілість моделі, пристрій залишається придатним для локальної мережі з невеликим навантаженням, оскільки його основними функціями є:

1. маршрутизація трафіку;
2. видача IP-адрес через DHCP;
3. організація доступу до локального сегмента;
4. забезпечення підключення до мережі Інтернет.

Додатково сам маршрутизатор розглядається як окремий внутрішній ресурс, тому що має власний веб-інтерфейс адміністрування, доступний за адресою 192.168.1.1.

Другим ключовим елементом архітектури виступає ноутбук або стаціонарний комп’ютер, який виконує роль локального сервера. На цьому пристрої встановлюється Docker Desktop, у межах якого запускаються необхідні контейнери системи доступу. Поточна IP-адреса серверного вузла в локальній мережі закріплюється як 192.168.1.2, що дозволяє використовувати її як постійну адресу для проксі-сервера та пов’язаних служб.

На серверному вузлі функціонує контейнеризований стек із двох основних сервісів. Першим є контейнер "nginx-proxy-manager-app", який відповідає за веб-інтерфейс керування, логіку маршрутизації HTTP/HTTPS-запитів, обробку проксі-хостів, взаємодію з SSL-сертифікатами та адміністративну панель системи. Другим виступає контейнер "nginx-proxy-manager-db", що містить базу даних MariaDB. Саме в ній зберігаються конфігурації проксі-хостів, параметри користувачів, інформація про сертифікати, правила доступу та інші службові записи.

Обидва контейнери підключаються до окремої внутрішньої Docker-мережі типу bridge з назвою "reverse-proxy", завдяки чому між ними забезпечується

ізольована службова взаємодія без необхідності відкривати внутрішні порти у зовнішню мережу. Така схема дозволяє чітко відокремити мережевий рівень від прикладного: маршрутизатор забезпечує фізичну зв'язність, Docker-хост відповідає за логіку доступу, а контейнери реалізують функціональні ролі системи (див. рис. 2.1).

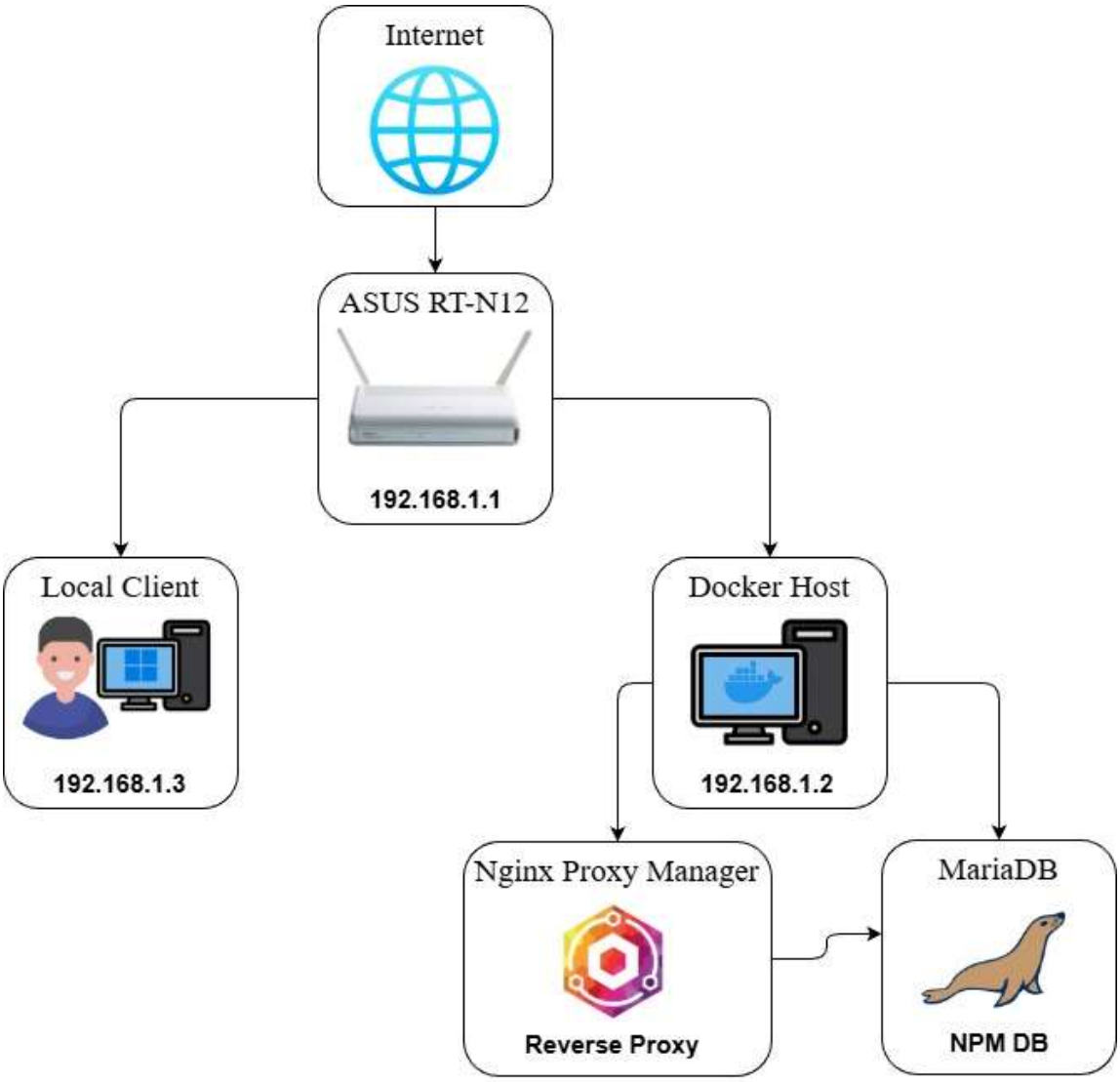


Рисунок 2.1 – Архітектура системи доступу на базі Docker та NPM

Вибір програмних компонентів (NPM, DuckDNS). Для реалізації даної системи були обрані програмні компоненти, що забезпечують:

1. мінімальну складність розгортання;

2. безкоштовне використання;
3. достатній функціонал для локальної мережевої інфраструктури.

Основним елементом виступає Nginx Proxy Manager, який часто позначається як NPM. За своєю суттю це спрощена надбудова над класичним Nginx Reverse Proxy, орієнтована на адміністрування через графічний веб-інтерфейс замість ручного редагування конфігураційних файлів. Такий підхід значно зменшує складність впровадження, оскільки адміністратор отримує готову панель керування, у межах якої можна створювати "proxy host", призначати доменні імена, перенаправляти трафік на локальні IP-адреси, активувати HTTPS-з'єднання, отримувати сертифікати Let's Encrypt, задавати правила доступу та переглядати журнали роботи сервісу [16].

Основною перевагою Nginx Proxy Manager у межах цієї реалізації є можливість швидко сформувати централізовану точку доступу без необхідності ручного написання конфігурацій для самого Nginx-сервера, оскільки традиційний підхід потребує окремого створення "server block", прописування SSL-параметрів, налаштування маршрутизації та тестування синтаксису конфігурацій. У випадку використання NPM зазначені операції виконуються через веб-інтерфейс, а зміни застосовуються автоматично. Завдяки цьому система стає доступною навіть у середовищах, де потрібне швидке впровадження без значних часових витрат [17].

Другим важливим компонентом виступає DuckDNS. Це безкоштовний Dynamic DNS сервіс, який дозволяє створити доменне ім'я третього рівня у зоні "duckdns.org". Користувач отримує можливість зареєструвати власний піддомен, наприклад "mylab.duckdns.org", або іншу вільну назву. У межах даної архітектури використовується домен формату "malikovdimaft.duckdns.org", що надалі слугує базовою адресою для внутрішніх сервісів.

Функціональне призначення DuckDNS у даному випадку не обмежується лише публікацією сервісів у глобальній мережі, оскільки сервіс використовується насамперед як зручне доменне іменування внутрішніх ресурсів локальної інфраструктури. На основі базового домену можуть створюватися логічні адреси для окремих служб, наприклад "npm.malikovdimaft.duckdns.org" для веб-інтерфейсу

Nginx Proxy Manager або "asus-router.malikovdimaft.duckdns.org" для панелі керування маршрутизатором. Подібна схема значно зручніша за використання числових адрес на кшталт 192.168.1.2 або 192.168.1.1, оскільки логічні імена простіше запам’ятовуються, краще масштабуються та створюють впорядковану структуру доступу (див. табл. 2.1).

Таблиця 2.1 – Програмні компоненти розробленої системи та їх призначення

Компонент	Призначення
Docker Desktop	Середовище контейнеризації для розгортання сервісів
Nginx Proxy Manager	Reverse Proxy з веб-інтерфейсом для маршрутизації та SSL
MariaDB	База даних для зберігання конфігурацій NPM
DuckDNS	Dynamic DNS для створення доменних імен
Let’s Encrypt	Генерація безкоштовних SSL/TLS сертифікатів
Router (Asus RT-N12)	Цільовий внутрішній ресурс (веб-інтерфейс)
Browser Client (Google Chrome)	Клієнт для доступу до сервісів через HTTPS

Обрані компоненти гармонійно поєднуються між собою, коректно працюють у Docker-середовищі та не потребують фінансових витрат на ліцензування чи придбання додаткових сервісів. Логіка їх взаємодії будується послідовно: спочатку виникає потреба централізованого доступу до ресурсів, після чого використовується reverse proxy як технічне рішення, далі з’являється необхідність у доменних іменах, яку закриває DuckDNS, надаючи безкоштовний DNS-механізм для подальшої інтеграції з HTTPS та автоматизованими сертифікатами.

Схема взаємодії локального клієнта з внутрішніми цілями. Після побудови інфраструктури користувач більше не звертається напряму до IP-адрес внутрішніх пристроїв, оскільки вся взаємодія відбувається через reverse proxy. Замість введення адреси "http://192.168.1.1" використовується доменний запис формату "https://asus-router.malikovdimaft.duckdns.org", що робить доступ зрозумілішим, зручнішим та безпечнішим.

Сам процес взаємодії починається з відкриття користувачем доменного імені у браузері. Після цього DNS-служба повертає адресу серверного вузла 192.168.1.2, на якому функціонує Nginx Proxy Manager. Подальший HTTP або HTTPS-запит

надходить саме до проксі-сервера, де виконується аналіз доменного імені, зазначеного в заголовку Host. Якщо воно відповідає створеному правилу "asus-router.malikovdimaft.duckdns.org", система автоматично перенаправляє трафік на внутрішню адресу маршрутизатора 192.168.1.1. У підсумку користувач отримує веб-інтерфейс маршрутизатора, хоча прямий доступ до його IP-адреси фактично не використовується (див. рис. 2.2).

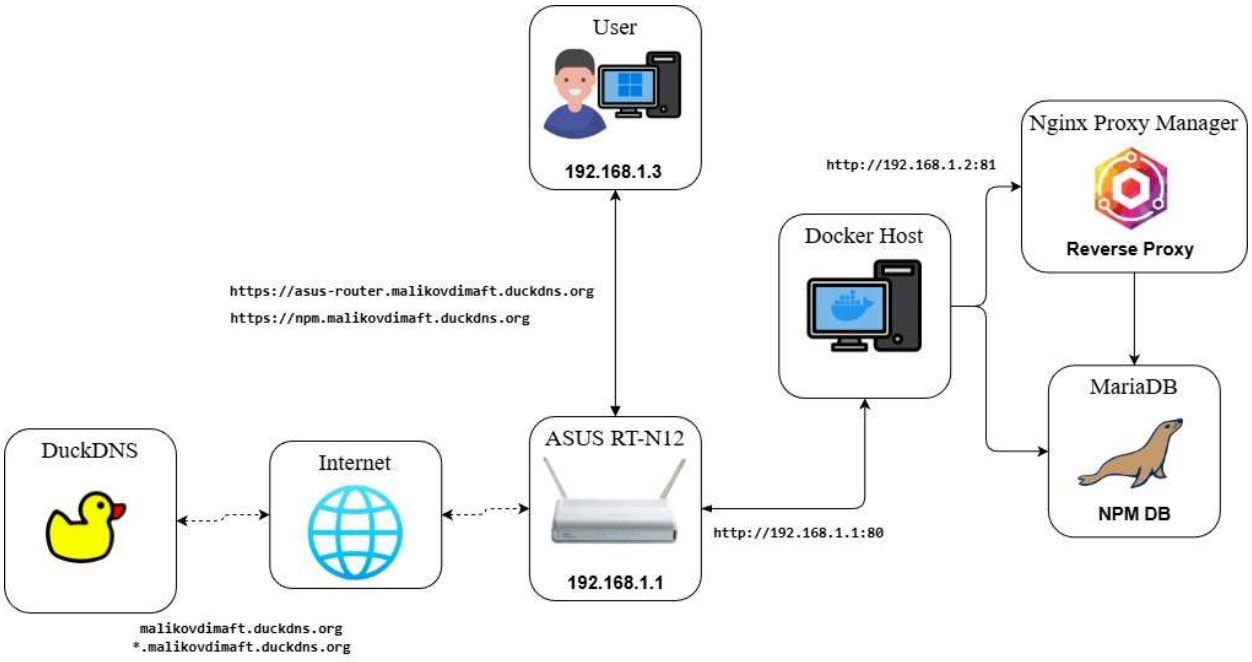


Рисунок 2.2 – Взаємодія клієнта з внутрішніми сервісами через доменне ім’я

Подібна модель дозволяє одному reverse proxy централізовано обслуговувати значну кількість внутрішніх ресурсів. Для кожного нового сервісу достатньо створити окремий "proxy host" та вказати локальну адресу призначення. Завдяки цьому зникає потреба пам’ятати численні IP-адреси, нестандартні порти або окремі шляхи доступу до різних систем.

Окремою перевагою виступає підвищення рівня безпеки, оскільки проксі-сервер стає єдиною точкою входу, через яку можуть реалізовуватися механізми авторизації, журналювання подій, фільтрації IP-адрес, обмеження доступу та контроль мережевої активності. Внутрішні вузли при цьому залишаються

прихованими від прямого звернення, а доступ до них здійснюється лише через централізований керований шлюз.

Ще однією суттєвою перевагою є масштабованість, тому що додавання нового сервісу означає лише створення нового "proxy host" у панелі Nginx Proxy Manager без необхідності перебудови всієї мережевої архітектури. Такий підхід особливо зручний у тестових середовищах, домашніх лабораторіях або невеликих офісах, де кількість служб поступово зростає.

2.2 Покрокова реалізація інфраструктури

Під час практичної реалізації системи захищеного доступу до внутрішніх ресурсів необхідно передбачати [29-30]:

1. послідовне розгортання контейнерного середовища;
2. налаштування доменного імені для локального вузла;
3. запуск Reverse Proxy сервера;
4. генерацію сертифікатів для подальшого використання HTTPS.

Такий підхід дозволяє сформувати завершену інфраструктуру, у якій кожен компонент виконує окрему функцію, але працює у межах єдиної логічної моделі. Хоча тестове розгортання здійснюється на ноутбуці з обмеженими апаратними ресурсами, після одноразового створення структури каталогів і конфігураційних файлів система легко переноситься на будь-який інший пристрій, здатний підтримувати Docker. До таких платформ можуть належати персональний комп’ютер, домашній сервер, міні-ПК, одноплатний комп’ютер або мережевий шлюз із достатнім рівнем продуктивності для роботи в режимі 24/7, що особливо важливо у випадках безперервного надання доступу до локальних сервісів.

Підготовка середовища Docker Desktop та написання docker-compose.yml. У тестовому середовищі реалізація виконувалась на ноутбуці, однак запропонований підхід не залежить від типу обладнання, оскільки за умови підтримки Docker

аналогічна система може бути розгорнута практично на будь-якій апаратній платформі. Універсальність контейнерного підходу забезпечує однакову модель запуску незалежно від того, чи використовується стаціонарний ПК, домашній сервер, сумісний одноплатний пристрій або інший вузол локальної мережі.

Початковим етапом виступає встановлення Docker Desktop як платформи, що забезпечує запуск контейнерів, керування образами, мережами та томами через графічний інтерфейс або командний рядок. Саме Docker Desktop був обраний для тестової реалізації через простоту локального адміністрування, зручність роботи з контейнерами та підтримку Docker Compose (див. рис. 2.3).

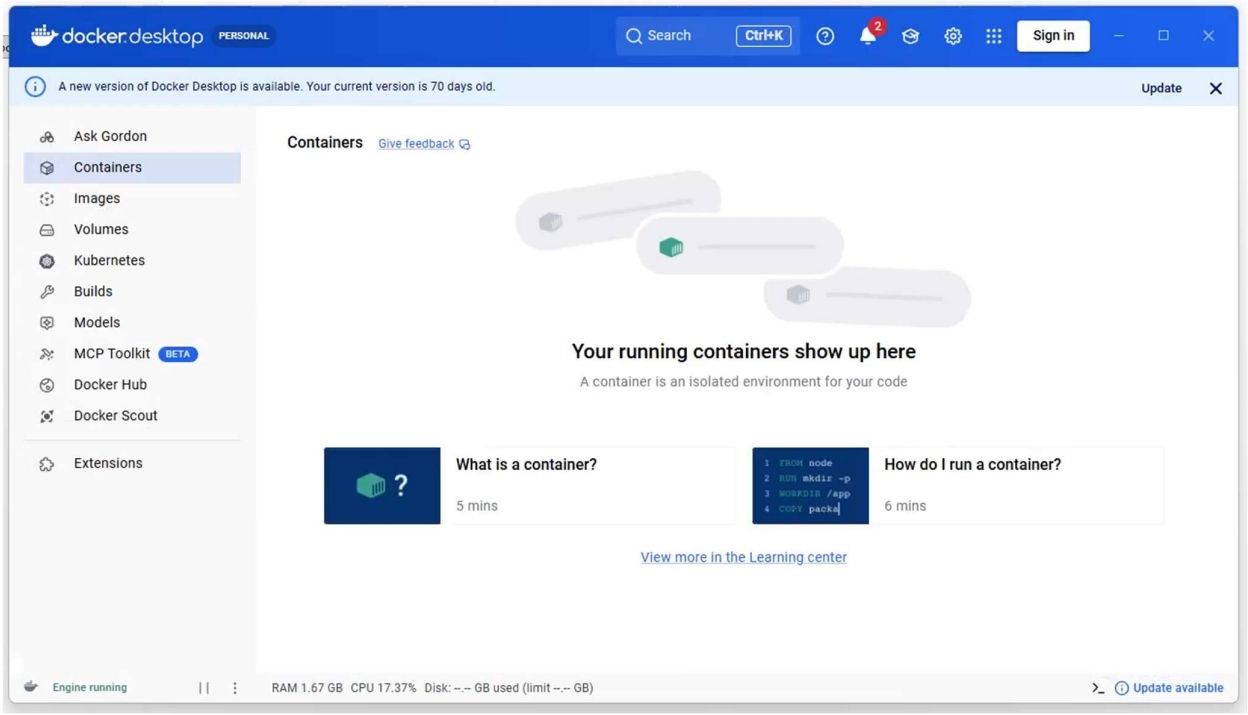


Рисунок 2.3 – Інтерфейс Docker Desktop перед розгортанням

Після встановлення формується окрема робоча директорія проєкту, у якій зберігатимуться всі конфігураційні файли, службові каталоги та змінні середовища (див. рис. 2.4) [13].

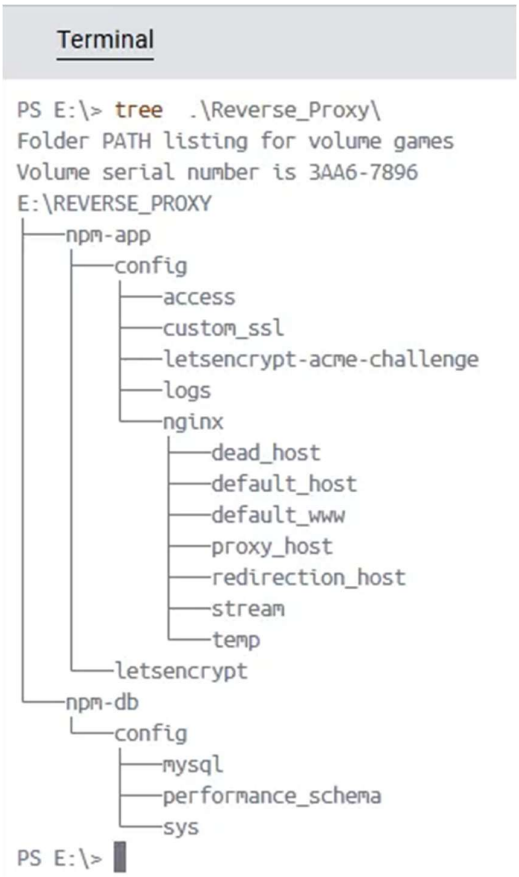


Рисунок 2.4 – Структура в межах тестового середовища

Подібна структура дозволяє фізично відокремити дані застосунку від бази даних, а також забезпечує збереження конфігурації після перезапуску контейнерів, повторного встановлення Docker або навіть повного видалення самих контейнерів, оскільки інформація зберігається безпосередньо на хості. Додатковою перевагою є можливість перенесення всієї директорії на інший пристрій із подальшим запуском без повторного налаштування сервісів.

Каталог "npm-app/config" використовується для зберігання конфігурації Nginx Proxy Manager, журналів роботи, шаблонів Nginx та правил "proxy host". Каталог "npm-app/letsencrypt" містить сертифікати Let’s Encrypt, приватні ключі та параметри автоматичного оновлення. Каталог "npm-db/config" використовується MariaDB для зберігання службових таблиць, записів користувачів, проксі-хостів, параметрів сертифікатів та інших налаштувань системи.

З архітектурної точки зору система складається з двох контейнерів. Перший контейнер "nginx-proxy-manager-app" є веб-застосунком Nginx Proxy Manager із графічною панеллю керування та механізмом маршрутизації трафіку. Другий контейнер "nginx-proxy-manager-db" реалізує базу даних MariaDB. Обидва сервіси працюють у межах внутрішньої мережі Docker під назвою "reverse-proxy", де взаємодіють між собою через службові імена контейнерів. Усередині мережі кожен контейнер має власну адресу, однак для зовнішнього сегмента вся система сприймається як один вузол із IP-адресою хоста, тобто 192.168.1.2.

Контейнер застосунку відкриває порти 80, 81 та 443. Порт 80 використовується для HTTP-з'єднань, порт 443 для HTTPS, а порт 81 призначений для веб-інтерфейсу адміністрування Nginx Proxy Manager. Контейнер бази даних зазвичай використовує внутрішній порт 3306, однак у даній реалізації він не публікується на хост, оскільки взаємодія з MariaDB потрібна лише самому застосунку всередині Docker-мережі.

Конфігураційний файл "docker-compose.yml" описує повну інфраструктуру системи, включаючи [21]:

— образи контейнерів;
— порти;
— томи;
— змінні середовища;
— залежності запуску;
— політики автоматичного відновлення після збою.

Особливе значення має параметр "depends_on" для контейнера застосунку, де вказується залежність від "nginx-proxy-manager-db", оскільки запуск веб-панелі можливий лише після старту бази даних. За відсутності доступу до MariaDB застосунок не зможе ініціалізувати конфігурацію та завершить роботу з помилкою (див. рис. 2.5).

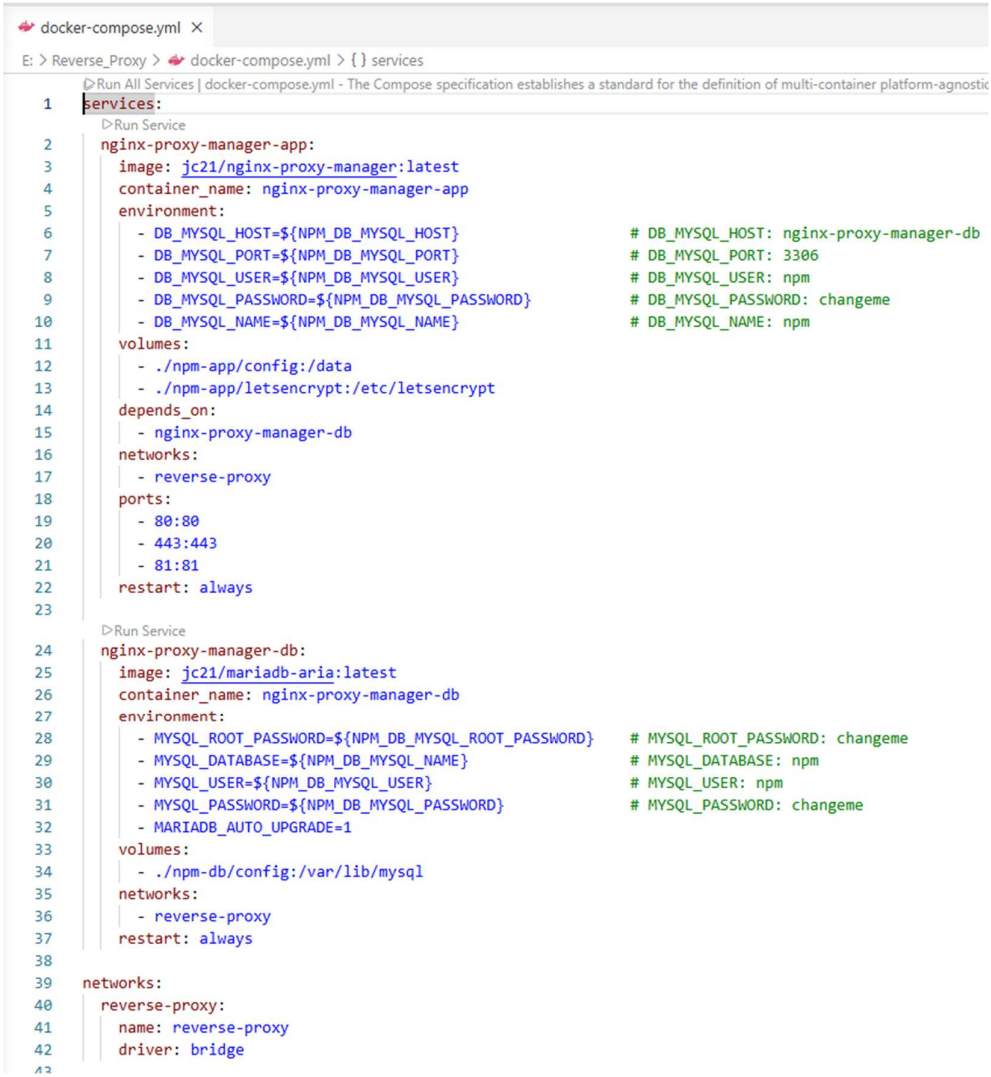


Рисунок 2.5 – Фрагмент файлу docker-compose.yml

Для підвищення безпеки використовується окремий файл ".env", у якому зберігаються службові змінні середовища (див. рис. 2.6).

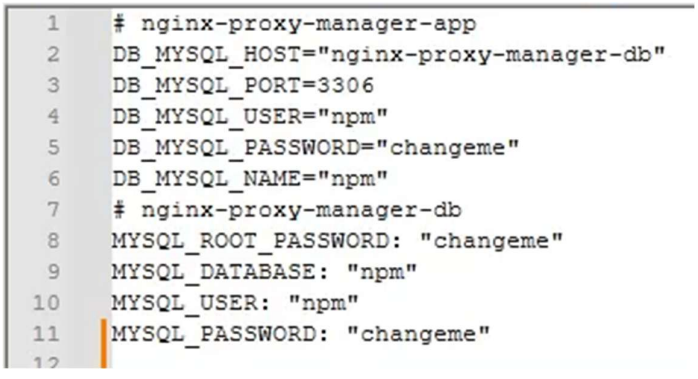


Рисунок 2.6 – Фрагмент секретного файлу.env

Такий підхід дозволяє винести конфіденційні дані за межі основного конфігураційного файлу, не залишаючи паролі у відкритому вигляді в "docker-compose.yml". Окреме зберігання секретів також спрощує міграцію системи, оскільки для перенесення достатньо скопіювати структуру каталогів, конфігураційний файл та .env, після чого виконати повторний запуск контейнерів на новому обладнанні. У результаті формується універсальна, відтворювана та зручна для супроводу інфраструктура, придатна як для тестового використання, так і для подальшого розгортання на постійному серверному вузлі.

Реєстрація та налаштування домену через DuckDNS API. Для організації доступу до локальних сервісів за доменним ім’ям замість прямого використання внутрішніх IP-адрес доцільним рішенням стало застосування сервісу DuckDNS, який надає можливість безкоштовного створення субдоменів третього рівня у форматі "назва.duckdns.org". Такий підхід значно спрощує взаємодію з внутрішніми ресурсами, оскільки замість запам’ятовування технічних адрес на зразок 192.168.1.2 або 192.168.1.1 використовується логічно зрозуміле ім’я, наприклад malikovdimaft.duckdns.org. У межах локальної інфраструктури подібне рішення є особливо корисним, адже дозволяє сформувати єдину точку доступу до сервісів та надалі використовувати її як базовий домен для маршрутизації через reverse proxy (див. рис. 2.7) [19].



Рисунок 2.7 – Реєстрація домену у DuckDNS

Перевага DuckDNS полягає не лише у безкоштовному характері використання, а й у підтримці API для оновлення IP-адрес, сумісності з механізмами автоматичного випуску SSL-сертифікатів через Let’s Encrypt, а також простоті налаштування навіть у тестових або домашніх середовищах. Завдяки цьому сервіс став доцільною альтернативою комерційним DNS-рішенням, які часто мають обмеження або вимагають додаткових витрат.

Процес налаштування починається з авторизації через обліковий запис Google, після чого відкривається особистий кабінет сервісу. У ньому автоматично формується унікальний API-токен, який надалі використовується для інтеграції із Nginx Proxy Manager під час генерації сертифікатів методом DNS Challenge. Після входу створюється бажаний субдомен, наприклад "malikovdimaft", а в полі прив’язки адреси вказується IP локального вузла, на якому розгорнуто проксі-сервер. У тестовому середовищі таким вузлом виступає Docker-host із адресою 192.168.1.2 (див. рис. 2.8) [19].

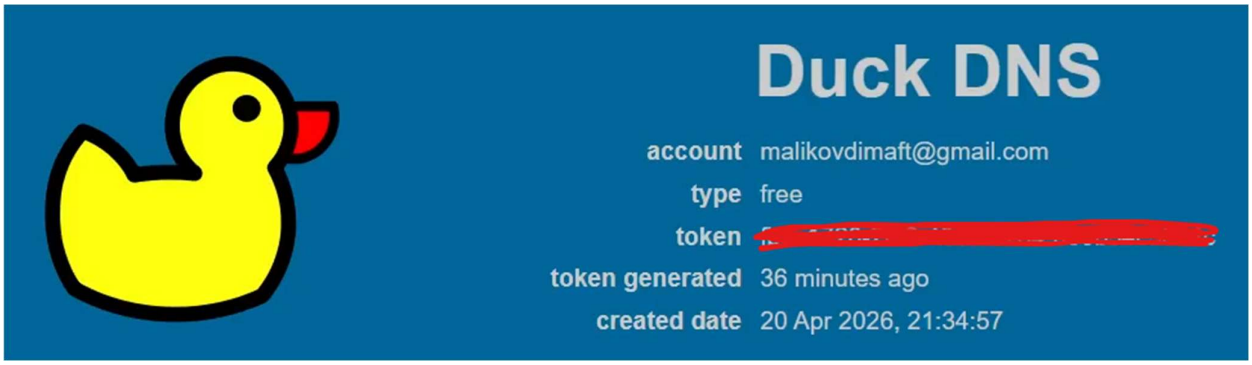


Рисунок 2.8 – Отримання токена DuckDNS API

Після збереження параметрів DNS-запис починає поширюватися мережею, а через декілька хвилин доменне ім’я стає доступним для використання. Перевірка коректності налаштування може здійснюватися стандартною командою ping або nslookup. Якщо конфігурація виконана правильно, у відповідь повертається локальна адреса 192.168.1.2, що свідчить про успішну прив’язку доменного імені до проксі-вузла (див. рис. 2.9).



Рисунок 2.9 – Перевірка корректності
зареєстрованого доменного імені через ping

Логіка роботи в такому випадку виглядає наступним чином: користувач вводить у браузері домен "malikovdimaft.duckdns.org", система DNS звертається до серверів DuckDNS, після чого отримує відповідь із записом 192.168.1.2 і перенаправляє трафік на локальний вузол із Nginx Proxy Manager. Надалі саме проксі-сервер виконує подальший аналіз запиту та визначає кінцевий сервіс, до якого необхідно передати трафік.

Застосування доменного імені створює низку практичних переваг. По-перше, усувається необхідність запам’ятовувати числові адреси внутрішньої мережі. По-друге, формується основа для подальшого використання HTTPS та автоматичних сертифікатів. По-третє, створюється універсальна адресна схема, яка може масштабуватися за рахунок додавання нових субдоменів другого рівня на зразок "npm.malikovdimaft.duckdns.org", "asus-router.malikovdimaft.duckdns.org" та інших службових назв.

Розгортання Nginx Proxy Manager та генерація SSL сертифікатів. Після завершення підготовки середовища контейнеризації та створення доменного запису наступним етапом стає запуск усієї інфраструктури reverse proxy на базі Docker Compose. Оскільки всі параметри сервісів були попередньо описані у файлі docker-compose.yml, розгортання системи здійснюється через "docker compose up -d" з робочої директорії проєкту (див. рис. 2.10).

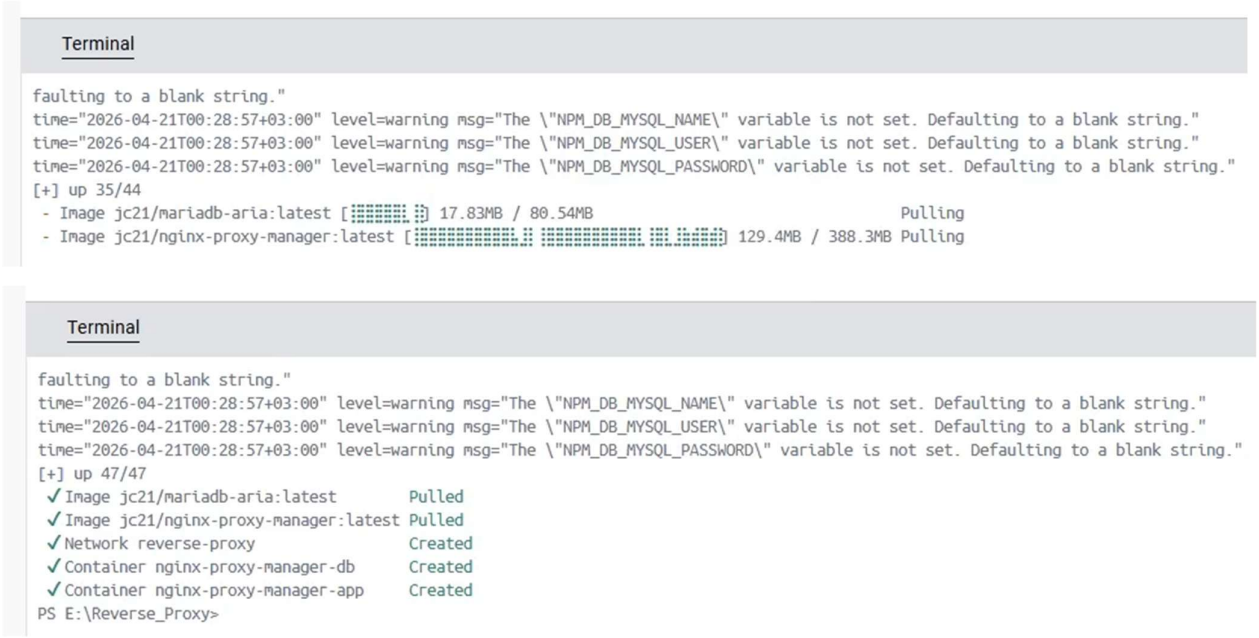


Рисунок 2.10 – Запуск контейнерів через командний рядок

Виконання цієї команди запускає автоматизований процес побудови інфраструктури. Спочатку Docker перевіряє наявність необхідних образів контейнерів та за потреби завантажує актуальні версії з репозиторіїв. Далі створюється внутрішня bridge-мережа "reverse-proxy", у межах якої працюватимуть обидва контейнери. Після цього запускається база даних MariaDB, а вже потім веб-застосунок Nginx Proxy Manager, що відповідає логіці параметра "depends_on". Така послідовність необхідна для коректного підключення застосунку до сховища конфігурації. Після завершення старту сервісів адміністративна панель стає доступною за адресою "http://192.168.1.2:81" (див. рис. 2.11).

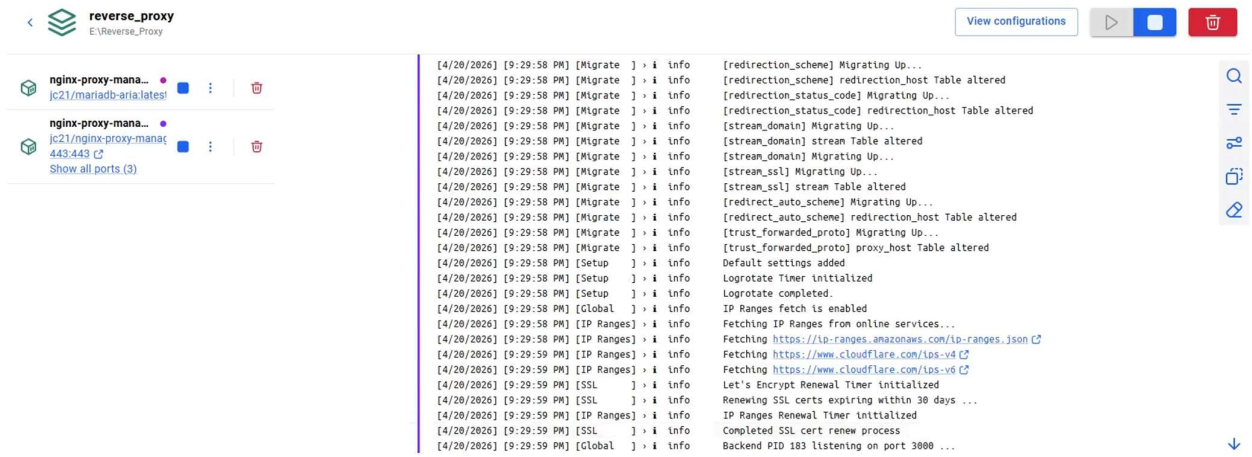


Рисунок 2.11 – Список активних контейнерів після запуску

Перший вхід супроводжується процедурою ініціалізації системи, під час якої задається ім’я адміністратора, адреса електронної пошти та новий пароль доступу. Після цього відкривається головний інтерфейс керування Nginx Proxy Manager, через який виконуються всі подальші налаштування маршрутизації, сертифікатів і доменів (див. рис. 2.12).

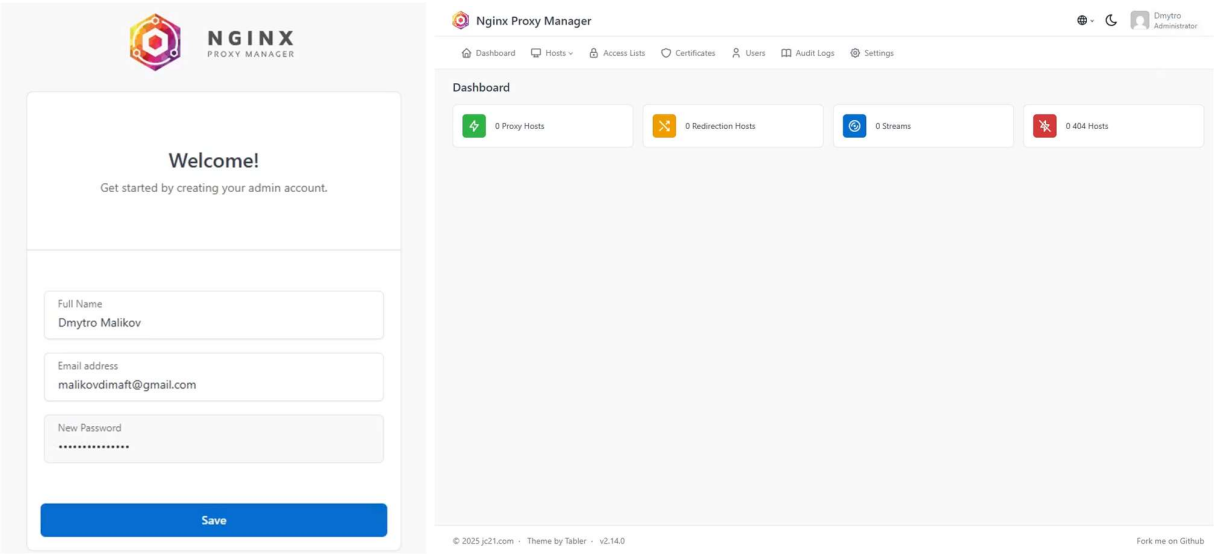


Рисунок 2.12 – Панель керування Nginx Proxy Manager

Наступним критично важливим етапом є генерація SSL-сертифіката. Для цього використовується вбудований механізм інтеграції з Let’s Encrypt через DNS Challenge. У розділі SSL Certificates обирається команда додавання нового

сертифіката, після чого задаються доменні імена "malikovdimaft.duckdns.org" та ".malikovdimaft.duckdns.org".

Перший запис відповідає основному домену, другий дозволяє обслуговувати будь-які дочірні субдомени за шаблоном wildcard. Далі як DNS-провайдер обирається DuckDNS, а в поле авторизації вставляється API-токен, отриманий у кабінеті сервісу. Для коректної перевірки встановлюється затримка поширення DNS-записів, наприклад 30 секунд [16].

Після підтвердження система автоматично виконує DNS Challenge: створює службовий TXT-запис через API DuckDNS, звертається до серверів Let’s Encrypt, проходить перевірку права володіння доменом та випускає цифровий сертифікат терміном дії 90 днів. Отриманий сертифікат зберігається у каталозі "npm-app/letsencrypt", а в подальшому автоматично оновлюється без ручного втручання (див. рис. 2.13) [5].

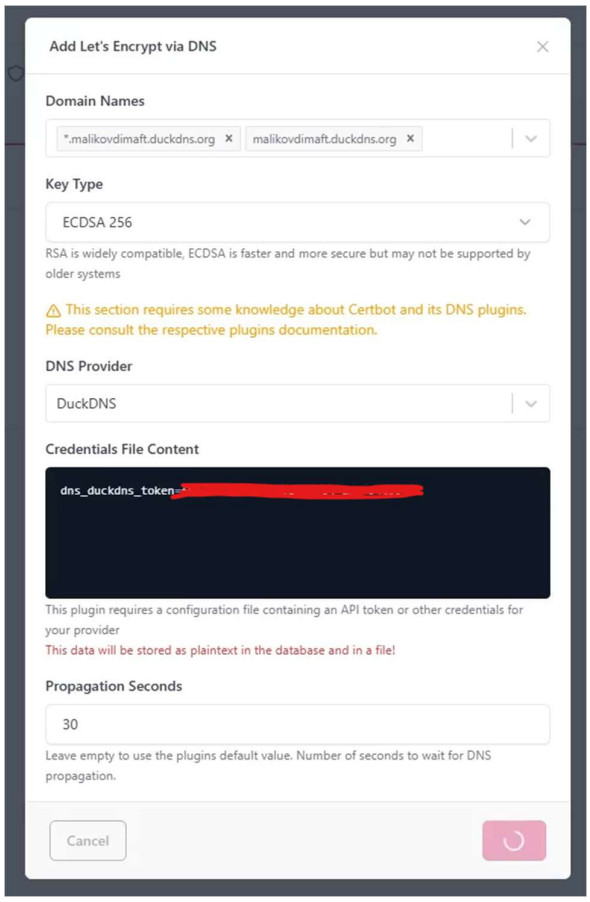


Рисунок 2.13 – Генерація SSL-сертифіката Let’s Encrypt

Наявність сертифіката ще не означає захисту сервісів, тому наступним етапом стає створення "proxy host" записів. Саме вони зв’язують доменне ім’я із конкретною внутрішньою ціллю (див. рис. 2.14).

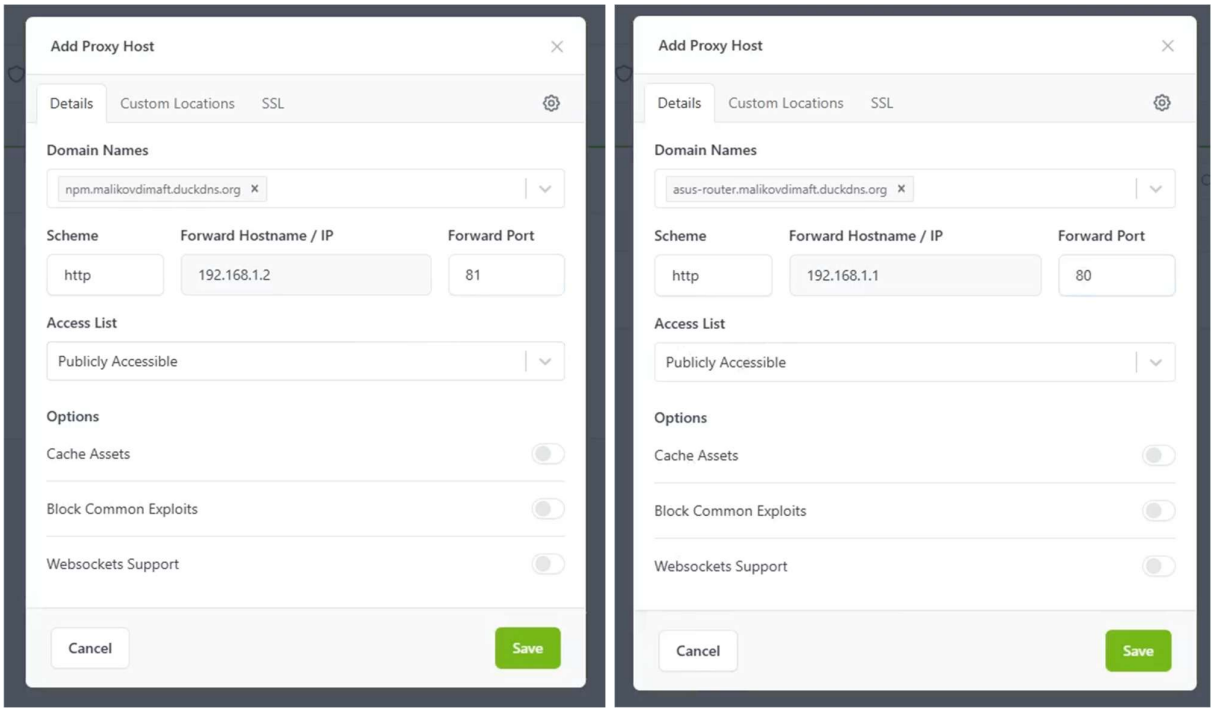


Рисунок 2.14 – Створення Proxy Host у веб-інтерфейсі

У такому випадку браузер звертається до адреси "https://npm.malikovdimaft.duckdns.org", а проксі-сервер приймає HTTPS-запит, завершує TLS-сесію та передає внутрішній HTTP-трафік на локальний порт 81. У результаті доступ до веб-панелі маршрутизатора виконується через захищений HTTPS-канал, хоча сам пристрій у локальній мережі може підтримувати лише звичайний HTTP. Саме Nginx Proxy Manager бере на себе функцію шифрування зовнішнього каналу та безпечної передачі даних (див. рис. 2.15) [16].

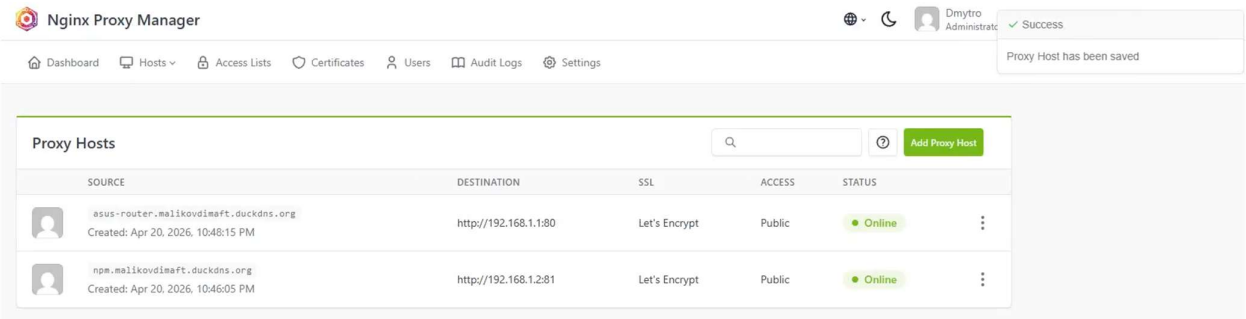


Рисунок 2.15 – Успішно доданий проксі-хост у системі

Архітектурно така схема працює наступним чином: клієнт відкриває доменне ім’я сервісу, DNS повертає адресу проксі-сервера, Nginx Proxy Manager аналізує заголовок Host, визначає потрібне правило маршрутизації та пересилає запит до відповідного внутрішнього ресурсу. У відповідь користувач отримує сторінку сервісу вже в захищеному вигляді.

Практична цінність подібного рішення полягає у відсутності потреби запам’ятовувати IP-адреси та порти, централізованому керуванні доступом до всіх сервісів, автоматичному випуску й оновленні сертифікатів, а також можливості швидкого масштабування. Для додавання нового ресурсу достатньо створити новий Proxy Host запис без змін у мережевій топології.

Окремою перевагою є повна переносимість системи. Усі налаштування зберігаються у bind-mounted каталогах на хості, тому конфігурація може бути перенесена на інший ПК, сервер, міні-ПК або одноплатну платформу шляхом копіювання директорії проєкту та повторного запуску "docker compose up -d". Після старту система автоматично відновлює сертифікати, правила маршрутизації, користувачів та інші параметри без повторного налаштування, що підтверджує її високу практичну придатність для домашніх, навчальних і малих корпоративних мереж.

2.3 Алгоритми обробки мережевих запитів

Після завершення розгортання контейнерної інфраструктури, налаштування доменного імені через DuckDNS та запуску Nginx Proxy Manager система переходить до основного експлуатаційного режиму, у межах якого здійснюється приймання, аналіз та обробка мережевих запитів від локальних клієнтів. На цьому етапі проксі-вузол перетворюється на централізовану точку входу, через яку проходить увесь трафік до внутрішніх сервісів локальної мережі. Саме тут відбувається [30]:

— визначення цільового маршруту;

— перевірка правил доступу;

— застосування SSL-захисту;

— коригування службових параметрів HTTP-протоколу;

— передача запиту до відповідного внутрішнього ресурсу.

На відміну від класичної моделі локальної мережі, де доступ до кожного пристрою здійснюється напряму за IP-адресою та портом, запропонований підхід вводить додатковий керований рівень у вигляді reverse proxy. Така проміжна ланка забезпечує централізований контроль над потоками даних, приховує внутрішню структуру мережі, дозволяє використовувати доменні імена замість числових адрес, автоматизує керування сертифікатами та створює можливість масштабування без зміни фізичної топології. Завдяки цьому кожен внутрішній сервіс стає логічно відокремленим, хоча фізично продовжує працювати у межах тієї самої мережі.

Функціонування системи у робочому режимі базується на декількох основних алгоритмічних процесах. До них належить:

1. перевірка дійсності SSL-сертифікатів;
2. їх автоматичне оновлення;
3. маршрутизація вхідних HTTP та HTTPS-запитів;
4. відповідно до доменного імені.

5. коригування службових заголовків перед передачею трафіку до внутрішніх застосунків.

Сукупність цих механізмів дозволяє забезпечити стабільну роботу інфраструктури без постійного ручного адміністрування.

Алгоритм перевірки та оновлення SSL сертифікатів (DNS Challenge). Однією з базових функцій системи є підтримка захищеного HTTPS-з'єднання між клієнтським пристроєм та проксі-сервером. Для реалізації такого каналу використовується цифровий сертифікат Let's Encrypt, отриманий через механізм DNS Challenge. На відміну від HTTP Challenge, який потребує зовнішньої доступності веб-сервера через Інтернет, DNS Challenge працює шляхом підтвердження права володіння доменом через DNS-записи, що робить його придатним навіть для локальних середовищ без публічної IP-адреси. Така властивість особливо важлива для домашніх лабораторій, тестових мереж і невеликих офісних сегментів, де прямий зовнішній доступ відсутній або небажаний [10].

Логіка роботи алгоритму побудована на циклічному контролі строку дії вже наявних сертифікатів. Nginx Proxy Manager періодично перевіряє дату завершення чинного сертифіката, а також наявність самого файлу сертифіката у сховищі. Якщо строк дії ще достатній, система не виконує жодних додаткових дій та повертається до режиму очікування наступної перевірки. Якщо ж сертифікат відсутній або наближається до завершення терміну дії, автоматично запускається процедура перевипуску.

На першому етапі формується службовий DNS-запис типу TXT, який використовується Let's Encrypt для перевірки контролю над доменним ім'ям. Через API сервісу DuckDNS проксі-система передає токен автентифікації та створює відповідний запис у DNS-зоні. Після цього запускається етап очікування поширення DNS-інформації, що необхідно для коректного проходження валідації. Саме на цій стадії іноді виникають короткочасні затримки, через які перша спроба може завершитися помилкою, а повторний запуск зазвичай успішно завершує процедуру [20].

Після завершення поширення запису сервери Let’s Encrypt виконують перевірку наявності TXT-запису. Якщо підтвердження успішне, формується новий сертифікат та приватний ключ, які автоматично зберігаються у директорії контейнера "letsencrypt". Після оновлення файлів конфігурація Nginx Proxy Manager застосовується повторно, а веб-сервіси продовжують роботу вже з новим сертифікатом без зупинки доступу користувачів [20].

У випадку помилки перевірки система фіксує подію у журналі та через певний проміжок часу повторює спробу. Такий механізм знижує потребу ручного втручання адміністратора та забезпечує стійкість до тимчасових збоїв DNS-провайдера або затримок мережевої реплікації (див. рис. 2.16).

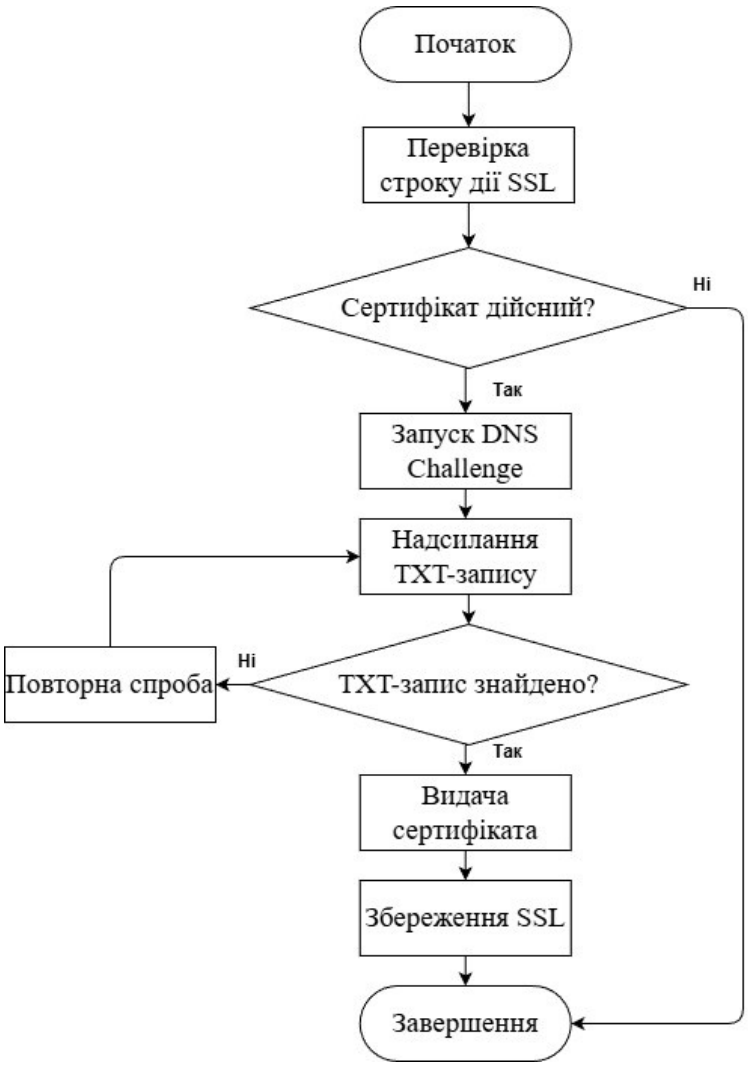


Рисунок 2.16 – Алгоритм перевірки SSL-сертифікатів

Завдяки такому алгоритму підтримка HTTPS відбувається у повністю автоматичному режимі. Відсутня необхідність ручного продовження сертифікатів кожні 90 днів, зменшується ризик втрати захищеного доступу до сервісів, а локальна інфраструктура отримує рівень криптографічного захисту, аналогічний зовнішнім публічним веб-ресурсам.

Алгоритм маршрутизації вхідного запиту до цільового сервісу. Головне призначення реверс-проксі полягає у визначенні того, до якого саме внутрішнього ресурсу необхідно передати отриманий мережевий запит. У традиційній локальній мережі звернення до сервісів зазвичай виконується безпосередньо через IP-адресу та номер порту, однак такий підхід поступово ускладнює використання інфраструктури, особливо при збільшенні кількості веб-застосунків, адміністративних панелей, файлових сховищ чи інших внутрішніх систем. Використання централізованого проксі-рівня дозволяє замінити технічні адреси зрозумілими доменними назвами та винести всю логіку маршрутизації в окремий керований вузол.

У побудованій системі вибір цільового сервісу здійснюється на основі доменного імені, яке вказується користувачем у браузері. Наприклад, адреса "npm.malikovdimaft.duckdns.org" може використовуватися для доступу до панелі керування Nginx Proxy Manager, а адреса "asus-router.malikovdimaft.duckdns.org" — для переходу до веб-інтерфейсу маршрутизатора. Для користувача така схема виглядає як робота зі звичайними веб-сайтами, хоча фактично вся обробка виконується всередині локальної мережі.

Після введення адреси у браузері система DNS повертає IP-адресу сервера, на якому розміщено контейнеризований реверс-проксі. Далі клієнт формує HTTP або HTTPS-запит та передає його на відповідний порт сервера. Якщо використовується захищене з’єднання, запит надходить на порт 443, після чого проксі завершує TLS-сеанс, зчитує службові параметри та аналізує заголовок Host. Саме значення заголовка Host містить доменне ім’я, за яким визначається необхідне правило маршрутизації [21].

У конфігурації Nginx Proxy Manager кожному домену відповідає окремий Proxy Host. У цьому записі зберігається інформація про цільову IP-адресу внутрішнього ресурсу, порт застосунку, тип протоколу, параметри SSL та додаткові правила доступу. Після отримання запиту система порівнює значення Host із наявними правилами. Якщо знайдено відповідність, запит автоматично перенаправляється до потрібного сервісу у внутрішньому сегменті мережі.

Наприклад, запит до адреси "https://asus-router.malikovdimaft.duckdns.org" після аналізу конфігурації може бути перетворений у внутрішній маршрут "http://192.168.1.1:80", де розташована панель адміністрування маршрутизатора. При цьому користувач продовжує працювати лише з доменним іменем, а фактична IP-адреса обладнання залишається прихованою від прямого доступу. Аналогічно адреса "https://npm.malikovdimaft.duckdns.org" може переспрямовуватися до "http://192.168.1.2:81", де працює веб-інтерфейс системи керування проксі.

Послідовність маршрутизації в загальному вигляді складається з кількох логічних етапів. Спочатку користувач відкриває веб-адресу у браузері. Далі DNS-служба повертає IP-адресу проксі-сервера. Після цього мережевий запит надходить на порт 80 або 443. Система зчитує значення Host та перевіряє наявність відповідного правила у конфігурації. Якщо запис знайдено, формується внутрішнє перенаправлення до цільового сервісу. Якщо збіг відсутній, повертається сторінка помилки, стандартна відповідь сервера або повідомлення про недоступність ресурсу (див. рис. 2.17) [21].

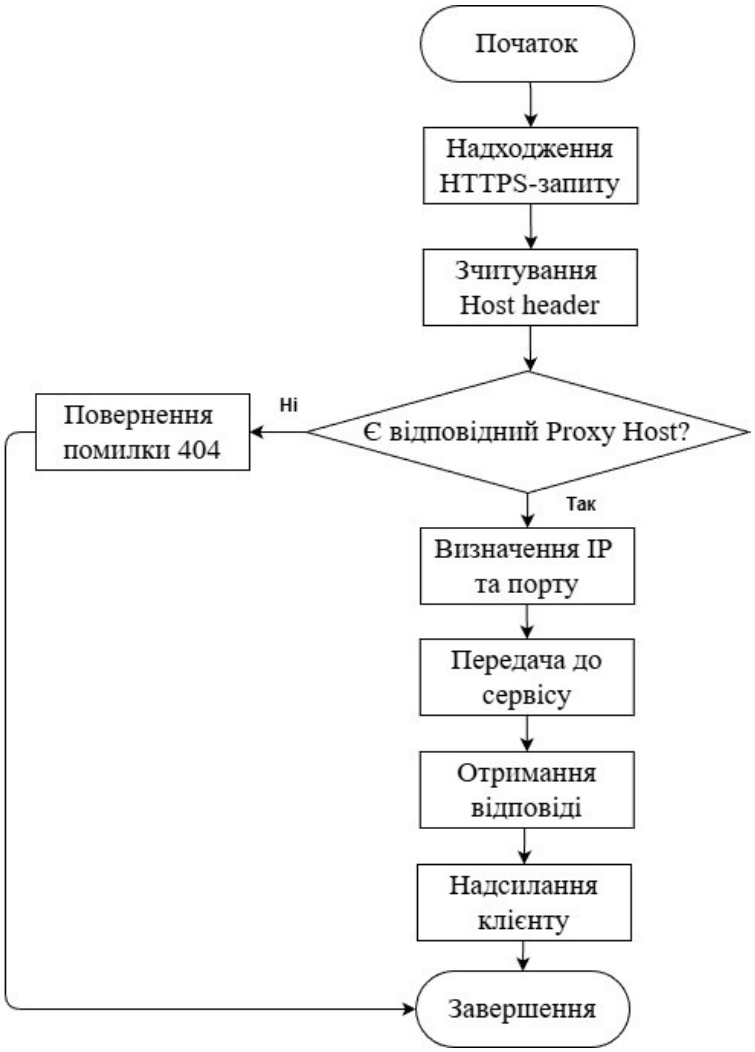


Рисунок 2.17 – Алгоритм маршрутизації вхідного запиту

Практична цінність такого механізму полягає у приховуванні внутрішньої адресації мережі. Користувачеві не потрібно запам’ятовувати IP-адреси, нестандартні порти або технічні параметри кожного окремого сервісу. Достатньо використовувати логічно зрозумілі назви, що значно спрощує адміністрування та повсякденну експлуатацію системи.

Додатковою перевагою є централізоване керування всією інфраструктурою доступу. Додавання нового ресурсу не потребує зміни конфігурації клієнтських пристроїв або ручного редагування системних файлів на кожному комп’ютері. Достатньо створити новий Proxy Host у панелі Nginx Proxy Manager, вказати

доменне ім’я та адресу цільового сервісу, після чого новий ресурс одразу стає доступним за заданою адресою.

Окремо варто відзначити безпековий аспект. Оскільки зовнішньою точкою входу залишається лише проксі-сервер, внутрішні сервіси не потребують прямого оприлюднення в мережі. Реальні адреси веб-панелей, допоміжних сервісів чи інших систем залишаються прихованими, а доступ до них контролюється через єдиний шлюз. За потреби на цьому рівні можуть додатково застосовуватися фільтрація IP-адрес, авторизація, журналювання або обмеження доступу за часом.

Алгоритм підміни заголовків для коректної роботи застосунків. Значна кількість внутрішніх веб-застосунків початково проєктується для прямого доступу за локальною IP-адресою без використання проміжного проксі-рівня, тому при перенесенні таких сервісів за реверс-проксі можуть виникати технічні ускладнення, пов’язані з неправильними перенаправленнями, втратою реальної адреси клієнта, некоректним визначенням протоколу з’єднання, циклічними редиректами між HTTP та HTTPS, а також помилками формування абсолютних посилань у веб-інтерфейсі. Причина полягає в тому, що внутрішній сервіс бачить запит уже не від браузера користувача, а від проксі-сервера, через що стандартна логіка роботи багатьох застосунків перестає відповідати реальній мережевій схемі.

Для усунення таких проблем використовується механізм підміни та доповнення службових HTTP-заголовків перед передачею запиту до внутрішнього ресурсу. Під час роботи реверс-проксі приймає клієнтський HTTPS-запит, аналізує параметри маршрутизації, після чого формує новий службовий контекст, який передається цільовому застосунку. Завдяки цьому внутрішній сервіс отримує інформацію не лише про проксі-вузол, а й про фактичні умови первинного звернення користувача.

Одним із базових заголовків є Host, який передає доменне ім’я, введене у браузері. Для застосунку така інформація є критично важливою, оскільки саме на її основі часто формуються внутрішні посилання, маршрути, політики безпеки або механізми авторизації. Якщо внутрішній сервіс замість реального домену отримує лише локальну адресу, наприклад 192.168.1.2, веб-інтерфейс може генерувати

некоректні URL-адреси або перенаправляти користувача на технічну IP-адресу замість зручного доменного імені.

Іншим важливим параметром є X-Real-IP, який передає фактичну адресу клієнтського пристрою. Без такого заголовка застосунок бачить лише IP-адресу самого проксі-сервера, а отже журнали доступу, статистика, системи контролю входу або обмеження за адресами втрачають практичну точність. Передача реальної адреси дозволяє зберігати коректні записи подій та контролювати джерела підключення [3].

Додатково використовується заголовок X-Forwarded-For, що містить ланцюг усіх проміжних проксі-вузлів, через які проходив запит. У локальній інфраструктурі така функція корисна для журналювання, діагностики та подальшого масштабування системи, якщо згодом буде впроваджено декілька мережевих рівнів маршрутизації [3].

Не менш важливим є заголовок X-Forwarded-Proto, який передає оригінальний протокол звернення користувача. Якщо браузер відкрив сторінку через HTTPS, внутрішній сервіс повинен знати саме цей факт, навіть якщо між проксі та застосунком використовується звичайний HTTP у межах локальної мережі. За відсутності такого параметра система може помилково вважати з’єднання незахищеним та почати повторно перенаправляти клієнта на HTTPS, утворюючи циклічний редирект [3].

Загальна послідовність роботи алгоритму виглядає таким чином: браузер користувача формує HTTPS-запит до доменного імені сервісу, запит надходить на реверс-проксі, система знаходить відповідний Proxy Host, після чого до запиту додаються або змінюються службові заголовки, далі сформований пакет передається внутрішньому застосунку, сервіс обробляє його з урахуванням отриманих параметрів та повертає відповідь назад через проксі до клієнта. Таким чином між зовнішнім зверненням і внутрішнім сервісом створюється узгоджений інформаційний шар, що дозволяє коректно працювати навіть тим застосункам, які не були спочатку орієнтовані на використання через проксі (див. рис. 2.18).

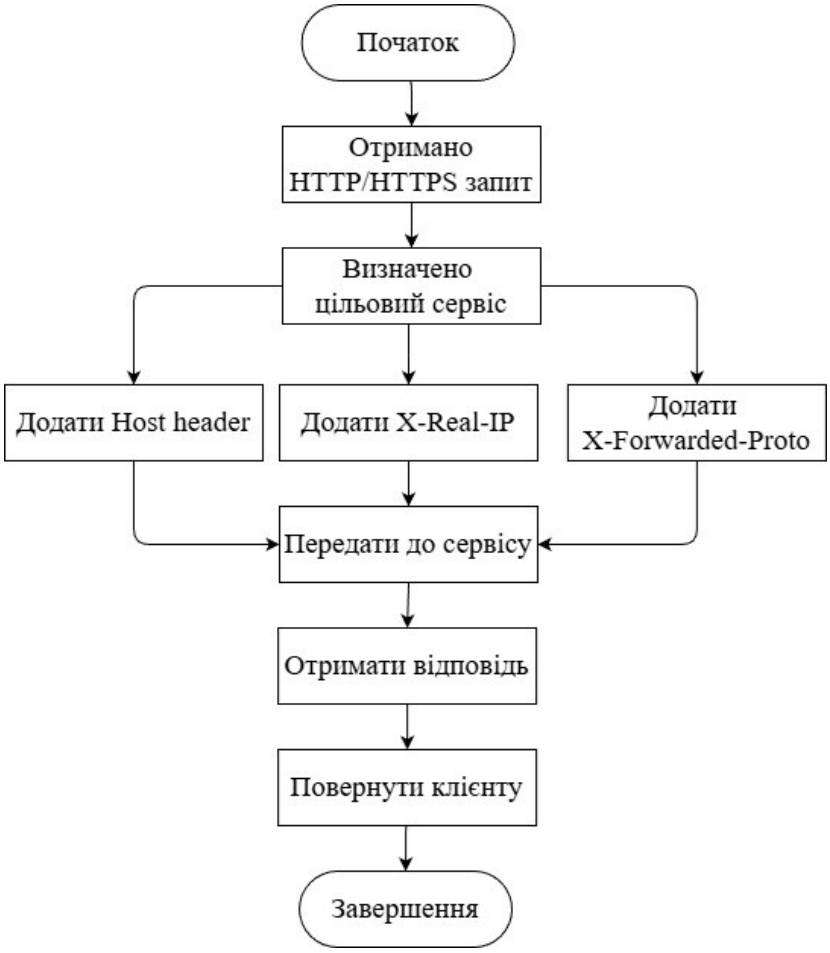


Рисунок 2.18 – Алгоритм підміни службових заголовків

Практичний приклад можна розглянути для адреси "https://npm.malikovdimaft.duckdns.org". Після проходження через проксі внутрішній сервіс може отримати заголовок Host зі значенням "npm.malikovdimaft.duckdns.org", заголовок X-Forwarded-Proto зі значенням https та X-Real-IP із локальною адресою клієнтського пристрою, наприклад 192.168.1.2. На основі цих даних панель керування правильно формує захищені посилання, зберігає коректну адресу джерела запиту в журналі та не створює конфліктів із повторними перенаправленнями.

У випадку адміністративних панелей маршрутизаторів, систем моніторингу, файлових веб-інтерфейсів або внутрішніх порталів така схема є особливо корисною, оскільки подібні сервіси часто чутливі до адреси доступу та протоколу

з’єднання. Без підміни заголовків сторінки можуть відкриватися частково, некоректно працювати після входу до облікового запису або повертати помилки авторизації через невідповідність домену.

Алгоритм підміни заголовків одночасно підвищує сумісність сервісів та зменшує необхідність ручного втручання у конфігурацію кожного окремого застосунку. Замість редагування параметрів десятків внутрішніх систем достатньо централізовано налаштувати поведінку реверс-проксі, після чого більшість веб-ресурсів починає коректно працювати через єдину точку входу.

2.4 Практичне тестування доступу до сервісів

Після завершення розгортання всієї інфраструктури наступним логічним етапом стало практичне підтвердження працездатності побудованої системи доступу до внутрішніх сервісів через механізм реверс-проксі. Основне завдання тестування полягало у перевірці того, чи локальні ресурси можуть відкриватися не напряму за IP-адресами та стандартними портами, а через доменні імена із використанням захищеного HTTPS-з’єднання. Додатково перевірялася коректність маршрутизації всередині локального сегмента мережі, стабільність DNS-резолюції, а також ступінь ізоляції внутрішніх ресурсів від зовнішніх мережевих звернень.

Практичні випробування проводилися в межах локальної мережі, де сервер із Docker Desktop та контейнерами Nginx Proxy Manager мав адресу 192.168.1.2, а маршрутизатор Asus RT-N12 використовував адресу 192.168.1.1. Для доступу застосовувався один базовий домен, створений через сервіс DuckDNS, на основі якого формувалися окремі субдомени для внутрішніх сервісів. Такий підхід дозволив централізовано використовувати одну DNS-зону без створення окремих доменів для кожного ресурсу, що спрощує керування системою та подальше масштабування.

Під час тестування оцінювалася поведінка системи у двох сценаріях:

— доступ із внутрішньої мережі;

— спроба звернення з іншого мережевого сегмента.

У межах локальної інфраструктури всі сервіси повинні були відкриватися коректно через доменні імена, тоді як зовнішні підключення не мали отримувати прямий доступ до внутрішніх ресурсів. Саме така модель підтверджує, що реверс-проксі використовується не як інструмент безконтрольної публікації локальних вузлів, а як керований шлюз із чітко визначеним середовищем роботи.

Налаштування проксіювання панелі управління маршрутизатором. Одним із перших практичних прикладів використання побудованої системи стало підключення до веб-інтерфейсу маршрутизатора через доменне ім’я. У типовому режимі доступ до більшості домашніх маршрутизаторів здійснюється за прямою локальною адресою, наприклад "http://192.168.1.1".

Такий спосіб має низку обмежень: використовується незахищений HTTP-протокол, відсутнє централізоване керування точками входу, користувач повинен пам’ятати технічну адресу пристрою, а можливості додаткового контролю доступу залишаються обмеженими.

Для усунення цих недоліків у Nginx Proxy Manager було створено окремий Proxy Host, який прив’язував субдомен до внутрішньої адреси маршрутизатора. Як цільовий вузол використовувалася адреса 192.168.1.1, порт 80, схема доступу HTTP, тоді як зовнішня взаємодія з користувачем виконувалася вже через HTTPS. Після збереження конфігурації всі звернення до відповідного доменного імені почали автоматично спрямовуватися до внутрішнього веб-інтерфейсу маршрутизатора через проксі-рівень.

Фактична послідовність роботи виглядала наступним чином: користувач вводив у браузері доменну адресу сервісу, після чого DNS-служба повертала IP-адресу проксі-сервера 192.168.1.2. Далі HTTPS-запит надходив до Nginx Proxy Manager, де за правилом маршрутизації визначався цільовий вузол, а потім перенаправлявся на "http://192.168.1.1". У результаті відкривалася панель керування маршрутизатором, хоча користувач безпосередньо не взаємодіяв із його реальною IP-адресою (див. рис. 2.19).

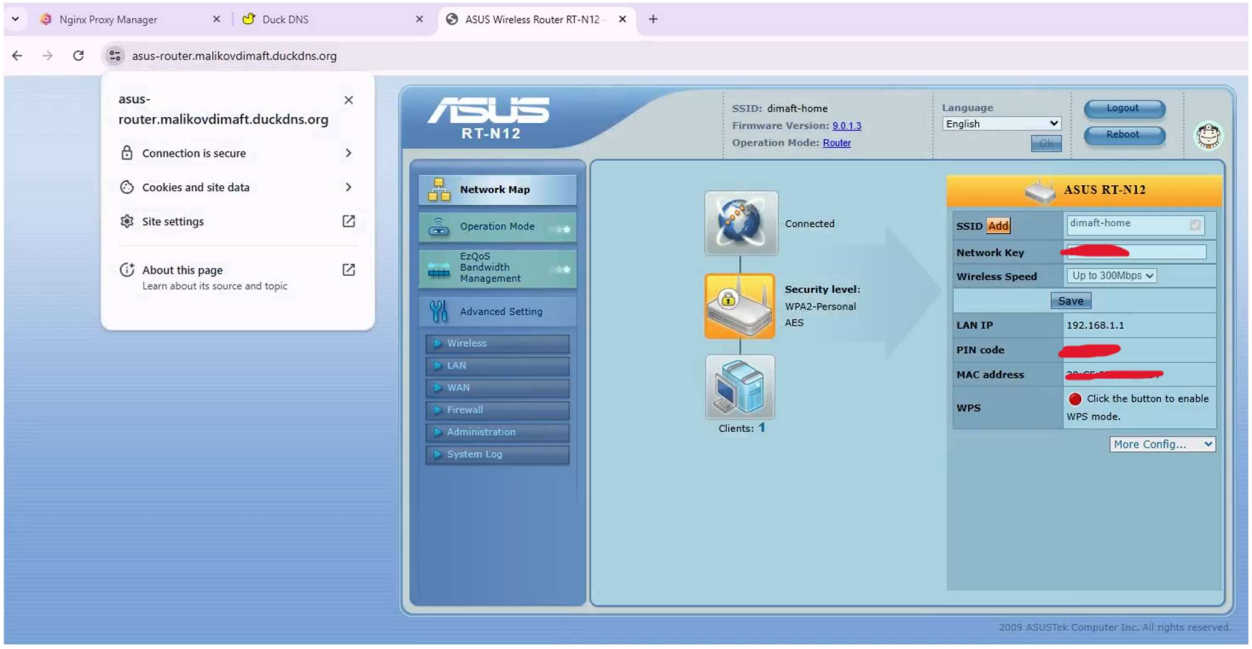


Рисунок 2.19 – Відкриття внутрішнього сервісу через HTTPS-домен

Подібна схема надає декілька суттєвих переваг. По-перше, маршрутизатор, який початково підтримує лише HTTP або застарілі механізми захисту, отримує сучасний спосіб доступу через TLS-шифрування на рівні проксі. По-друге, вся система переходить до моделі єдиної точки входу, де кожен сервіс відкривається за зрозумілим доменним ім’ям. По-третє, у подальшому до такого Proxy Host можуть додаватися додаткові механізми захисту, зокрема базова автентифікація, фільтрація IP-адрес, журналювання входів, примусове перенаправлення HTTP на HTTPS та інші правила безпеки.

Практичне тестування підтвердило, що навіть застаріле мережеве обладнання може бути інтегроване у сучасну систему керованого доступу без зміни штатної прошивки чи встановлення додаткового програмного забезпечення на сам пристрій. Усі функції безпеки та маршрутизації переносяться на проміжний проксі-рівень, що дозволяє подовжити актуальність існуючого обладнання та підвищити зручність адміністрування локальної мережі.

Окремо було перевірено стабільність повторних підключень, коректність авторизації у веб-інтерфейсі маршрутизатора та відсутність циклічних перенаправлень. Усі ці перевірки засвідчили, що проксіювання адміністративної

панелі працює штатно, а затримки відкриття сторінок залишаються мінімальними, оскільки весь трафік проходить у межах локального сегмента без звернення до віддалених серверів.

Перевірка доступу за доменним ім'ям та валідація HTTPS з'єднання. Після завершення розгортання інфраструктури наступним етапом стала практична перевірка працездатності системи доступу до внутрішніх сервісів через механізм реверс-проксі. Основною метою тестування було підтвердження того, що локальні ресурси можуть відкриватися не шляхом прямого звернення до внутрішніх IP-адрес і портів, а через доменні імена із захищеним HTTPS-з'єднанням. Додатково перевірялася ізоляція внутрішніх вузлів від зовнішніх мережевих звернень, коректність маршрутизації всередині локального сегмента, стабільність роботи DNS-резолюції та правильність застосування SSL-сертифікатів. Тестування виконувалося у локальній мережі, де вузол із Docker Desktop та контейнерами Nginx Proxy Manager мав адресу 192.168.1.2, а маршрутизатор Asus RT-N12 використовував адресу 192.168.1.1. Для доступу застосовувався один основний домен DuckDNS, на базі якого були створені окремі піддомени для різних сервісів. Такий підхід дозволив зберегти централізовану структуру і не створювати надлишкову кількість окремих DNS-записів чи сертифікатів.

Одним із перших практичних прикладів стало налаштування проксіювання панелі керування маршрутизатором. У стандартному режимі доступ до більшості домашніх маршрутизаторів здійснюється через локальну адресу 192.168.1.1 із використанням незахищеного HTTP-протоколу. Подібна схема має низку недоліків: відсутність шифрування, необхідність запам'ятовування адреси, неможливість централізованого контролю доступу та відсутність сучасних механізмів журналювання чи автентифікації на проміжному рівні. Для усунення таких обмежень у Nginx Proxy Manager було створено Proxy Host, у якому доменне ім'я піддомену маршрутизатора було прив'язане до внутрішньої адреси 192.168.1.1 та порту 80. Після збереження конфігурації всі звернення до вказаного доменного імені автоматично почали передаватися через проксі-сервер. Користувач у браузері відкривав HTTPS-адресу піддомену, запит надходив на сервер 192.168.1.2, де Nginx

Proxy Manager виконував аналіз Host-заголовка, знаходив відповідне правило та перенаправляв трафік до веб-інтерфейсу маршрутизатора. У результаті взаємодія з IP-адресою маршрутизатора більше не вимагалася, а весь доступ відбувався через єдину контрольовану точку входу. Додатковою перевагою стала можливість надалі застосувати базову автентифікацію, обмеження за IP-адресами, примусовий перехід HTTP до HTTPS, журналювання входів та інші механізми захисту. Навіть застарілий маршрутизатор без власної підтримки сучасного TLS отримав захищений спосіб доступу через проміжний проксі-рівень (див. рис. 2.20).

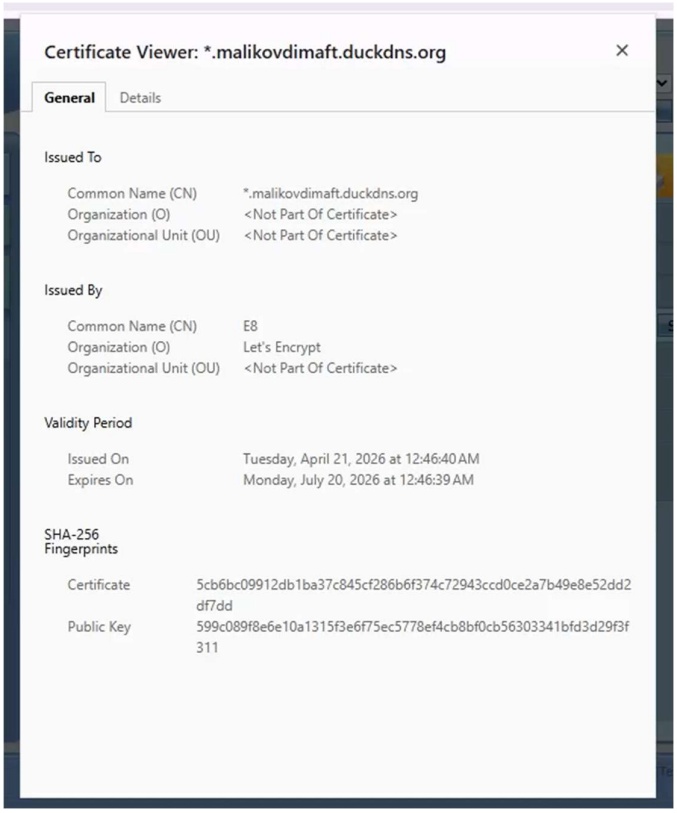


Рисунок 2.20 – Валідний SSL-сертифікат у браузері

Наступним етапом стала перевірка доступності сервісів за доменними іменами та аналіз правильності роботи SSL-сертифікатів. Для тестування використовувалися два ресурси [22]:

— адміністративна панель Nginx Proxy Manager;

— веб-інтерфейс маршрутизатора.

Після введення адреси панелі керування у браузері сторінка відкривалася із позначкою захищеного з'єднання. Протокол HTTPS визначався автоматично завдяки налаштованому перенаправленню та активному сертифікату, тому окреме введення схеми доступу не вимагалося. Під час перегляду відомостей про сертифікат було підтверджено, що він виданий центром сертифікації Let's Encrypt, має типовий строк дії 90 днів, відповідає створеному домену DuckDNS та містить wildcard-покриття для піддоменів. Логіка роботи під час такого підключення виглядала наступним чином: браузер виконував DNS-запит, отримував адресу 192.168.1.2, після чого ініціював TLS-handshake із сервером проксі. Nginx Proxy Manager надавав сертифікат, браузер перевіряв його дійсність і лише після успішної перевірки відкривав сторінку. Якщо сертифікат був би недійсним або простроченим, підключення супроводжувалося б попередженням безпеки. Практичний результат підтвердив можливість використання повноцінного HTTPS навіть у середовищі з локальними IP-адресами без придбання платного домену чи використання зовнішнього хостингу.

Практичне значення TLS-захисту всередині локальної мережі є суттєвим. Шифрування запобігає перехопленню логінів і паролів у мережі Wi-Fi, зменшує ризик компрометації cookie-файлів, виключає підміну сторінок або MITM-сценарії, дозволяє перевіряти справжність сервера та загалом підвищує рівень безпеки внутрішніх веб-сервісів. Особливо актуально це для середовищ, де до безпровідної мережі підключаються мобільні пристрої, гостьові користувачі або різноманітне IoT-обладнання [22].

Тестування ізоляції від зовнішніх мережевих запитів. Окремим напрямком тестування стала перевірка ізоляції системи від зовнішніх мережевих звернень. Домен DuckDNS був навмисно прив'язаний не до публічної адреси провайдера, а до локальної IP-адреси 192.168.1.2. Під час виконання ping-запитів до основного домену та до піддомену маршрутизатора у відповідь поверталася саме адреса проксі-сервера. Навіть якщо піддомен фактично відповідав сервісу маршрутизатора, зовнішній DNS-запис усе одно вказував лише на 192.168.1.2. Оскільки ця адреса належить до приватного діапазону RFC1918 і не

маршрутизується в глобальній мережі Інтернет, з’єднання поза локальним сегментом встановити неможливо. Зовнішній клієнт отримував тайм-аут або помилку маршрутизації. Така поведінка підтвердила природний рівень ізоляції: домен існує, сертифікат дійсний, локальний сервіс працює, однак іззовні він фактично недоступний (див. рис. 2.21).

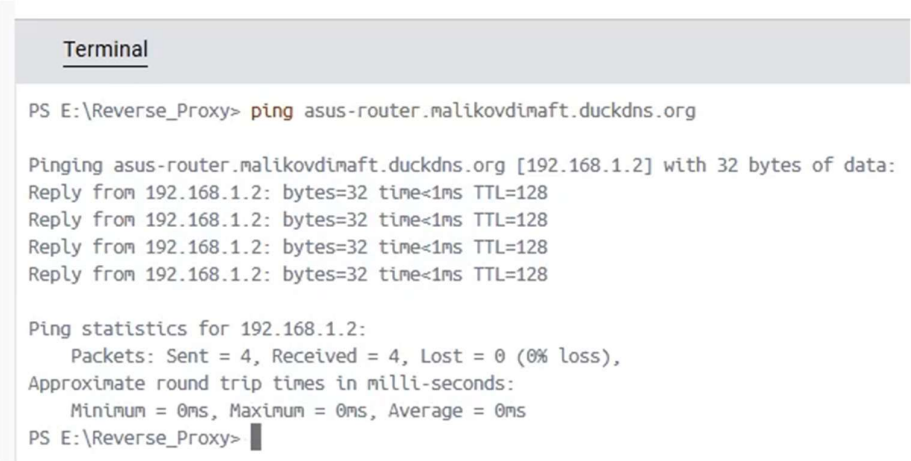


Рисунок 2.21 – Результат перевірки DNS-імені командою ping

Додатково спостерігався ефект часткового приховування внутрішньої структури мережі. Зовнішньому спостерігачу не видно реальних адрес внутрішніх сервісів, не визначається кількість вузлів локального сегмента, а всі звернення концентруються лише на одному вузлі — сервері реверс-проксі. У реальній практиці це означає зменшення інформації, яку може отримати потенційний зловмисник під час первинного сканування мережі. Внутрішня адресація маршрутизатора, файлового сховища, адміністративних панелей чи інших ресурсів не розкривається напряму (див. рис. 2.22).

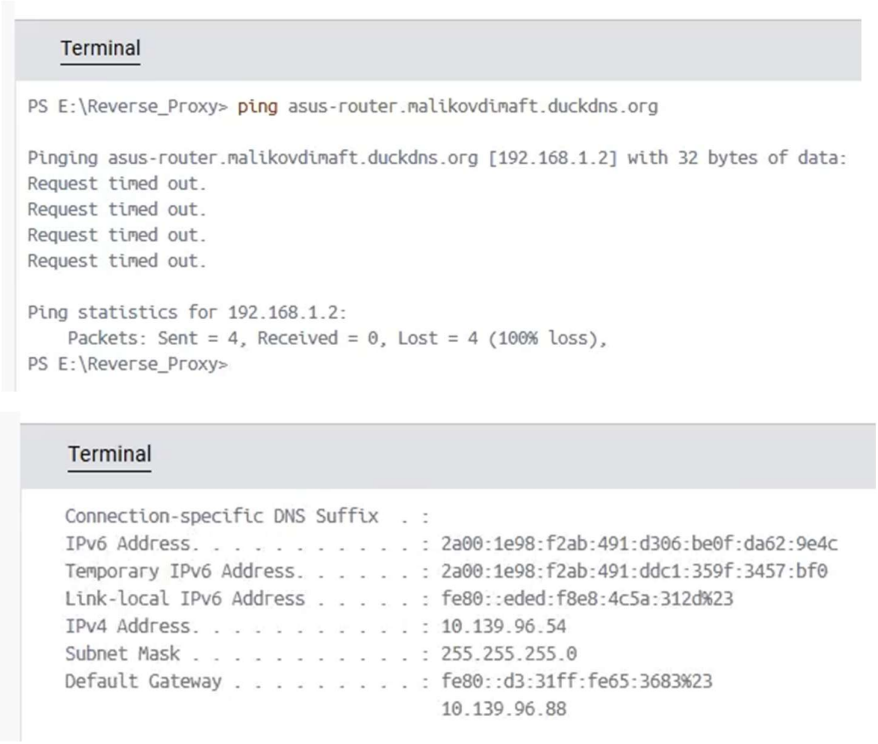


Рисунок 2.22 – Демонстрація ізоляції локального сервісу від зовнішньої мережі

Підсумкові результати тестування підтвердили, що локальні ресурси успішно відкриваються за доменними іменами, використовують валідні SSL-сертифікати, не потребують прямого звернення до IP-адрес і можуть централізовано адмініструватися через єдиний вузол доступу. Система забезпечує зручну маршрутизацію, захищене HTTPS-з’єднання, приховування внутрішніх сервісів, спрощення користування мережею та відсутність зовнішнього доступу без окремого цілеспрямованого налаштування. Отримані результати демонструють доцільність застосування Nginx Proxy Manager як універсального механізму організації безпечного доступу до сервісів у домашніх, навчальних та малих корпоративних мережах, де важливими є простота впровадження, мінімальні витрати та високий рівень контролю над локальною інфраструктурою.

3. АНАЛІЗ ЕФЕКТИВНОСТІ ТА НАПРЯМКИ ПОКРАЩЕННЯ ПРОКСІ-СИСТЕМИ

3.1 Методи оптимізації роботи Nginx Proxy Manager

Після завершення розгортання, початкового налаштування та практичного тестування системи доступу до внутрішніх ресурсів на базі Docker і Nginx Proxy Manager наступним логічним етапом стає оцінка ефективності отриманого рішення. Якщо попередні розділи були зосереджені переважно на побудові архітектури, запуску інфраструктури та перевірці базової працездатності, то на цьому етапі основна увага приділяється стабільності функціонування, раціональному використанню ресурсів, типових проблемах експлуатації та способах їх усунення. Навіть у локальному середовищі проксі-сервер залишається критично важливим компонентом, оскільки саме через нього проходять усі HTTP та HTTPS-запити клієнтів, а від швидкості його роботи, коректності конфігурації, доступності контейнера та продуктивності залежить загальна зручність використання сервісів, час відкриття сторінок і надійність доступу до внутрішніх ресурсів.

Під час практичної експлуатації було встановлено, що система загалом працює стабільно, однак присутні окремі технічні нюанси, які потребують додаткової оптимізації. До таких аспектів належать затримки під час генерації SSL-сертифікатів через DNS Challenge, обмеження ресурсів локального комп’ютера, вплив великих файлів або повільних відповідей внутрішніх сервісів на стабільність з’єднання, а також доцільність кешування статичних даних для зменшення повторного навантаження. Nginx Proxy Manager поєднує зручний графічний інтерфейс адміністрування з можливостями повноцінного Nginx-сервера, тому навіть за простоти використання система дозволяє виконувати гнучке налаштування параметрів продуктивності. Практика також показала доцільність

використання одного домену та одного wildcard-сертифіката для декількох сервісів, оскільки такий підхід зменшує кількість окремих сертифікатів, спрощує керування DNS-записами, скорочує кількість відкритих портів і полегшує масштабування інфраструктури.

Вирішення проблем з таймаутами та великими пакетами даних. Одним із найпоширеніших напрямків оптимізації є усунення проблем із таймаутами та передаванням великих пакетів даних. У процесі роботи проксі-сервера можуть виникати ситуації, коли внутрішній сервіс повертає відповідь занадто довго або передає значний обсяг інформації. Подібні випадки характерні для веб-панелей адміністрування, файлових сховищ, систем відеоспостереження, резервного копіювання, завантаження архівів або обробки великих конфігураційних сторінок. Якщо стандартні часові межі очікування перевищуються, користувач може отримати помилки типу "504 Gateway Timeout", "Upstream Timeout", "Request Timed Out" або розрив з’єднання під час завантаження сторінки [23].

У тестовому середовищі подібні ситуації могли виникати під час звернення до повільних локальних пристроїв, зокрема до застарілого маршрутизатора, а також до сервісів, розгорнутих на слабкому обладнанні з обмеженим процесором або повільним накопичувачем. Для усунення таких проблем рекомендовано коригування стандартних параметрів Nginx, зокрема "proxy_connect_timeout", "proxy_send_timeout", "proxy_read_timeout" та "client_max_body_size". Збільшення часу очікування дозволяє проксі-серверу довше очікувати відповідь від внутрішнього вузла без передчасного завершення сесії, а розширення допустимого розміру тіла запиту дає можливість передавати великі файли без штучних обмежень [17; 23].

Логіка обробки в такому випадку є простою: після надходження запиту система перевіряє розмір передаваних даних, оцінює допустимість параметрів і, якщо обмеження не перевищені, передає запит внутрішньому сервісу та очікує відповідь у межах встановленого часу. Якщо ж файл перевищує дозволений розмір або вузол не відповідає навіть після збільшеного таймауту, користувач отримує контрольовану помилку замість хаотичного обриву сесії. Після коригування

параметрів у практичному використанні зникали випадкові відключення, великі сторінки відкривалися стабільніше, файли передавалися без помилок, а кількість повторних запитів зменшувалася (див. рис. 3.1).

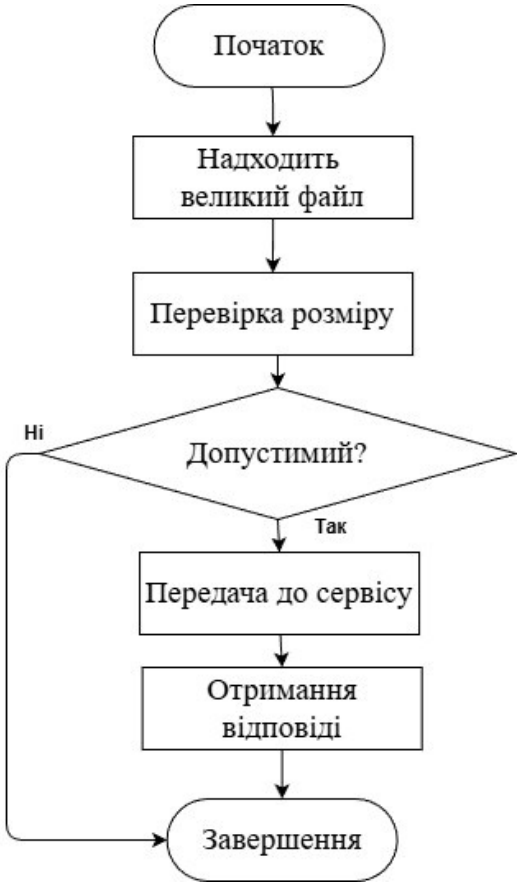


Рисунок 3.1 – Вирішення таймаутів для стабільності з’єднання

Для локальних мереж така оптимізація має особливе значення, оскільки внутрішні сервіси часто працюють не на серверному обладнанні, а на маршрутизаторах, старих комп’ютерах, міні-ПК, одноплатних системах або інших пристроях із невисокою продуктивністю. У подібних умовах стандартні параметри, орієнтовані на швидкі сервери дата-центрів, не завжди відповідають реальним можливостям локальної інфраструктури. Саме тому гнучке налаштування таймаутів і лімітів передавання даних дозволяє адаптувати проксі-систему до слабших вузлів без необхідності заміни обладнання. У результаті навіть бюджетне

або застаріле середовище може забезпечувати стабільний і зручний доступ до внутрішніх ресурсів через єдину захищену точку входу.

Оптимізація використання ресурсів контейнером на локальному ПК. Ураховуючи той факт, що система була розгорнута не на окремому сервері, а на звичайному ноутбуці, питання ефективного використання апаратних ресурсів набуло особливого значення. Контейнери Docker самі по собі значно легші за класичні віртуальні машини, оскільки не потребують окремого гостьового ядра операційної системи та використовують спільне середовище хоста, однак навіть за такої моделі при неправильному налаштуванні можливо спровокувати надмірне споживання пам’яті, зайве навантаження на процесор, накопичення службових даних на диску та загальне погіршення швидкодії комп’ютера. Особливо помітно це проявляється на слабших ноутбуках, старих персональних комп’ютерах або системах, де одночасно працюють інші ресурсоємні програми.

Серед найбільш поширених проблем у подібному середовищі спостерігаються повільний старт контейнерів після перезавантаження системи, підвищене використання оперативної пам’яті, зайве навантаження на накопичувач через журнали, кеші та тимчасові файли, конфлікти з іншими програмами, які також активно використовують ресурси, а також затримки під час відкриття панелі керування Nginx Proxy Manager. Якщо ноутбук використовується одночасно як робочий пристрій і як локальний сервер, такі фактори стають особливо відчутними, оскільки будь-яке перевантаження впливає як на користувацькі задачі, так і на стабільність мережевої інфраструктури (див. рис. 3.2) [24].

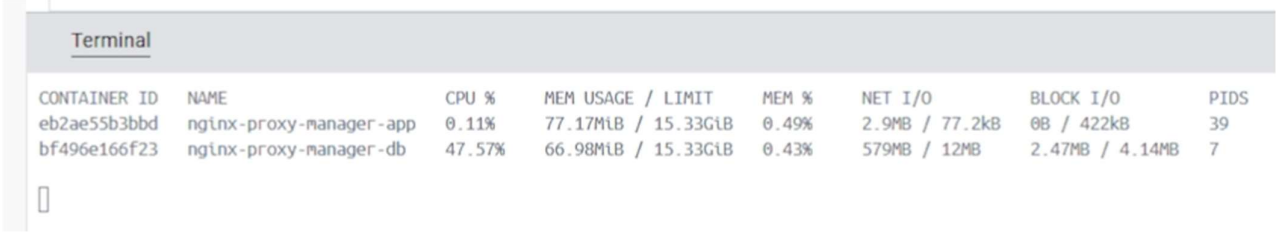


Рисунок 3.2 – Використання ресурсів контейнером до оптимізації

Для зменшення навантаження доцільно використовувати лише ті контейнери, які реально необхідні для роботи системи. Якщо в Docker-середовищі одночасно розгорнуті додаткові тестові сервіси, бази даних, експериментальні веб-застосунки або непотрібні фонові модулі, вони споживають ресурси навіть без активного використання. Тому доцільним є вимкнення зайвих контейнерів, зупинка тимчасових сервісів та запуск лише основного стеку, який складається з Nginx Proxy Manager і необхідної бази даних. Подібний підхід одразу зменшує використання оперативної пам’яті та скорочує фонове навантаження на систему.

Додатковим інструментом оптимізації є налаштування обмежень ресурсів у Docker Desktop. У середовищі керування можна задати максимальний обсяг оперативної пам’яті, кількість доступних процесорних ядер та інші параметри, що дозволяє запобігти ситуації, коли контейнерна інфраструктура починає займати надмірну частину ресурсів ноутбука. Завдяки цьому зберігається баланс між стабільністю роботи сервісів і комфортною роботою основної операційної системи.

Важливий вплив має також тип накопичувача. Зберігання контейнерних даних, бази даних MariaDB, конфігураційних файлів та логів на SSD-диску значно прискорює запуск сервісів, запис журналів, оновлення контейнерів та загальну реакцію веб-панелі. На класичних HDD-дисках система також працюватиме, однак затримки під час старту контейнерів, відкриття панелі керування або інтенсивних операцій введення-виведення будуть помітно більшими. Для локального серверного використання навіть бюджетний SSD суттєво підвищує комфорт експлуатації.

Корисною процедурою залишається регулярне очищення старих образів, тимчасових контейнерів, кешів збірки та невикористовуваних томів. У процесі оновлення Docker часто накопичує застарілі версії образів, які більше не використовуються, але займають місце на диску. З часом це може створювати значний обсяг непотрібних даних, уповільнювати систему та ускладнювати обслуговування середовища. Періодичне очищення дозволяє підтримувати акуратну структуру сховища та уникати дефіциту вільного простору.

Практичні результати після базової оптимізації свідчать, що система здатна працювати стабільно навіть у фоновому режимі на звичайному ноутбуці без відчутного дискомфорту для користувача. Панель керування відкривається швидше, контейнери запускаються стабільніше, а загальне навантаження на систему зменшується. Такий досвід підтверджує можливість використання подібної архітектури без дорогого серверного обладнання та демонструє високу гнучкість контейнерного підходу. Одночасно стає очевидним, що за потреби інфраструктура може бути легко перенесена на міні-ПК, одноплатний комп'ютер, домашній сервер, маршрутизатор із Linux або інший пристрій, здатний працювати у режимі 24/7 (див. рис. 3.3).

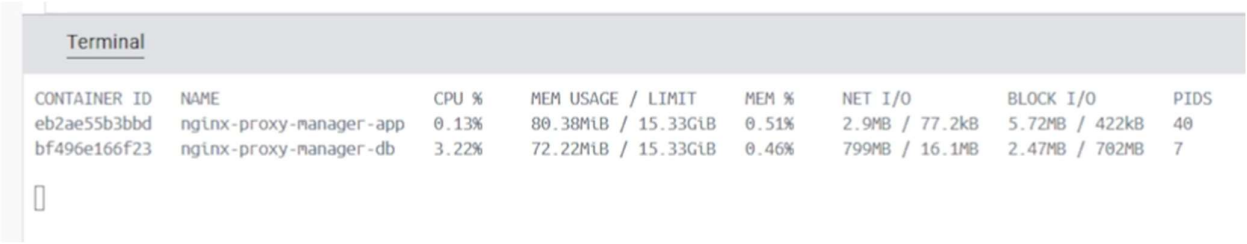


Рисунок 3.3 – Використання ресурсів контейнером після оптимізації

Налаштування кешування статичного контенту на рівні проксі. Ще одним ефективним способом підвищення продуктивності є кешування статичних ресурсів безпосередньо на рівні проксі-сервера. До таких ресурсів зазвичай належать CSS-файли, JavaScript-скрипти, зображення, шрифти, статичні HTML-компоненти та інші елементи, які рідко змінюються й можуть багаторазово використовуватися при повторному відкритті сторінок. У стандартному режимі без кешування браузер або внутрішній сервіс змушений щоразу повторно передавати однакові файли, навіть якщо вони вже були отримані раніше, що створює зайве навантаження на мережу та локальні пристрої.

Принцип роботи кешування є достатньо простим. Після надходження клієнтського запиту проксі-сервер перевіряє наявність потрібного файлу у власному кеші. Якщо ресурс уже збережений і термін його актуальності не завершився, відбувається швидка видача без повторного звернення до

внутрішнього сервісу. Якщо файл відсутній або застарів, проксі звертається до цільового вузла, отримує ресурс, передає його клієнту та одночасно зберігає копію у кеші для наступних звернень. Надалі цикл повторюється, а кількість зайвих запитів зменшується (див. рис. 3.4) [25].

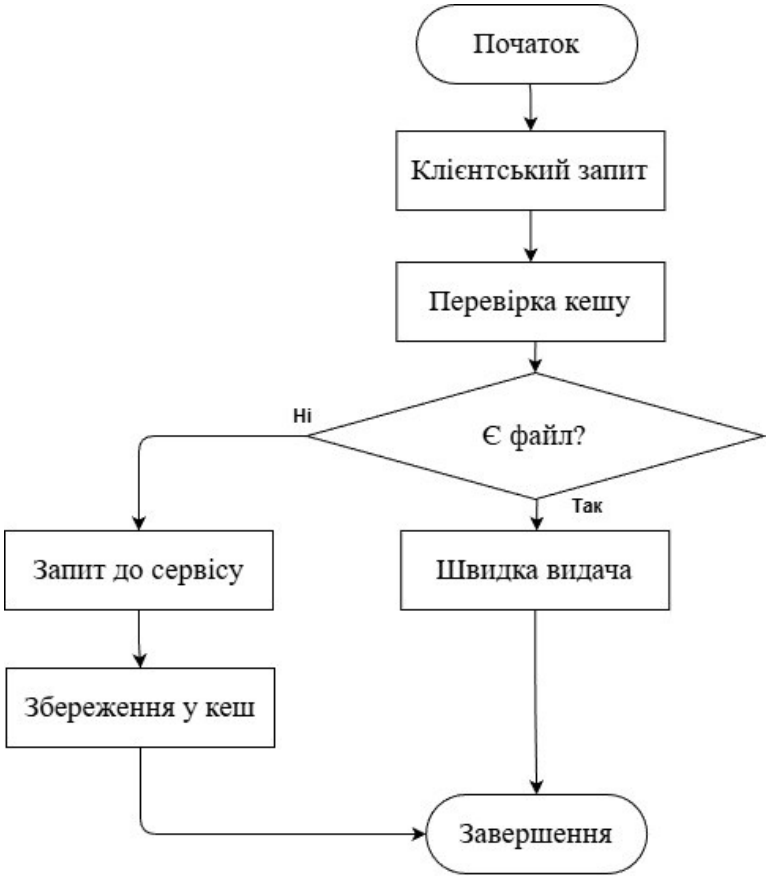


Рисунок 3.4 – Алгоритм кешування статичного контенту

Переваги такого підходу особливо помітні у локальних мережах із малопотужними пристроями. Сторінки відкриваються швидше, зменшується навантаження на маршрутизатори, старі веб-інтерфейси працюють стабільніше, кількість повторних HTTP-запитів скорочується, а ресурси внутрішніх вузлів використовуються раціональніше. Якщо, наприклад, сторінка конфігурації маршрутизатора містить однакові таблиці стилів, скрипти та графічні елементи, немає потреби завантажувати їх заново при кожному переході між вкладками.

Проксі може віддати вже збережені дані значно швидше, ніж сам маршрутизатор [25].

Особливо корисним кешування стає для застарілих пристроїв, де веб-сервер має слабкий процесор і повільну пам’ять. У такому випадку навіть невелике зменшення кількості запитів позитивно впливає на швидкодію інтерфейсу. Крім того, знижується навантаження на безпровідний сегмент мережі, якщо клієнти використовують Wi-Fi-з’єднання.

Водночас кешування потребує обережного застосування. Його не слід використовувати для сторінок авторизації, адміністративних панелей із динамічним вмістом, персоналізованих кабінетів користувачів, статистики в реальному часі або будь-яких даних, які повинні миттєво оновлюватися. У протилежному випадку можливе відображення застарілої інформації, помилки сесій або некоректна робота інтерактивних елементів. Саме тому доцільно кешувати лише статичні ресурси, залишаючи динамічний контент безпосередньо за внутрішнім сервісом.

У підсумку налаштування кешування на рівні Nginx Proxy Manager дозволяє не лише прискорити відкриття сторінок, а й зменшити навантаження на локальну інфраструктуру, що особливо важливо у середовищах із бюджетним або застарілим обладнанням. Поєднання кешування з правильно підібраними таймаутами та оптимізованим використанням ресурсів контейнера формує стабільну й ефективну модель роботи всієї проксі-системи.

3.2 Порівняння із традиційними підходами до маршрутизації

Побудова системи доступу на базі Docker та Nginx Proxy Manager доцільна не лише з позиції технічної реалізації, а й з точки зору зіставлення з іншими поширеними моделями організації мережевого доступу. У сучасних локальних мережах найчастіше застосовуються декілька підходів, серед яких класичне

прокидання портів, прямий доступ за внутрішніми IP-адресами, використання VPN-рішень або централізований реверс-проксі. У межах даної реалізації проведено порівняльний аналіз розробленої системи з традиційними методами за такими критеріями, як рівень безпеки, зручність додавання нових сервісів, підтримка HTTPS, приховування внутрішньої структури мережі, простота адміністрування та вимоги до технічної підготовки адміністратора. Отримані результати свідчать про те, що використання Nginx Proxy Manager є збалансованим компромісом між простотою налаштування, функціональністю та рівнем захисту, особливо для домашніх серверів, навчальних середовищ, невеликих офісів і локальних інфраструктур, де необхідно швидко організувати впорядкований доступ до внутрішніх ресурсів без значних фінансових витрат.

Зіставлення розробленого рішення з класичним прокиданням портів (Port Forwarding). Класичне прокидання портів полягає у створенні правил на маршрутизаторі, які перенаправляють вхідний або внутрішній трафік з певного порту на конкретний пристрій локальної мережі. Наприклад, порт 80 може бути спрямований на веб-сервер, порт 443 на панель керування сервісом, інші порти — на додаткові ресурси. Подібний підхід тривалий час залишався стандартним способом надання доступу до внутрішніх вузлів, однак зі збільшенням кількості сервісів починає проявляти низку суттєвих обмежень.

Насамперед ускладнюється сама структура правил маршрутизатора, оскільки для кожного нового сервісу необхідно окремо резервувати порт, перевіряти відсутність конфліктів, контролювати правильність перенаправлення та уникати дублювання налаштувань. Чим більше сервісів додається до мережі, тим вищою стає ймовірність помилок конфігурації, випадкового відкриття небажаних служб або створення правил, які перетинаються між собою. Додатковою проблемою є відсутність централізованої HTTPS-моделі, оскільки для кожного ресурсу окремо потрібно налаштовувати TLS, отримувати сертифікати, контролювати строки їх дії та оновлювати конфігурації. У багатьох випадках це призводить до використання звичайного HTTP або застарілих самопідписаних сертифікатів.

Ще одним недоліком прямого прокидання портів залишається відкриття внутрішньої структури мережі. Під час доступу часто стають очевидними реальні адреси вузлів, наприклад 192.168.1.1 для маршрутизатора, 192.168.1.2 для серверного вузла, інші адреси для додаткових пристроїв. Така схема не лише менш зручна для користувача, а й спрощує розуміння внутрішньої топології мережі.

У запропонованому рішенні використовується лише один центральний вузол доступу до реверс-проксі сервер, розміщений на локальному хості. Саме він приймає всі клієнтські звернення та розподіляє їх між внутрішніми сервісами згідно з доменними іменами. Замість запам’ятовування портів і адрес застосовуються логічні імена на зразок "npm.malikovdimaft.duckdns.org", "asus-router.malikovdimaft.duckdns.org" та інші подібні варіанти. Для користувача така модель значно простіша, оскільки взаємодія відбувається через зрозумілі доменні назви, а вся складність маршрутизації прихована всередині системи.

Окремою перевагою є централізоване керування HTTPS. Один проксі-сервер може обслуговувати декілька сервісів, автоматично випускати та оновлювати сертифікати, забезпечувати перенаправлення HTTP на HTTPS, вести журнали доступу, застосовувати додаткову автентифікацію або фільтрацію за IP-адресами. У випадку додавання нового ресурсу достатньо створити новий Proxy Host у веб-панелі без зміни правил маршрутизатора або відкриття нових портів.

Порівняння Nginx Proxy Manager з Traefik та Caddy. Серед сучасних рішень для організації реверс-проксі існує декілька популярних платформ, які активно застосовуються як у локальних мережах, так і в корпоративному середовищі. Поширення подібних інструментів пояснюється тим, що реверс-проксі на сьогодні є одним із найбільш актуальних способів централізації доступу до сервісів, підвищення безпеки, впорядкування маршрутизації запитів та впровадження HTTPS без складної перебудови всієї інфраструктури. Серед найбільш відомих рішень найчастіше розглядаються Nginx Proxy Manager, Traefik та Caddy, кожне з яких має власні переваги, обмеження та орієнтоване на різні сценарії використання.

У випадку Nginx Proxy Manager основною перевагою є наявність зручного веб-інтерфейсу, через який більшість адміністративних дій виконується без

ручного редагування конфігураційних файлів. Додавання нового сервісу, створення Proxy Host, налаштування доменного імені, активація HTTPS, генерація сертифікатів та зміна параметрів маршрутизації здійснюються у декілька кроків через графічну панель керування. Такий підхід значно спрощує розгортання системи, особливо у випадках, коли необхідно швидко отримати працездатний результат без детального занурення у синтаксис конфігурацій Nginx. Окремою перевагою є зрозуміла структура налаштувань, завдяки чому конфігурацію легко повторно відтворити на іншому пристрої або перенести в нове середовище. До недоліків можна віднести дещо меншу гнучкість порівняно з більш спеціалізованими платформами та орієнтацію переважно на GUI-модель керування, що не завжди є оптимальним у великих автоматизованих інфраструктурах.

Traefik вважається потужнішим рішенням для динамічних середовищ контейнеризації. Однією з його головних переваг є автоматичне виявлення контейнерів Docker, інтеграція з Kubernetes, підтримка динамічного створення маршрутів, балансування навантаження та високий рівень масштабованості. У складних системах, де сервіси постійно створюються, змінюються або перезапускаються, Traefik демонструє значні переваги завдяки автоматизації. Разом із цим налаштування такого рішення суттєво складніше, оскільки вимагає розуміння "labels", "routers", "services", "middleware" та загальної логіки внутрішньої взаємодії компонентів. Для початкового рівня або тестових локальних реалізацій подібний інструмент часто є надмірним, а витрати часу на впровадження можуть перевищувати практичну користь. Саме тому Traefik доцільніше застосовувати там, де потрібне масштабоване середовище з великою кількістю контейнеризованих сервісів і високим рівнем автоматизації.

Caddy також належить до популярних рішень реверс-проксі та привертає увагу своєю простотою. Його конфігураційний файл зрозумілий, структура записів лаконічна, запуск системи відбувається швидко, а автоматичне отримання та оновлення сертифікатів реалізоване дуже зручно. Для невеликих проєктів, домашніх веб-сервісів або базових тестових середовищ Caddy є достатньо

привабливим варіантом. Водночас він менш поширений у порівнянні з двома попередніми рішеннями, має менше варіантів візуального адміністрування та рідше використовується у складних сценаріях централізованого керування великою кількістю сервісів (див. рис. 3.1).

Таблиця 3.1 – Порівняння Reverse Proxy-рішень Nginx Proxy Manager, Traefik, Caddy

Критерій	Nginx Proxy Manager	Traefik	Caddy
GUI	Є	Обмежений	Відсутній
Автоматичний SSL	Так	Так	Так
Складність налаштування	Низька	Висока	Низька
Docker integration	Добра	Відмінна	Добра
Масштабованість	Середня	Висока	Середня
Зручність для початківця	Висока	Низька	Висока

У межах даної реалізації основними вимогами були швидке тестове розгортання, простота адміністрування, зрозумілий інтерфейс, робота в локальній мережі та можливість використання HTTPS без складних ручних конфігурацій. Саме за таких умов Nginx Proxy Manager виявляється найбільш доцільним вибором, оскільки поєднує зручність використання, достатній набір функцій та мінімальний поріг входження. Для локальної мережі, домашнього сервера, навчальної лабораторії або малого офісу подібна модель є практичнішою, ніж складніші рішення корпоративного рівня.

Оцінка зручності адміністрування (GUI проти CLI). Окремого розгляду потребує питання способу адміністрування системи, оскільки на практиці саме зручність керування часто визначає успішність подальшої експлуатації. Класичний підхід через CLI передбачає ручне редагування конфігураційних файлів, роботу з каталогами типу "nginx.conf", "sites-enabled", "certificates", використання команд перезапуску сервісів, перегляд журналів через "docker logs", а також внесення змін у текстових редакторах на кшталт nano або vim. Такий варіант є ефективним для

досвідчених адміністраторів, які добре орієнтуються у структурі Linux-систем та конфігураціях Nginx, однак для менш підготовленого користувача він часто супроводжується ризиком синтаксичних помилок, неправильних шляхів до сертифікатів, складністю пошуку причин несправностей та значними витратами часу навіть на типові операції [26-27].

У Nginx Proxy Manager більшість дій виконується через веб-панель керування. Саме там створюються проксі-хости, генеруються SSL-сертифікати, вказуються IP-адреси внутрішніх сервісів, налаштовуються перенаправлення, активується HTTPS та додаткові параметри безпеки. Весь процес відбувається у візуальному режимі, де поля структуровані та логічно розділені. Спочатку вказується доменне ім’я, далі цільовий вузол, порт, схема підключення, після чого додається сертифікат і застосовуються додаткові параметри. Така структура суттєво зменшує кількість технічних помилок.

Серед переваг GUI-адміністрування насамперед варто відзначити швидкість роботи, оскільки новий сервіс може бути доданий протягом кількох хвилин без ручного редагування файлів. Другим фактором є зниження ризику неправильних налаштувань через контрольовану структуру введення даних. Третім аспектом виступає простота супроводу системи в майбутньому, адже навіть через тривалий час структура сервісів легко читається без необхідності аналізувати великі конфігураційні файли. Для локальних мереж і невеликих інфраструктур це особливо важливо, оскільки адміністрування часто виконується періодично, а не на постійній основі.

У підсумку запропонована система доступу на базі Docker та Nginx Proxy Manager перевершує традиційне прокидання портів за рівнем зручності, масштабованості та безпеки. У порівнянні з Traefik і Caddy розроблене рішення виглядає найбільш збалансованим саме з позиції поєднання функціональності та простоти впровадження. Для локальних мереж, домашніх серверів, навчальних лабораторій і малих офісів модель Nginx Proxy Manager зі DuckDNS та Docker є одним із найбільш доцільних і практичних варіантів організації централізованого доступу до внутрішніх сервісів [26-27].

3.3 Економічний ефект та вектори модернізації

Розроблена система доступу до внутрішніх сервісів на базі Docker, Nginx Proxy Manager та DuckDNS має не лише технічні переваги, що були розглянуті раніше, але й виражений економічний ефект. В умовах малих організацій, домашніх лабораторій, навчальних середовищ або невеликих офісів організація безпечного та централізованого доступу до ресурсів часто обмежується фінансовими можливостями, тому використання рішень, що не потребують значних витрат, набуває особливої актуальності. Запропонована модель базується на програмному забезпеченні з відкритим вихідним кодом, безкоштовних зовнішніх сервісах та можливості застосування вже наявного обладнання, такого як ноутбуки, персональні комп'ютери, міні-ПК, одноплатні системи або інші пристрої, здатні працювати в постійному режимі. У межах даного підрозділу доцільно оцінити економічну цінність запропонованого рішення, а також визначити перспективні напрями його подальшої модернізації [30].

Оцінка нульової вартості реалізації (використання Open Source та безкоштовних сервісів). Загальна економічна характеристика даного рішення полягає у тому, що на відміну від багатьох комерційних платформ для організації безпечного доступу до сервісів система не потребує придбання ліцензованого програмного забезпечення, окремих апаратних шлюзів, платних сервісів керування сертифікатами чи дорогих корпоративних інструментів адміністрування. У базовому сценарії впровадження фактична вартість може дорівнювати нулю за умови використання вже наявного обладнання.

До безкоштовних компонентів, на яких побудована система, належить Docker як платформа контейнеризації, що дозволяє швидко розгортати сервіси в ізольованому середовищі без додаткових ліцензійних витрат. Nginx Proxy Manager як інструмент реверс-проксі та централізованого керування також розповсюджується безкоштовно. База даних MariaDB використовується як сховище конфігурацій та службових даних без потреби у комерційній ліцензії.

DuckDNS надає безкоштовний динамічний DNS-сервіс для створення доменних імен. Сертифікати Let’s Encrypt випускаються безкоштовно та забезпечують повноцінний HTTPS-захист. Операційною основою може виступати вже наявний Linux або Windows-хост, що додатково зменшує витрати на старті.

Якщо порівнювати з комерційними рішеннями, типова корпоративна інфраструктура може включати платні DNS-послуги, оренду доменів преміум-рівня, VPN-шлюзи, хмарні проксі-сервіси, балансувальники навантаження, комерційні сертифікати, ліцензії на керування доступом та інші регулярні платежі у вигляді щомісячної або щорічної підписки. Навіть базове впровадження подібних систем часто вимагає суттєвих стартових витрат, а також додаткових витрат на супровід і технічну підтримку.

Практичний ефект запропонованої моделі полягає у можливості побудувати повноцінну систему доступу до внутрішніх ресурсів без окремого бюджету, без щомісячних оплат та без необхідності придбання нового мережевого обладнання на початковому етапі. Додатковою перевагою є зменшення залежності від сторонніх постачальників послуг, оскільки більшість компонентів контролюється локально. Використання Docker дозволяє не встановлювати сервіси напряму в операційну систему, уникати конфліктів бібліотек, швидко виконувати повторне розгортання після збоїв або перенесення на інший пристрій, скорочувати час простою та зменшувати адміністративні витрати (див. рис. 3.5).

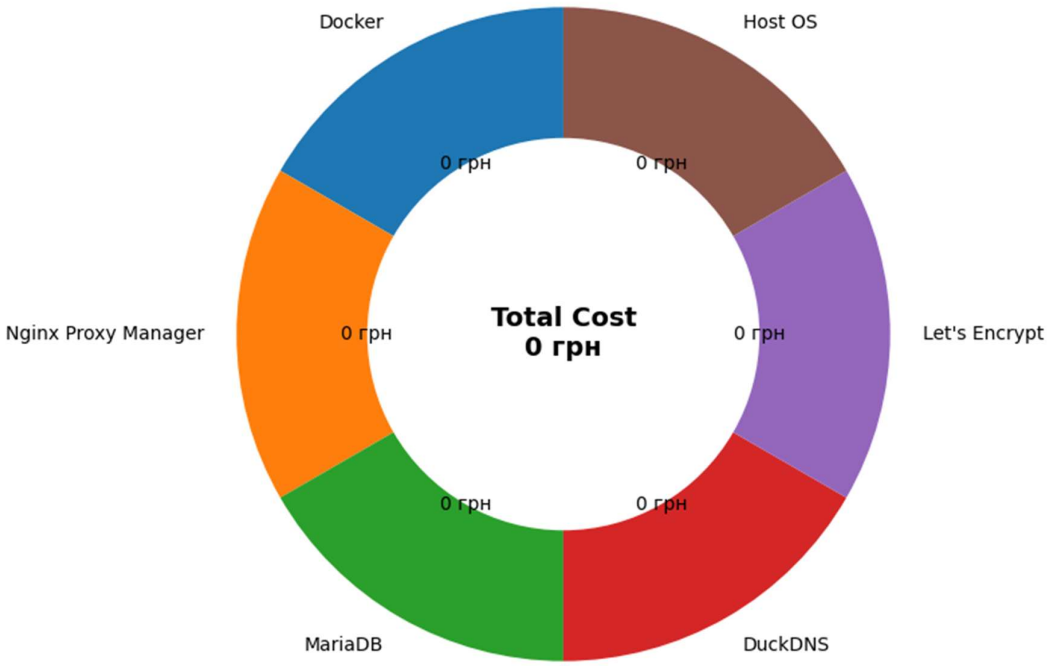


Рисунок 3.5 – Структура нульової вартості впровадження системи

Фактично така система має нульову або близьку до нульової собівартість впровадження за умови використання наявного обладнання, що робить її особливо привабливою для домашніх мереж, навчальних лабораторій, тестових середовищ і малого бізнесу, де кожне технічне рішення оцінюється не лише за функціональністю, але й за економічною доцільністю.

Перспективи додавання механізмів автентифікації на рівні проксі (Basic Auth). Перспективним напрямом подальшого розвитку системи є впровадження додаткових механізмів автентифікації безпосередньо на рівні реверс-проксі. Попри те, що вже реалізований HTTPS-захист забезпечує шифрування трафіку, захист логінів, паролів і службових даних під час передавання мережею, для повноцінного контролю доступу до внутрішніх сервісів доцільним є застосування ще одного рівня безпеки у вигляді перевірки користувача до моменту передачі запиту до внутрішнього ресурсу. Одним із найпростіших і водночас ефективних варіантів такого підходу є використання Basic Auth, що підтримується Nginx Proxy Manager та легко інтегрується у вже створену інфраструктуру.

Суть механізму полягає в тому, що після відкриття доменного імені користувач спочатку отримує запит на введення логіна та пароля, після успішної перевірки доступ дозволяється, а запит передається до цільового сервісу. У випадку неправильних облікових даних доступ блокується ще на рівні проксі, тобто внутрішній сервіс навіть не отримує звернення. Така схема особливо корисна для веб-інтерфейсів маршрутизаторів, панелей керування, адміністративних сторінок або інших ресурсів, де небажано надавати доступ лише за фактом знання адреси (див. рис. 3.6).

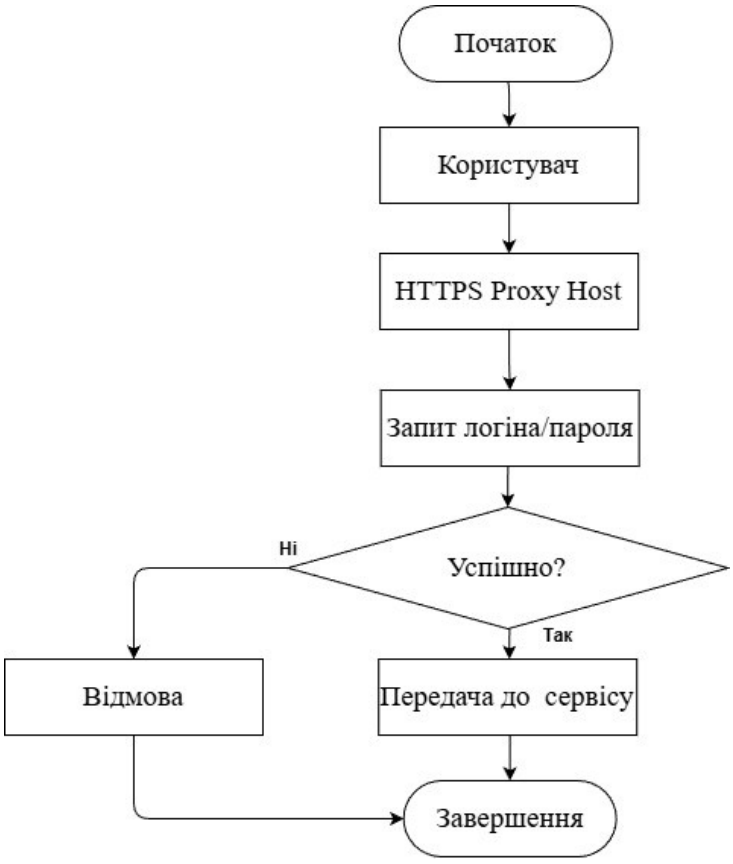


Рисунок 3.6 – Схема додавання Basic Auth на рівні проксі

Практична перевага Basic Auth полягає у тому, що навіть якщо сторонній користувач знає доменне ім’я на зразок asus-router.malikovdimaft.duckdns.org, доступ без правильного логіна та пароля не буде надано. Таким чином додається додатковий бар’єр захисту без необхідності змінювати сам внутрішній сервіс. Налаштування такого механізму виконується через графічний інтерфейс Nginx

Proxy Manager, де достатньо створити облікові дані та прив’язати їх до конкретного Proxy Host. Завдяки цьому не виникає потреби у ручному редагуванні конфігураційних файлів або використанні сторонніх модулів.

Ще однією перевагою є мінімальний вплив на продуктивність системи. Basic Auth не створює суттєвого навантаження на процесор чи оперативну пам’ять, а перевірка облікових даних відбувається швидко навіть на малопотужному обладнанні. Саме тому такий механізм може бути впроваджений навіть у домашніх мережах або тестових середовищах без ризику погіршення швидкодії.

У перспективі система може бути розширена більш сучасними методами автентифікації, серед яких LDAP-авторизація, інтеграція з Single Sign-On, використання облікових записів Google або Microsoft, а також двофакторна автентифікація. Проте саме Basic Auth залишається найбільш простим способом швидкого підвищення безпеки без ускладнення інфраструктури, особливо на початковому етапі експлуатації, коли велика кількість внутрішніх сервісів уже має власні локальні облікові записи.

Можливість міграції конфігурації на виділений апаратний мікросервер (Raspberry Pi). Під час тестування система була розгорнута на ноутбуці, що повністю виправдано на етапі перевірки працездатності, налаштування та початкового тестування. Водночас для постійної експлуатації більш доцільним рішенням є перенесення інфраструктури на окремий малопотужний пристрій, який працюватиме автономно у режимі 24/7. Найбільш придатними варіантами для такого сценарію можуть бути Raspberry Pi, Orange Pi, Mini PC, Intel NUC, thin client або інші компактні пристрої з низьким енергоспоживанням (див. рис. 3.7) [28; 30].

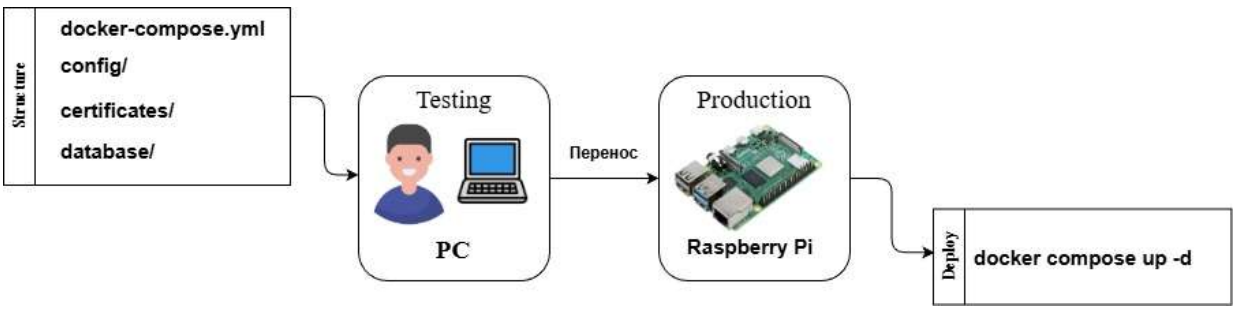


Рисунок 3.7 – Міграція конфігурації на Raspberry Pi

Основна перевага такого перенесення полягає у тому, що робочий комп’ютер або ноутбук більше не потрібно тримати постійно увімкненим лише заради роботи проксі-сервера. Виділений вузол може безперервно функціонувати поруч із маршрутизатором, виконуючи роль окремого мережевого шлюзу доступу до сервісів. Це знижує споживання електроенергії, зменшує шум, підвищує загальну стабільність та звільняє основний комп’ютер для інших завдань [28; 30].

Окремою перевагою є компактність. Подібний пристрій можна розмістити біля маршрутизатора, у шафі з мережевим обладнанням або в будь-якому іншому зручному місці без потреби у великому просторі. Для домашньої або малої офісної інфраструктури це особливо зручно, оскільки мережевий вузол стає майже непомітним і працює постійно без додаткового втручання.

Міграція системи виконується швидко завдяки контейнерному підходу. Оскільки вся інфраструктура описана у docker-compose.yml, перенесення зазвичай зводиться до копіювання каталогу проєкту разом із файлами конфігурації, змонтованими директоріями з даними та сертифікатами, встановлення Docker на новому пристрої й запуску контейнерів стандартною командою. Після старту система автоматично відновлюється з усіма доменами, сертифікатами, правилами маршрутизації та параметрами доступу.

Подібний процес можна порівняти з класичним переходом від тестового середовища до продуктивного. Спочатку рішення перевіряється на тимчасовому обладнанні, після чого переноситься на постійний вузол, де вже працює як повноцінний сервіс. Саме контейнеризація робить таку модель максимально зручною, оскільки перенесення займає мінімум часу, не потребує повторного ручного налаштування та дозволяє швидко відновлювати роботу навіть у разі заміни обладнання.

Практичний сценарій використання виглядає таким чином: на ноутбуці створюється тестова інфраструктура, перевіряється маршрутизація, сертифікати, доступність сервісів, після чого вся система переноситься на Raspberry Pi або інший мікросервер. Пристрій встановлюється біля маршрутизатора, працює автономно та постійно забезпечує централізований доступ до внутрішніх ресурсів.

Отже, міграція на окремий мікросервер є логічним етапом розвитку побудованої системи. Такий крок підвищує стабільність, автономність, енергоефективність і готовність інфраструктури до подальшого масштабування. Загалом уся архітектура демонструє виражений економічний ефект завдяки використанню Open Source-компонентів та безкоштовних сервісів, забезпечуючи майже нульову вартість впровадження при збереженні функціональності, характерної для значно дорожчих комерційних рішень. Подальший розвиток можливий у трьох основних напрямах [30]:

1. посилення безпеки через автентифікацію на рівні проксі;
2. перенесення інфраструктури на енергоефективний виділений вузол;
3. масштабування до повноцінного продуктивного середовища.

Саме це робить розроблену систему перспективною як для навчального, так і для практичного використання у сфері доступу до внутрішніх ресурсів локальної мережі.

ВИСНОВКИ

У ході виконання практичної частини було спроєктовано архітектуру захищеного доступу до внутрішніх сервісів локальної мережі із застосуванням Docker та Nginx Proxy Manager. Побудована модель передбачає використання єдиної точки входу, через яку здійснюється доступ до веб-ресурсів за зрозумілими доменними іменами без необхідності прямого звернення до внутрішніх IP-адрес і портів, що спрощує адміністрування мережевих ресурсів.

У процесі реалізації було підготовлено контейнерне середовище, налаштовано домен через DuckDNS та розгорнуто Nginx Proxy Manager із подальшим створенням SSL-сертифікатів Let’s Encrypt. Використання DNS Challenge дозволило отримати валідні сертифікати навіть для локального середовища без придбання платного домену чи зовнішнього хостингу. У результаті внутрішні сервіси отримали підтримку HTTPS-з’єднання та автоматичне оновлення сертифікатів.

Проведене тестування підтвердило коректність маршрутизації запитів до цільових ресурсів, працездатність механізму підміни службових заголовків і стабільне відкриття внутрішніх веб-інтерфейсів через доменні адреси. Додатково встановлено, що прив’язка домену до приватної IP-адреси забезпечує природну ізоляцію від зовнішніх мережевих запитів, унаслідок чого внутрішні ресурси залишаються недоступними з Інтернету.

Аналіз ефективності показав, що запропонована система має низьку вартість впровадження завдяки використанню відкритого програмного забезпечення та вже наявного обладнання. Оптимізація таймаутів, ресурсів контейнерів і кешування дозволяє підвищити стабільність роботи навіть на звичайному ноутбуці. Порівняно з класичним Port Forwarding рішення забезпечує вищий рівень безпеки та зручності, а подальший розвиток можливий шляхом додавання Basic Auth, перенесення інфраструктури на енергоефективний мікросервер і масштабування до повноцінного продуктивного середовища.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. NGINX, Inc. What Is a Reverse Proxy? URL: https://www.nginx.com/resources/glossary/reverse-proxy-server/ (дата звернення: 25.02.2026).
2. Cloudflare. Forward Proxy vs Reverse Proxy. URL: https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/ (дата звернення: 01.03.2026).
3. MDN Web Docs. HTTP headers. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers (дата звернення: 03.03.2026).
4. ISC. Split DNS Explained. URL: https://www.isc.org/blogs/split-dns/ (дата звернення: 04.03.2026).
5. Duck DNS. Duck DNS Free Dynamic DNS hosted on Amazon VPC. URL: https://www.duckdns.org/ (дата звернення: 05.03.2026).
6. No-IP. What is Dynamic DNS? URL: https://www.noip.com/support/knowledgebase/what-is-dynamic-dns/ (дата звернення: 07.03.2026).
7. DigiCert. What is PKI? URL: https://www.digicert.com/faq/public-trust-and-certificates/what-is-pki (дата звернення: 09.03.2026).
8. Cloudflare. What is SSL/TLS? URL: https://www.cloudflare.com/learning/ssl/what-is-ssl/ (дата звернення: 11.03.2026).
9. Let’s Encrypt. About Let’s Encrypt. URL: https://letsencrypt.org/about/ (дата звернення: 11.03.2026).
10. Let’s Encrypt. Challenge Types. URL: https://letsencrypt.org/docs/challenge-types/ (дата звернення: 14.03.2026).
11. Docker Inc. What is a Container? URL: https://www.docker.com/resources/what-container/ (дата звернення: 15.03.2026).
12. Docker Docs. Networking overview. URL: https://docs.docker.com/network/ (дата звернення: 15.03.2026).
13. Docker Docs. Docker Desktop Overview. URL: https://docs.docker.com/desktop/ (дата звернення: 18.03.2026).
14. Cisco. Man-in-the-Middle Attack Overview. URL: https://www.cisco.com/c/en/us/products/security/what-is-man-in-the-middle-attack.html (дата звернення: 19.03.2026).
15. Microsoft. Zero Trust security model. URL: https://www.microsoft.com/security/business/zero-trust (дата звернення: 23.03.2026).
16. Nginx Proxy Manager. Official Documentation. URL: https://nginxproxymanager.com/guide/ (дата звернення: 25.03.2026).

17. GitHub. Nginx Proxy Manager Repository. URL: https://github.com/NginxProxyManager/nginx-proxy-manager (дата звернення: 27.03.2026).
18. Docker Docs. Compose file reference. URL: https://docs.docker.com/compose/compose-file/ (дата звернення: 03.04.2026).
19. LinuxServer.io. DuckDNS Docker Container. URL: https://docs.linuxserver.io/images/docker-duckdns/ (дата звернення: 05.04.2026).
20. NGINX, Inc. Load Balancing and Request Routing. URL: https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/ (дата звернення: 06.04.2026).
21. NGINX, Inc. Using the X-Forwarded-For Header. URL: https://www.nginx.com/resources/wiki/start/topics/examples/forwarded/ B(дата звернення: 10.04.2026).
22. SSL Labs. SSL Server Test. URL: https://www.ssllabs.com/ssltest/ (дата звернення: 13.04.2026).
23. NGINX Docs. proxy_connect_timeout Directive. URL: https://nginx.org/en/docs/http/ngx_http_proxy_module.html (дата звернення: 16.04.2026).
24. Docker Docs. Resource constraints. URL: https://docs.docker.com/config/containers/resource_constraints/ (дата звернення: 18.04.2026).
25. NGINX Docs. Content Caching. URL: https://docs.nginx.com/nginx/admin-guide/content-cache/content-caching/ (дата звернення: 19.04.2026).
26. Traefik Labs. Traefik Proxy Documentation. URL: https://doc.traefik.io/traefik/ (дата звернення: 21.04.2026).
27. Caddy Server. Documentation. URL: https://caddyserver.com/docs/ (дата звернення: 23.04.2026).
28. Raspberry Pi Foundation. Raspberry Pi Documentation. URL: https://www.raspberrypi.com/documentation/ (дата звернення: 27.04.2026).
29. Маліков Д.В., Торошанко Я.І. Система контролю доступу до внутрішніх сервісів на базі Reverse Proxy та безкоштовних DNS-сервісів. «Сучасний стан та перспективи розвитку IoT» : VII Науково-техн. конф., м. Київ, 20 квітня 2026 р. С.101-103.
30. Маліков Д.В., Торошанко Я.І. Контейнеризована інфраструктура безпечного доступу до внутрішніх сервісів на базі Docker, Reverse Proxy та Let’s Encrypt. «Застосування програмного забезпечення в ІКТ» : VII Всеукр. науково-техн. конф., м. Київ, 23 квітня 2026 р. (подано до публікації).

Додаток А

Фрагмент файлу docker-compose.yml для розгортання Nginx Proxy Manager

```
services:
  nginx-proxy-manager-app:
    image: jc21/nginx-proxy-manager:latest
    container_name: nginx-proxy-manager-app
    environment:
      - DB_MYSQL_HOST=${NPM_DB_MYSQL_HOST}                    # DB_MYSQL_HOST:
nginx-proxy-manager-db
      - DB_MYSQL_PORT=${NPM_DB_MYSQL_PORT}                    # DB_MYSQL_PORT: 3306
      - DB_MYSQL_USER=${NPM_DB_MYSQL_USER}                    # DB_MYSQL_USER: npm
      - DB_MYSQL_PASSWORD=${NPM_DB_MYSQL_PASSWORD}            # DB_MYSQL_PASSWORD:
changeme
      - DB_MYSQL_NAME=${NPM_DB_MYSQL_NAME}                    # DB_MYSQL_NAME: npm
    volumes:
      - ./npm-app/config:/data
      - ./npm-app/letsencrypt:/etc/letsencrypt
    depends_on:
      - nginx-proxy-manager-db
    networks:
      - reverse-proxy
    ports:
      - 80:80
      - 443:443
      - 81:81
    restart: always

  nginx-proxy-manager-db:
    image: jc21/mariadb-aria:latest
    container_name: nginx-proxy-manager-db
    environment:
      - MYSQL_ROOT_PASSWORD=${NPM_DB_MYSQL_ROOT_PASSWORD}     #
MYSQL_ROOT_PASSWORD: changeme
      - MYSQL_DATABASE=${NPM_DB_MYSQL_NAME}                   # MYSQL_DATABASE: npm
      - MYSQL_USER=${NPM_DB_MYSQL_USER}                       # MYSQL_USER: npm
      - MYSQL_PASSWORD=${NPM_DB_MYSQL_PASSWORD}               # MYSQL_PASSWORD:
changeme
      - MARIADB_AUTO_UPGRADE=1
    volumes:
      - ./npm-db/config:/var/lib/mysql
    networks:
      - reverse-proxy
    restart: always

networks:
  reverse-proxy:
    name: reverse-proxy
    driver: bridge
```

Додаток Б

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра комп'ютерної інженерії



КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

НА ТЕМУ: «СИСТЕМА КОНТРОЛЮ ДОСТУПУ ДО ВНУТРІШНІХ СЕРВІСІВ НА БАЗІ ТЕХНОЛОГІЙ REVERSE PROXY ТА LET'S ENCRYPT»

Виконав студент групи КІД-41
Маліков Дмитро Вікторович
Керівник кваліфікаційної роботи:
Торошанко Ярослав Іванович,
канд. техн. наук, доц., с.н.с

Київ – 2026

Мета, об’єкт, предмет дослідження

Мета роботи – спрощення та підвищення безпеки доступу до внутрішніх сервісів локальної мережі за рахунок використання Reverse Proxy, автоматизації HTTPS-з’єднань та контейнеризації мережевих служб

Об’єкт дослідження – процеси організації доступу до внутрішніх веб-сервісів у локальних мережах

Предмет дослідження – засоби та технології побудови, налаштування й оптимізації систем контролю доступу на базі Reverse Proxy, Let’s Encrypt, Dynamic DNS та Docker

Актуальність роботи

У роботі розглянуто теоретичні основи побудови систем централізованого доступу до внутрішніх ресурсів, принципи роботи Reverse Proxy, особливості маршрутизації HTTP/HTTPS-запитів, використання доменних імен у закритих середовищах, сервіси Dynamic DNS, механізми шифрування TLS та принципи роботи центру сертифікації Let's Encrypt. Також виконано порівняльний аналіз традиційного прокидання портів і сучасних Reverse Proxy-рішень з позиції безпеки, масштабованості та зручності адміністрування.

Особливу увагу приділено практичній реалізації системи доступу із використанням Docker Compose, Nginx Proxy Manager та DuckDNS, автоматичній генерації SSL-сертифікатів через DNS Challenge, маршрутизації запитів до внутрішніх сервісів і тестуванню HTTPS-доступу. Додатково розглянуто методи оптимізації роботи системи, перспективи впровадження автентифікації та перенесення інфраструктури на енергоефективний мікросервер.

Архітектура розробленої системи

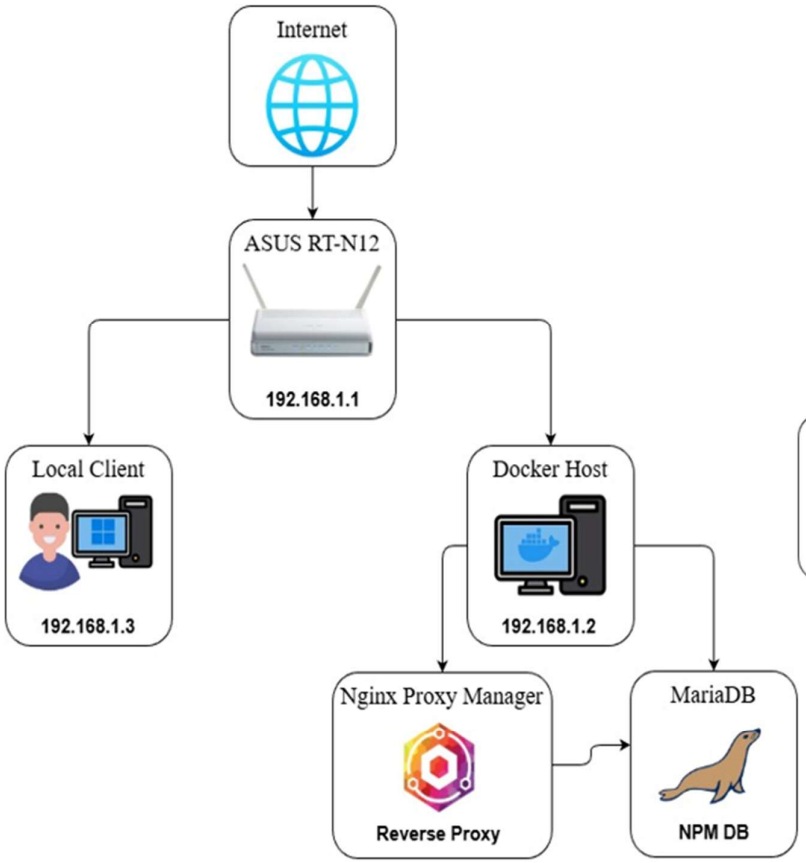


Рис. 2.1 – Архітектура системи доступу на базі Docker та NPM

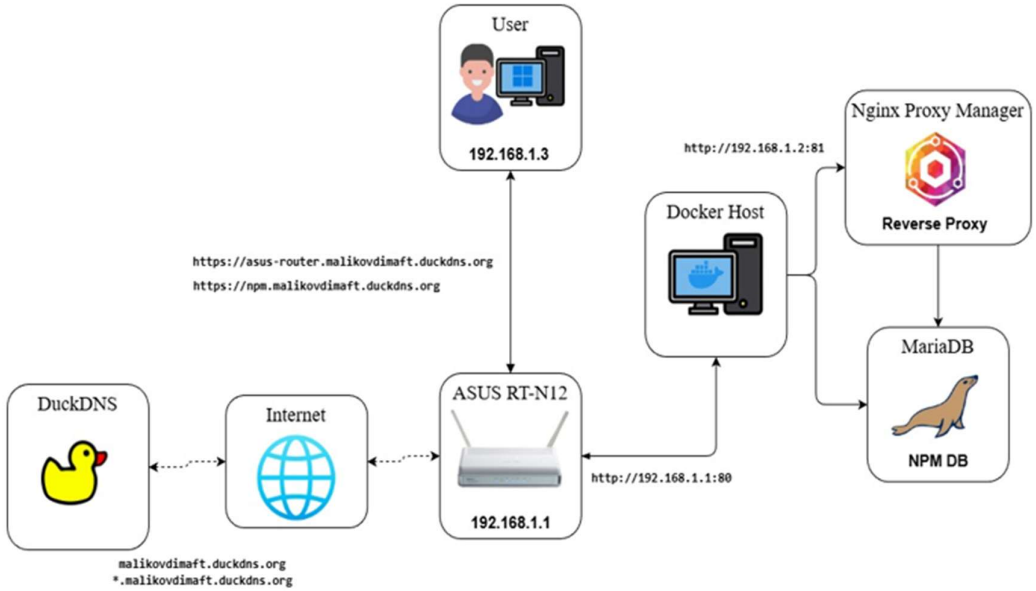


Рис. 2.2 – Взаємодія клієнта з внутрішніми сервісами через доменне ім'я

Використані компоненти

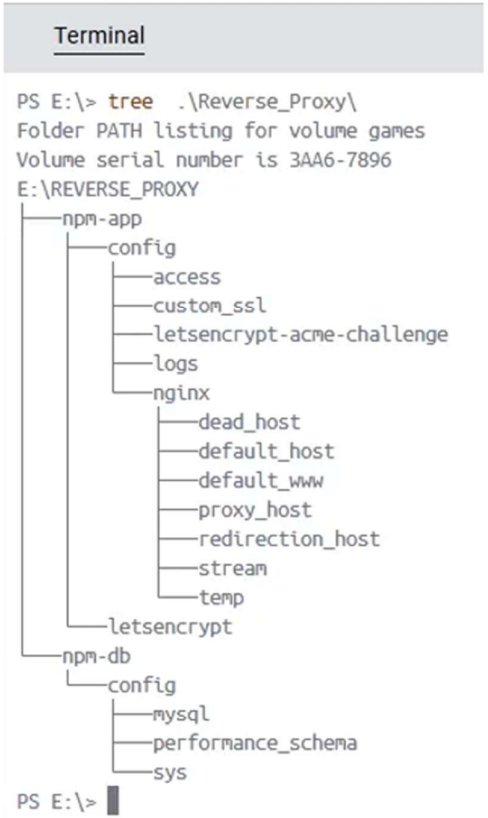


Рис. 2.4 – Структура в межах тестового середовища

Табл. 2.1 – Програмні компоненти розробленої системи та їх призначення

Компонент	Призначення
Docker Desktop	Середовище контейнеризації для розгортання сервісів
Nginx Proxy Manager	Reverse Proxy з веб-інтерфейсом для маршрутизації та SSL
MariaDB	База даних для зберігання конфігурацій NPM
DuckDNS	Dynamic DNS для створення доменних імен
Let's Encrypt	Генерація безкоштовних SSL/TLS сертифікатів
Router (Asus RT-N12)	Цільовий внутрішній ресурс (веб-інтерфейс)
Browser Client (Google Chrome)	Клієнт для доступу до сервісів через HTTPS

Практична реалізація системи

docker-compose.yml

E: > Reverse_Proxy > docker-compose.yml > { } services

```yaml
services:
  nginx-proxy-manager-app:
    image: jc21/nginx-proxy-manager:latest
    container_name: nginx-proxy-manager-app
    environment:
      - DB_MYSQL_HOST=${NPM_DB_MYSQL_HOST}                # DB_MYSQL_HOST: nginx-proxy-manager-db
      - DB_MYSQL_PORT=${NPM_DB_MYSQL_PORT}                # DB_MYSQL_PORT: 3306
      - DB_MYSQL_USER=${NPM_DB_MYSQL_USER}                # DB_MYSQL_USER: npm
      - DB_MYSQL_PASSWORD=${NPM_DB_MYSQL_PASSWORD}        # DB_MYSQL_PASSWORD: changeme
      - DB_MYSQL_NAME=${NPM_DB_MYSQL_NAME}                # DB_MYSQL_NAME: npm
    volumes:
      - ./npm-app/config:/data
      - ./npm-app/letsencrypt:/etc/letsencrypt
    depends_on:
      - nginx-proxy-manager-db
    networks:
      - reverse-proxy
    ports:
      - 80:80
      - 443:443
      - 81:81
    restart: always

  nginx-proxy-manager-db:
    image: jc21/mariadb-aria:latest
    container_name: nginx-proxy-manager-db
    environment:
      - MYSQL_ROOT_PASSWORD=${NPM_DB_MYSQL_ROOT_PASSWORD}    # MYSQL_ROOT_PASSWORD: changeme
      - MYSQL_DATABASE=${NPM_DB_MYSQL_NAME}                  # MYSQL_DATABASE: npm
      - MYSQL_USER=${NPM_DB_MYSQL_USER}                      # MYSQL_USER: npm
      - MYSQL_PASSWORD=${NPM_DB_MYSQL_PASSWORD}              # MYSQL_PASSWORD: changeme
      - MARIADB_AUTO_UPGRADE=1
    volumes:
      - ./npm-db/config:/var/lib/mysql
    networks:
      - reverse-proxy
    restart: always

networks:
  reverse-proxy:
    name: reverse-proxy
    driver: bridge
```

Рис. 2.5 – Фрагмент файлу docker-compose.yml

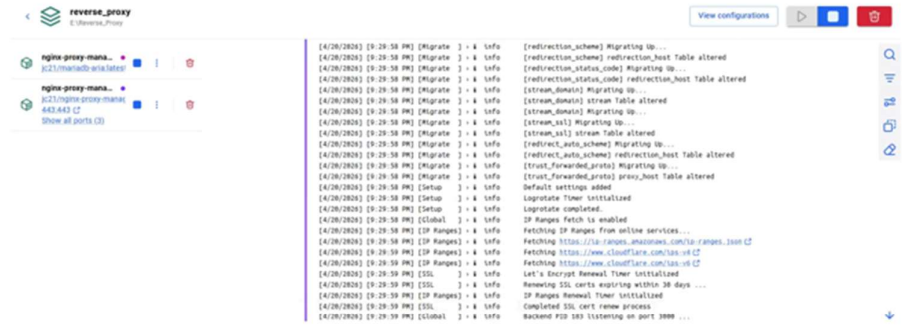


Рис. 2.11 – Список активних контейнерів після запуску

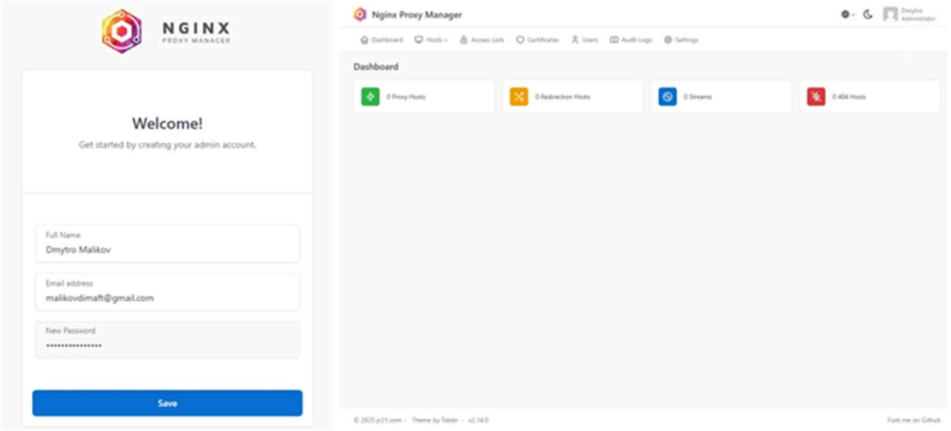


Рис. 2.12 – Панель керування Nginx Proxy Manager

Налаштування домену та SSL



Рис. 2.7 – Реєстрація домену у DuckDNS



Рис. 2.8 – Отримання токена DuckDNS API

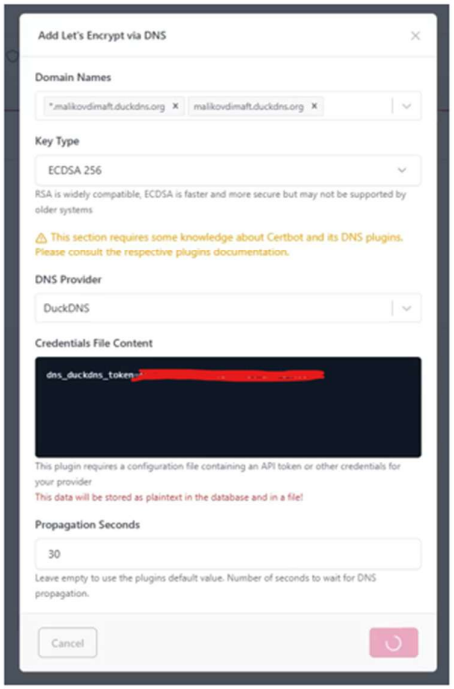


Рис. 2.13 – Генерація SSL-сертифіката Let's Encrypt

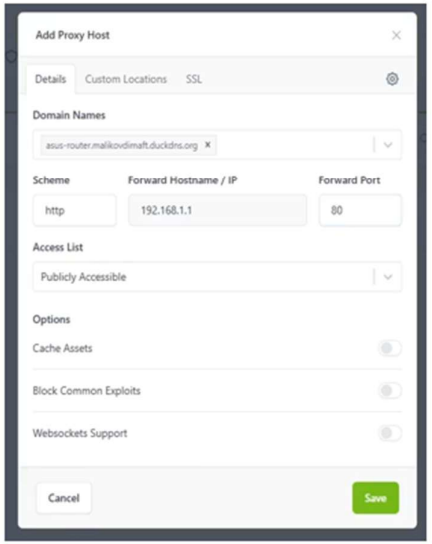


Рис. 2.14 – Створення Proxy Host у веб-інтерфейсі

Практичне тестування

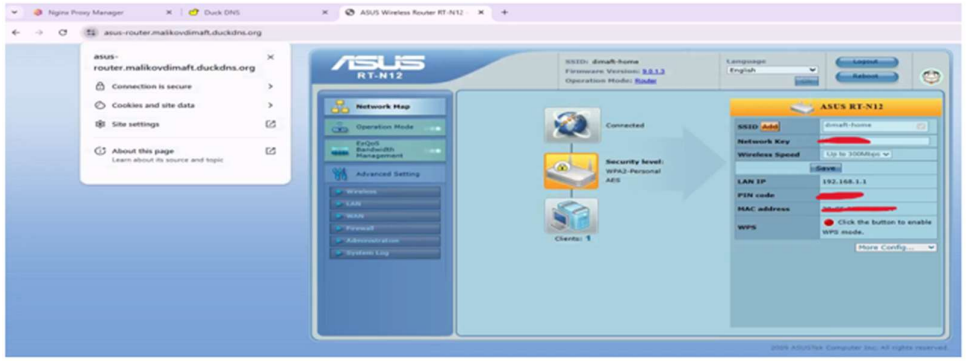


Рис. 2.19 – Відкриття внутрішнього сервісу через HTTPS-домен

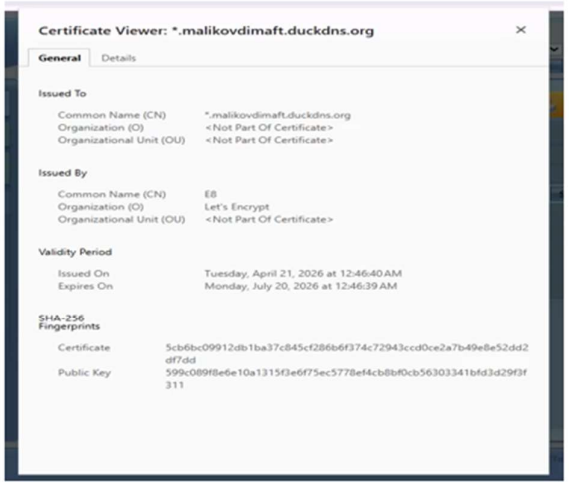


Рис. 2.20 – Валідний SSL-сертифікат у браузері

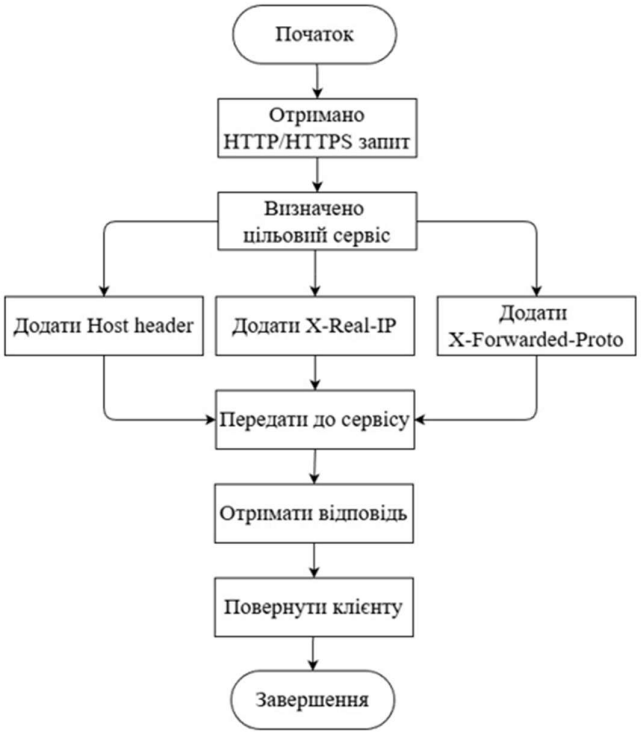


Рис. 2.18 – Алгоритм підміни службових заголовків

Практичне тестування

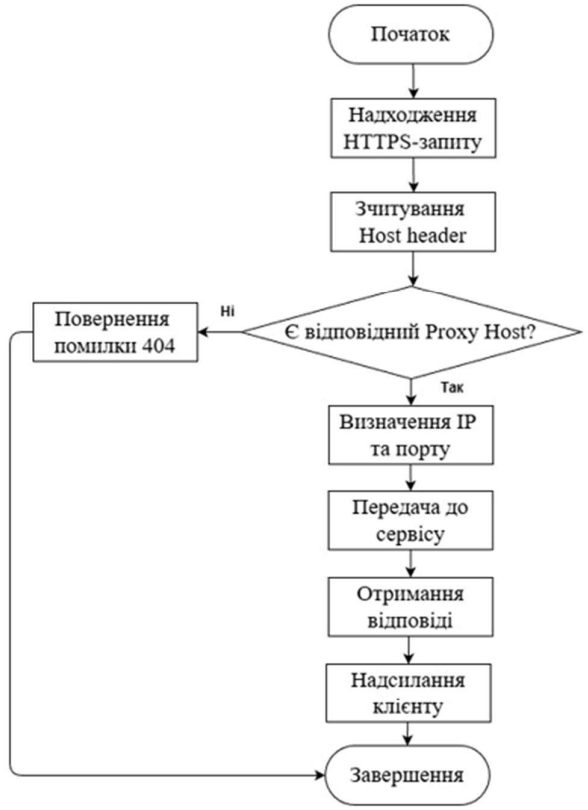


Рис. 2.17 – Алгоритм маршрутизації вхідного запиту

```
Terminal

PS E:\Reverse_Proxy> ping asus-router.malikovdimaft.duckdns.org

Pinging asus-router.malikovdimaft.duckdns.org [192.168.1.2] with 32 bytes of data:
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
PS E:\Reverse_Proxy>
```

Рис. 2.21 – Результат перевірки DNS-імені командою ping

```
Terminal

PS E:\Reverse_Proxy> ping asus-router.malikovdimaft.duckdns.org

Pinging asus-router.malikovdimaft.duckdns.org [192.168.1.2] with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
PS E:\Reverse_Proxy>
```

```
Terminal

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2a00:1e98:f2ab:491:d306:be0f:da62:9e4c
   Temporary IPv6 Address. . . . . . : 2a00:1e98:f2ab:491:ddc1:359f:3457:bf0
   Link-local IPv6 Address . . . . . : fe80::eded:f8e8:4c5a:312d%23
   IPv4 Address. . . . . . . . . . . : 10.139.96.54
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::d3:31ff:fe65:3683%23
                                       10.139.96.88
```

Рис. 2.22 – Демонстрація ізоляції локального сервісу від зовнішньої мережі

Порівняння з альтернативними рішеннями

Табл. 3.1 – Порівняння Reverse Proxy-рішень Nginx Proxy Manager, Traefik, Caddy

Критерій	Nginx Proxy Manager	Traefik	Caddy
GUI	Є	Обмежений	Відсутній
Автоматичний SSL	Так	Так	Так
Складність налаштування	Низька	Висока	Низька
Docker integration	Добра	Відмінна	Добра
Масштабованість	Середня	Висока	Середня
Зручність для початківця	Висока	Низька	Висока

Оптимізація продуктивності

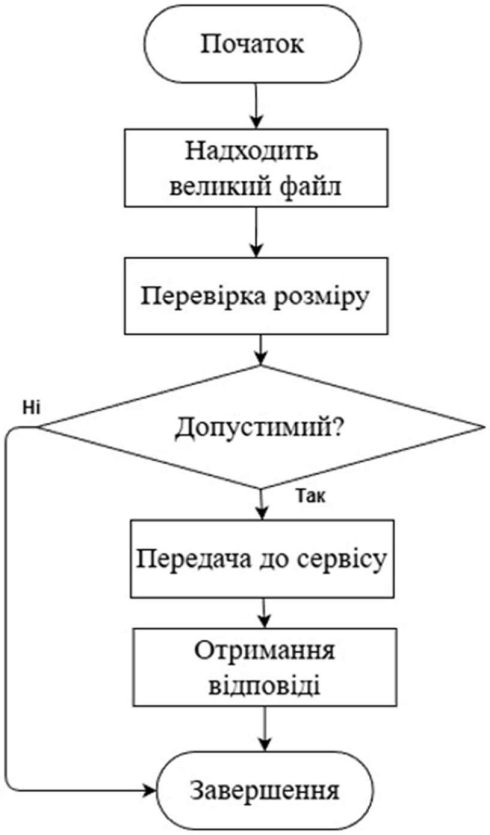


Рис. 3.1 – Вирішення таймаутів для стабільності з'єднання

Terminal

```
CONTAINER ID   NAME                        CPU %     MEM USAGE / LIMIT     MEM %     NET I/O         BLOCK I/O         PIDS
eb2ae55b3bbd   nginx-proxy-manager-app     0.11%     77.17MiB / 15.33GiB   0.49%     2.9MB / 77.2kB  0B / 422kB        39
bf496e166f23   nginx-proxy-manager-db      47.57%    66.98MiB / 15.33GiB   0.43%     579MB / 12MB    2.47MB / 4.14MB   7
```

Рис. 3.2 – Використання ресурсів контейнером до оптимізації



Рис. 3.4 – Алгоритм кешування статичного контенту

Terminal

```
CONTAINER ID   NAME                        CPU %     MEM USAGE / LIMIT     MEM %     NET I/O         BLOCK I/O         PIDS
eb2ae55b3bbd   nginx-proxy-manager-app     0.13%     80.38MiB / 15.33GiB   0.51%     2.9MB / 77.2kB  5.72MB / 422kB    40
bf496e166f23   nginx-proxy-manager-db      3.22%     72.22MiB / 15.33GiB   0.46%     799MB / 16.1MB  2.47MB / 702MB    7
```

Рис. 3.3 – Використання ресурсів контейнером після оптимізації

Напрямки модернізації

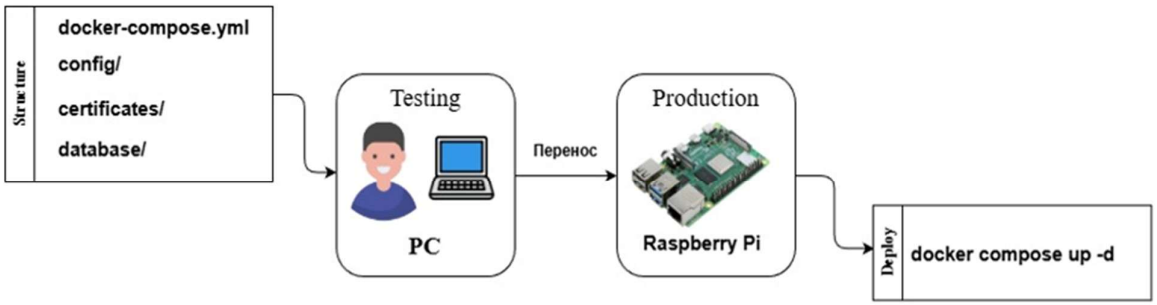


Рис. 3.7 – Міграція конфігурації на Raspberry Pi

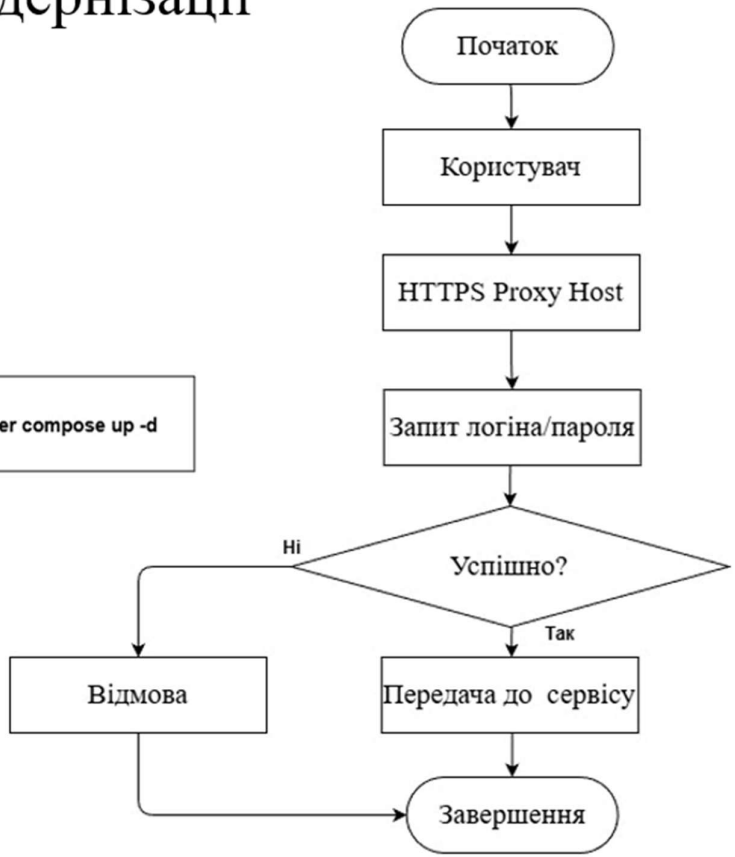


Рис. 3.6 – Схема додавання Basic Auth на рівні проксі

Висновки

У практичній частині було спроєктовано та реалізовано систему захищеного доступу до внутрішніх сервісів локальної мережі на базі Docker, Nginx Proxy Manager, DuckDNS і Let’s Encrypt. Побудована архітектура забезпечує централізований доступ до веб-ресурсів через доменні імена без використання внутрішніх IP-адрес і портів, а застосування DNS Challenge дозволило отримати валідні SSL-сертифікати та організувати захищене HTTPS-з’єднання навіть у локальному середовищі.

Проведене тестування підтвердило коректність маршрутизації запитів, стабільну роботу проксі-механізму та ізоляцію внутрішніх сервісів від зовнішніх мереж. Аналіз показав, що запропоноване рішення має мінімальну вартість впровадження завдяки використанню Open Source компонентів, а оптимізація ресурсів і кешування дозволяє ефективно використовувати систему навіть на звичайному обладнанні з можливістю подальшого масштабування.

Результати впровадження

1.Маліков Д.В., Торошанко Я.І. Система контролю доступу до внутрішніх сервісів на базі Reverse Proxy та безкоштовних DNS-сервісів. «Сучасний стан та перспективи розвитку IoT» : VII Науково-техн. конф., м. Київ, 20 квітня 2026 р. С.101-103.

2.Маліков Д.В., Торошанко Я.І. Контейнеризована інфраструктура безпечного доступу до внутрішніх сервісів на базі Docker, Reverse Proxy та Let’s Encrypt. «Застосування програмного забезпечення в ІКТ» : VII Всеукр. науково-техн. конф., м. Київ, 23 квітня 2026 р. (подано до публікації).