ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: **«Автоматизація міграції доменів за допомогою інструментів Python»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та технології
спеціальності 126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мз_2023[1]
Кузьменко Олег Миколайович
Керівник: Флегантов Леонід Олексійович
Рецензент: Ковальчук Станіслав Богданович

Полтава – 2024 року

ВСТУП

Актуальність теми даної роботи полягає в необхідності забезпечення ефективності, надійності та безперебійної роботи корпоративних мереж у процесі їх зростання та модернізації. У сучасних умовах, коли корпоративні мережі стають дедалі складнішими та масштабнішими, міграція доменів без автоматизації супроводжується значними витратами ресурсів та ризиком виникнення помилок, що можуть вплинути на доступність мережевих послуг і цілісність даних.

Основні проблєми, що виникають при традиційній міграції доменів, включають затримки в роботі мережі (ручні процеси міграції потребують багато часу та ресурсів, що може призвести до зупинки або уповільнення доступу до критично важливих мережевих ресурсів), високий ризик людських помилок (під час ручної міграції можливі помилки у конфігурації доменів, що може спричинити збої у доступі до даних, втрату інформації або проблєми з безпекою), складність у забезпеченні контролю за процесом (відсутність автоматизованих рішень ускладнює моніторинг та контроль міграційних дій, що підвищує ризики неправильного налаштування та недотримання стандартів безпеки).

Необхідність автоматизації міграції доменів обґрунтовується також її перевагами: оптимізація ресурсів – автоматизація дозволяє зменшити потребу в ручній роботі, що знижує ризик помилок і скорочує час міграції, вивільняючи ресурси для інших важливих завдань; підвищення надійності та швидкості – використання автоматизованих скриптів забезпечує швидке виконання завдань без перерв у роботі мережі, що підвищує загальну надійність процесу; полегшення моніторингу та контролю – автоматизовані рішення дозволяють здійснювати постійний моніторинг стану міграції, вчасно фіксувати проблєми та відстежувати історію змін для забезпечення відповідності стандартам.

Отже, тема роботи є актуальною, оскільки впровадження автоматизації у процес міграції доменів здатне суттєво покращити ефективність, надійність та

безпечність корпоративних мереж, що є важливими факторами стабільного функціонування сучасних організацій.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконана у відповідності до науково-дослідної ініціативної теми «Організаційно-методологічні аспекти впровадження інформаційно-комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» ДРН 0123U105060.

Мета роботи: розробити ефективний підхід до автоматизації міграції доменів у корпоративних мережах.

Завдання роботи: аналіз існуючих інструментів автоматизації міграції доменів, розробка Python-скриптів для автоматизації міграції доменів, оцінка ефективності запропонованих рішень, формування пропозицій.

Об'єкт дослідження: процес міграції доменів у корпоративних мережах.

Предмет дослідження: автоматизація міграції доменів за допомогою інструментів Python.

Методи дослідження:

- огляд наукових та технічних джерел для вивчення існуючих підходів до автоматизації міграції доменів і аналізу можливостей Python для реалізації таких завдань;

- аналіз можливостей Python для автоматизації процесу міграції доменів;

- порівняльний аналіз Python-інструментів з іншими рішеннями автоматизації;

- проведення практичних експериментів шляхом розробки та тестування Python-скриптів для автоматизованої міграції доменів у тестовому середовищі;

- аналіз експериментальних результатів досліджень процесу автоматизованої міграції;

- синтез висновків та рекомендацій щодо використання Python-інструментів для автоматизації процесу міграції доменів у корпоративних мережах.

Інформаційна база дослідження: науково-технічна література – книги та підручники з автоматизації процесів, мережевих технологій та інформаційних

систем; наукові статті та публікації з тематики автоматизації міграції доменів, доступні в наукових базах даних Google Scholar, IEEE Xplore, SpringerLink, ScienceDirect, Scopus, ResearchGate; вітчизняні та міжнародні стандарти та методології з управління мережею та автоматизації процесів; офіційна документація та ресурси щодо Python-бібліотек для автоматизації мережевих операцій, таких як DNSPython, Paramiko, Fabric; документація з мережевих протоколів та інструментів (DNS, SSH); інтернет-джерела аналітичного характеру, блоги, статті та технічні звіти експертів у галузі Python-програмування, мережевих технологій та автоматизації корпоративних процесів.

Елементи наукової новизни: проведений порівняльний аналіз Python-бібліотек для автоматизації міграції доменів з іншими засобами автоматизації (на прикладі Ansible) дозволив виявити переваги та обмеження Python-інструментів у контексті міграції доменів; дослідження ефективності автоматизації міграційних процесів за допомогою Python-скриптів надало можливість оцінити оптимальні підходи для підвищення надійності, швидкості та зниження ризиків помилок під час міграції.

Практична значущість: робота надає практичні рекомендації щодо використання Python для автоматизації процесу міграції доменів у корпоративних мережах; отримані результати дозволяють розробити рекомендації з вибору бібліотек та розробки скриптів для ефективної автоматизації міграції доменів, що сприяє зменшенню витрат на обслуговування мережі та підвищенню її надійності.

Апробація результатів дослідження. За результатами проведеного дослідження опубліковано тези доповіді: «Міграція доменів: основні етапи та методи», Матер. XXI щорічного міждисциплінарного семінару «Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій ННІ ЕУП та ІТ ПДАУ», 20 листопада 2024 року, м. Полтава (Додаток А).

Структура та обсяг кваліфікаційної роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Основний текст роботи викладений на 66 сторінках, містить 8 рисунків і 20 таблиць. Список використаних джерел налічує 82 найменувань.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ТА МЕТОДОЛОГІЧНІ АСПЕКТИ МІГРАЦІЇ ДОМЕНІВ

1.1 Етапи та методи міграції доменів

Міграція доменів – це процес перенесення доменних імен та пов'язаних з ними налаштувань з однієї інфраструктури або сервера на іншу платформу або середовище. Міграція полягає у перенесенні доменних імен та їх налаштувань, таких як DNS-записи, з однієї інфраструктури або сервера на іншу платформу або середовище, може включати також зміну провайдера хостингу, переміщення домену між різними мережами або перехід на нову архітектуру системи [1-6].

Основними етапами процесу міграції доменів є підготовка, перенесення даних, перевірка цілісності, активація домену. Кожен з цих етапів є важливим для забезпечення надійності міграції та безперебійної роботи домену в новому середовищі [7-10].

На етапі підготовки вирішуються наступні завдання:

- резервне копіювання всіх налаштувань і даних, щоб запобігти їх втраті під час міграції;
- оцінка поточного стану домену та всіх пов'язаних із ним DNS-записів;
- розробка плану міграції, включаючи перевірку сумісності нової платформи із системою домену.

Перенесення даних включає:

- перенесення DNS-записів, таких як A, CNAME, MX, TXT, для нового місцезнаходження домену;
- оновлення записів на новій платформі та налаштування необхідних параметрів.

Перевірка цілісності передбачає:

- тестування доступності домену в новому середовищі для перевірки коректності всіх налаштувань;

- використання інструментів моніторингу, щоб переконатися, що всі сервіси та дані працюють належним чином.

Активація домену передбачає:

- офіційне завершення перенесення домену, змінення відповідного TTL (Time to Live) для нових записів DNS.

- сповіщення користувачів про завершення міграції та тестування остаточного доступу до домену.

Кожен етап міграції доменів є важливим для забезпечення стабільності та доступності ресурсів у новому середовищі. Етап підготовки відіграє фундаментальну роль, оскільки без належного резервного копіювання всієї конфігурації та даних існує ризик втрати критичної інформації. Оцінка поточного стану домену та DNS-записів дозволяє визначити можливі проблєми чи обмеження, які можуть вплинути на міграцію. Це включає аналіз сумісності нової платформи, перевірку структур записів і підготовку плану дій, що допомагає мінімізувати простої та забезпечити послідовність процесу [11, 12].

Перенесення даних – це технічна реалізація змін, де особливу увагу приділяють коректному перенесенню ключових DNS-записів, таких як A, CNAME, MX, TXT. У цьому етапі важливо враховувати налаштування параметрів відповідно до нової інфраструктури, щоб забезпечити точність адресації та функціональність послуг. Злагодженість дій на цьому етапі допомагає уникнути розриву зв'язку з вебресурсами або електронною поштою, які залежать від домену.

Етап перевірки цілісності є обов'язковим для перевірки доступності домену після міграції. Тестування забезпечує впевненість у правильності перенесених налаштувань, а інструменти моніторингу гарантують, що всі сервіси працюють стабільно. Завершальний етап активації передбачає остаточне налаштування, зокрема оновлення TTL для прискорення поширення нових записів DNS. Сповіщення користувачів і додаткове тестування доступу дозволяють упевнитися, що всі зміни впроваджені без порушень, а система працює стабільно в новому середовищі.

Послідовність дій при виконанні основних етапів міграції доменів представлена на рисунку 1.1.

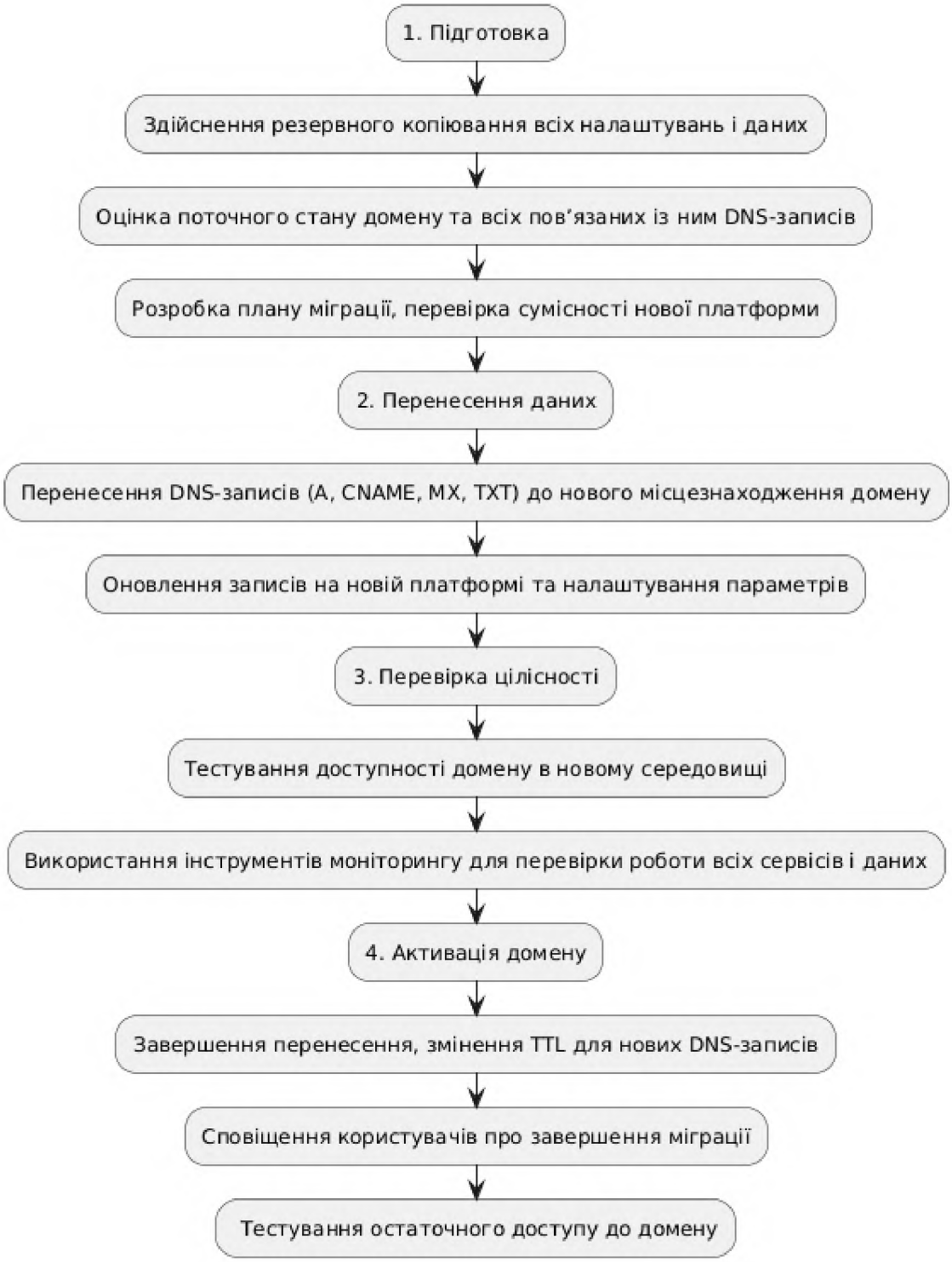

Рисунок 1.1 – Послідовність виконання основних етапів міграції домену

Існують різні методи і підходи до міграції доменів. Вони залежать від конкретних вимог, типу даних та масштабів мережі [13-15].

Одними з найбільш простих але й водночас найбільш трудомістких методів є ручні методи міграції доменів. Зазвичай, вони реалізуються за безпосередньої участі адміністратора IT-інфраструктури, який власноруч виконує всі налаштування, пов'язані з перенесенням DNS-записів, резервуванням даних, налаштуванням доменних серверів та іншими операціями. Внаслідок цього, ручні методи міграції доменів мають свої сильні та слабкі сторони (таблиця 1.1).

Таблиця 1.1 – Переваги та обмеження ручних методів міграції доменів

Сильні сторони ручних методів	Слабкі сторони ручних методів
Детальний контроль над процесом, можливість врахування всіх специфічних налаштувань домену.	Високий ризик помилок через людський фактор (наприклад, неправильне налаштування DNS-записів).
Можливість швидкого реагування на нестандартні ситуації або проблєми, що виникають під час міграції.	Затрати часу на виконання кожного етапу, особливо для великих мереж із численними доменами.
Гнучкість у налаштуванні параметрів та корекції процесу на різних етапах.	Відсутність можливості автоматизації, що обмежує масштабованість процесу.

На основі сказаного вище, можна зауважити, що ручні методи міграції доменів мають суттєві недоліки, такі як висока ймовірність помилок, тривалий час виконання та залежність від людського фактору.

В умовах великої кількості доменів або складної мережевої архітектури такі обмеження можуть призвести до простоїв, втрати даних чи порушення роботи сервісів.

Для подолання слабких сторін ручних методів природним рішенням є застосування автоматизованих методів міграції доменів, які дозволяють, зокрема, мінімізувати ризики притаманні складним процесам міграції доменів та скоротити час, необхідний для виконання кожного з етапів процесу міграції, що є особливо важливим в умовах великої кількості доменів або складної мережевої архітектури.

1.2 Інструменти автоматизованої міграції доменів

1.2.1 Консольні сценарії міграції доменів

До інструментів автоматизації міграції доменів належать застосування консольних сценаріїв міграції, розроблення скриптів автоматизації міграції на мові Python та використання спеціалізованих програмних засобів DevOps [3, 7, 9].

Сценарії міграції – це послідовність заздалегідь налаштованих кроків, які автоматично виконуються у певному порядку. Прості сценарії пишуть на мовах сценаріїв для виконання у консольних оболонках Bash або PowerShell, і використовують для автоматичного перенесення DNS-записів, зміни налаштувань сервера, оновлення конфігурацій тощо [16-18].

Сценарії Bash використовуються для автоматизації завдань адміністрування у середовищі Linux/Unix. Оболонка PowerShell використовується для автоматизації завдань адміністрування, налаштування системи та серверів у середовищі Windows.

Типовими застосуваннями Bash-сценаріїв є перевірка доступності вебсайту та резервне копіювання даних та налаштувань. Наприклад, наступний код виконує автоматизовану перевірку доступності вебсайту [19]:

```
#!/bin/bash
# Перевірка доступності вебсайту

URL="http://example.com"
if curl -s --head "$URL" | grep "200 OK" > /dev/null
then
  echo "Сайт доступний."
else
  echo "Сайт недоступний."
fi
```

Перший рядок цього коду вказує, що даний скрипт буде виконуватись у середовищі Bash:

```
#!/bin/bash
```

Другий рядок – це коментар, який пояснює призначення скрипту:

```
# Перевірка доступності вебсайту
```

Третій рядок визначає змінну URL, що містить адресу вебсайту, доступність якого потрібно перевірити:

```
URL="http://example.com"
```

Четвертий рядок містить основний функціонал скрипту:

```
if curl -s --head "$URL" | grep "200 OK" > /dev/null
```

Тут виконується команда `curl` з ключами:

`-s (silent)` – виключає відображення зайвої інформації;

`--head` – надсилає лише HTTP-заголовки (HEAD-запит).

Команда `curl`, яка використовується у скрипті, виконує HTTP-запит до вказаного вебсайту (URL) і повертає у відповідь HTTP-заголовки, що містять інформацію про статус відповіді сервера (наприклад, 200 OK, 404 Not Found тощо). Отриманий результат передається команді `grep "200 OK"`, яка шукає в ньому HTTP-статус «200 OK» (успішна відповідь).Тобто, команда `grep "200 OK"` отримує цей результат виконання `curl` (заголовки) і перевіряє, чи міститься в них рядок 200 OK. Якщо такий рядок буде знайдено, це означає, що сервер успішно відповів на запит, і сайт доступний. Далі, команда `> /dev/null` – перенаправляє стандартний вихідний потік у «порожнечу», щоб уникнути зайвого виводу у консоль.

Наступний рядок then відкриває блок команд, що виконуються, коли умова `if` є істинною. Тобто, якщо команда `grep "200 OK"` знаходить 200 OK, то скрипт виводить повідомлення, що сайт доступний: echo "Сайт доступний." В іншому випадку, else відкриває альтернативний блок, який виконується, якщо умова `if` є хибною: `echo "Сайт недоступний."` Тобто, якщо статус 200 OK не знайдено, виводиться повідомлення, що сайт недоступний.

Насамкінець, оператор fi закриває блок if.

Алгоритм роботи розглянутого скрипту можна подати наступною блок-схемою (рисунок 1.2).

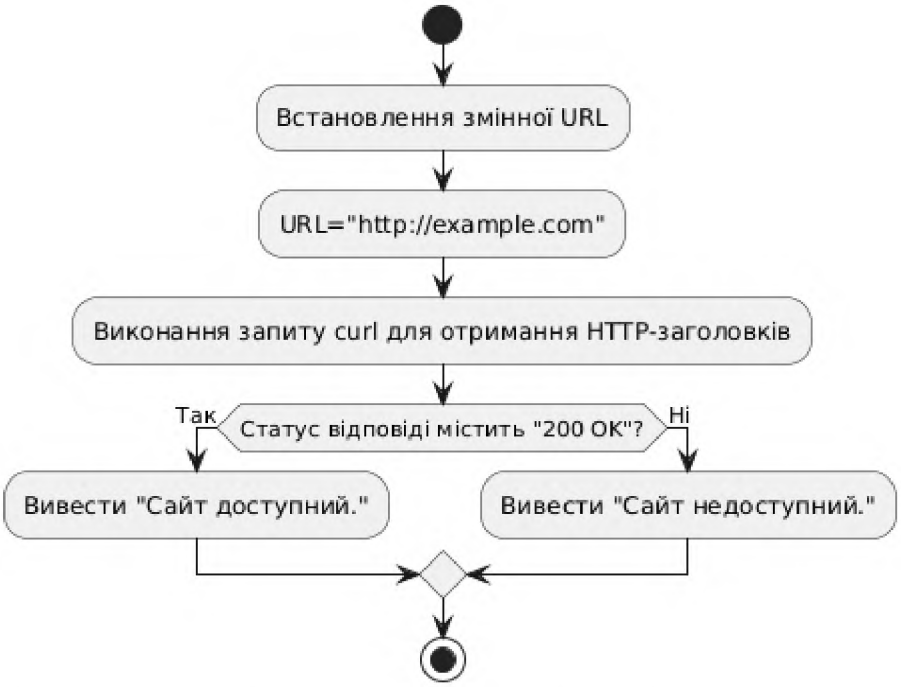

Рисунок 1.2 – Алгоритм роботи Bash-сценарію для автоматизації визначення доступності вебсайту

Запуск скриптів автоматизації у середовищі Bash здійснюється у такій послідовності [19]:

a) спочатку потрібно створити файл із кодом скрипту:
   - відкрити текстовий редактор (наприклад, nano);
   - створити файл, наприклад, check_website.sh:
     `nano check_website.sh`
   - вставити код скрипту в цей файл.

b) зберегти файл:
   - Ctrl + O, ввести назву файлу та натиснути Enter;
   - натиснути Ctrl + X, щоб вийти з редактора.

c) надати файлу права на виконання – виконати команду, що зробить цей файл виконуваним:

```
chmod +x check_website.sh
```

d) запустити скрипт на виконання у терміналі:

```
./check_website.sh
```

Після запуску скрипт виведе одне з двох повідомлень: «Сайт доступний» – якщо на запис скрипту вебсайт відповідає статусом 200 OK, або «Сайт недоступний» – якщо вебсайт не відповідає або доступ до нього обмежений.

Цей скрипт можна використати для перевірки доступності будь-якого іншого URL. Для цього потрібно змінити значення змінної URL у скрипті на інший вебсайт або передати його як параметр, таким чином:

```
URL="http://newwebsite.com" ./check_website.sh
```

Аналогічно, можна виконати й автоматизацію резервного копіювання файлів. Відповідний скрипт автоматизації має вид:

```
#!/bin/bash
# Резервне копіювання файлів у папку backup

SOURCE_DIR="/home/user/documents"
BACKUP_DIR="/home/user/backup"
DATE=$(date +%Y-%m-%d)

mkdir -p "$BACKUP_DIR"
cp -r "$SOURCE_DIR" "$BACKUP_DIR/backup_$DATE"
echo "Резервне копіювання завершено."
```

Типовими сценаріями PowerShell для Windows є також перевірка доступності серверів та резервне копіювання файлів. Скрипт PowerShell для перевірки доступності заданого URL, який виконує дії, повністю аналогічні розглянутому вище Bash-сценарію, має вид [18]:

```
# Перевірка доступності вебсайту

# Встановлення URL
$URL = "http://example.com"

# Виконання запиту до вебсайту
$response = Invoke-WebRequest -Uri $URL -Method Head -
ErrorAction SilentlyContinue

# Перевірка статусу відповіді
if ($response.StatusCode -eq 200) {
    Write-Output "Сайт доступний."
} else {
    Write-Output "Сайт недоступний."
}
```

Тут змінна `$URL` зберігає адресу вебсайту, доступність якого потрібно перевірити:

```
$URL = "http://example.com"
```

В основній частині, директива `Invoke-WebRequest` виконує HTTP-запит до вказаного URL з такими параметрами:

`-Uri $URL` – адреса для запиту;

`-Method Head` – надсилає тільки заголовки HTTP-запиту, аналогічно `curl -head`;

`-ErrorAction SilentlyContinue` – приховує помилки, якщо сервер не відповідає.

Метод `$response.StatusCode` використовується у коді скрипту для отримання статусу відповіді сервера: наприклад, статус 200 означає, що сайт доступний. В умові `if`, якщо буде отримано статус 200, виводиться «Сайт доступний», інакше виводиться «Сайт недоступний».

Щоб виконати цей скрипт потрібно:

a) відкрити оболонку PowerShell;

b) створити файл скрипту, наприклад, CheckWebsite.ps1;

c) вставити код скрипту у файл;

d) запустити скрипт у PowerShell:

```
.\CheckWebsite.ps1
```

Цей скрипт працює аналогічно Bash-скрипту, але використовує можливості PowerShell для роботи з вебзапитами.

Прикладом дещо іншого підходу у реалізації цього завдання у PowerShell є перевірка доступності сервера за допомогою команди `Test-Connection`, реалізованої у наступному коді:

```
# Перевірка доступності сервера за допомогою команди
Test-Connection

$server = "example.com"
```

```
if (Test-Connection -ComputerName $server -Count 2 -
Quiet) {
    Write-Output "Сервер доступний."
} else {
    Write-Output "Сервер недоступний."
}
```

У PowerShell, як і у Bash, можна достатньо легко автоматизувати резервне копіювання файлів [20]:

```
# Резервне копіювання файлів у папку Backup

$sourceDir = "C:\Users\user\Documents"
$backupDir = "C:\Users\user\Backup"
$date = Get-Date -Format "yyyy-MM-dd"

if (!(Test-Path -Path $backupDir)) {
    New-Item -ItemType Directory -Path $backupDir
}

Copy-Item -Path $sourceDir -Destination
"$backupDir\Backup_$date" -Recurse
Write-Output "Резервне копіювання завершено."
```

Логіка роботи цього скрипту є зрозумілою без додаткових пояснень.

1.2.2 Використання Python для управління доменами

Python є потужним інструментом автоматизації рутинних завдань, особливо у мережевих операціях. Типовими застосуваннями Python для автоматизації мережевих операцій є налаштування DNS-записів, виконання SSH-з'єднань та команд на віддаленому сервері, інтеграція додаткових налаштувань [21-25].

Основними інструментами Python для автоматизації управління доменами є три спеціалізовані бібліотеки DNSPython, Paramiko і Fabric.

DNSPython – це бібліотека Python для роботи з DNS-записами, яка дозволяє читати, створювати та змінювати DNS-записи. Типовим є наступний приклад використання бібліотеки DNSPython для зміни DNS-запису:

```
import dns.resolver
result = dns.resolver.resolve('example.com', 'A')
for ip in result:
    print('IP Address:', ip.to_text())
```

В цьому коді використовується dns.resolver – модуль для отримання інформації про DNS-записи, що входить до складу бібліотеки DNSPython.

Paramiko – бібліотека для роботи з SSH-з'єднаннями. З її допомогою можна автоматично підключатися до віддалених серверів і виконувати на стороні сервера різні команди, що необхідно для налаштування доменів або перенесення даних. Наступний код демонструє використання бібліотеки Paramiko для підключення до віддаленого сервера:

```
import paramiko
client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPol
icy())
client.connect('hostname', username='user',
password='password')
stdin, stdout, stderr = client.exec_command('ls -l')
print(stdout.read().decode())
client.close()
```

Бібліотека Fabric – це потужний інструмент для автоматизації завдань та налаштувань через SSH-з'єднання. Fabric дозволяє виконувати команду на віддалених серверах та автоматизувати комплексні сценарії. Наступний приклад ілюструє використання бібліотеки Fabric для виконання віддаленої команди:

```
from fabric import Connection
c = Connection('user@hostname')
c.run('uname -s')
```

Ще одним практичним прикладом застосування бібліотеки DNSPython є наступний скрипт для автоматичного налаштування DNS-записів:

```
import dns.resolver
import dns.update
import dns.query

# Параметри
domain = "example.com"
dns_server = "192.168.1.1"

# Налаштування нового A-запису
update = dns.update.Update(domain)
update.replace("www", 300, "A", "192.168.1.100")

# Відправка оновлення на DNS-сервер
```

```
dns.query.tcp(update, dns_server)
print("DNS-запис оновлено.")
```

В цьому коді, крім dns.resolver, використовуються також dns.update – модуль для створення оновлень DNS-записів, та dns.query – модуль для надсилання запитів до DNS-серверів, які також входять до складу бібліотеки DNSPython, яка використовується для роботи з DNS-записами.

Використання бібліотеки paramiko для виконанння SSH-з'єднання та команди на віддаленому сервері демонструє наступний скрипт:

```
import paramiko

# Параметри підключення
hostname = "example.com"
username = "user"
password = "password"

# Підключення до сервера через SSH
client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPol
icy())
client.connect(hostname, username=username,
password=password)

# Виконання команди
stdin, stdout, stderr = client.exec_command("ls -l
/home/user")
print(stdout.read().decode())

# Закриття з'єднання
client.close()
print("Команда виконана на віддаленому сервері.")
```

Серед завдань автоматизації, які вирішуються з використанням Python-бібліотек DNSPython, Paramiko і Fabric, типовим завданням є інтеграція додаткових налаштувань, зокрема, створення резервної копії файлів перед зміною конфігурації, що може бути реалізована у такий спосіб:

```
import shutil
import os

# Параметри
config_path = "/etc/myapp/config.yaml"
```

```
backup_path = "/etc/myapp/backup/config_backup.yaml"

# Створення резервної копії файлу
if os.path.exists(config_path):
    shutil.copy(config_path, backup_path)
    print("Резервну копію створено.")

# Виконання змін у конфігураційному файлі (приклад)
with open(config_path, "a") as file:
    file.write("\nnew_setting: true")
    print("Конфігурація оновлена.")
```

На початку цього коду виконується імпорт бібліотек shutil (для роботи з файлами та їх копіюванням) та os (для взаємодії з файловою системою, перевірки наявності файлів тощо):

```
import shutil
import os
```

Наступним кроком встановлюються параметри config_path (шлях до файлу конфігурації, який буде оновлюватись) та backup_path (шлях, куди буде збережено резервну копію конфігураційного файлу):

```
config_path = "/etc/myapp/config.yaml"
backup_path = "/etc/myapp/backup/config_backup.yaml"
```

Далі відбувається створення резервної копії налаштувань:

```
if os.path.exists(config_path):
    shutil.copy(config_path, backup_path)
    print("Резервну копію створено.")
```

Метод os.path.exists(config_path) перевіряє, чи існує файл за вказаним шляхом, shutil.copy(config_path, backup_path) копіює файл конфігурації до резервного місця. Якщо файл конфігурації існує, створюється резервна копія та виводиться повідомлення.

Насамкінець, відбувається оновлення файлу конфігурації:

```
with open(config_path, "a") as file:
    file.write("\nnew_setting: true")
    print("Конфігурація оновлена.")
```

Метод open(config_path, "a") відкриває файл у режимі дописування (a – append), file.write("\nnew_setting: true") додає новий параметр new_setting: true до файлу, виводиться повідомлення про оновлення конфігурації.

Хоча розглянутий код резервного копіювання та оновлення файлу напряму не використовує DNSPython, Paramiko або Fabric, він демонструє основи автоматизації конфігурацій, які можуть бути інтегровані з цими бібліотеками. Зокрема, DNSPython може використовувати цей код для автоматичного оновлення DNS-записів після внесення змін у конфігураційний файл. Наприклад, з DNSPython після оновлення конфігураційного файлу (редагування конфігурації) можна автоматично оновити A-записи для DNS:

```
import dns.update
import dns.query

domain = "example.com"
update = dns.update.Update(domain)
update.replace("www", 300, "A", "192.168.1.100")
dns.query.tcp(update, "192.168.1.1")
```

Бібліотека Paramiko використовується для віддаленого управління серверами через SSH. Отже, код резервного копіювання та оновлення файлу може бути виконаний на віддаленому сервері через SSH-з'єднання, ініційоване Paramiko. Наприклад, можна автоматично скопіювати файл конфігурації з локальної машини на віддалену та оновити файл конфігурації на сервері, виконавши цей код на віддаленому сервері через SSH:

```
import paramiko

ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy
())
ssh.connect("remote_server_ip", username="user",
password="password")
ssh.exec_command("python /path/to/script.py")
ssh.close()
```

Бібліотека Fabric застосовується для автоматизації виконання завдань одночасно на кількох серверах. Тому, код резервного копіювання та оновлення файлу може бути частиною Fabric-скрипта, який виконується одночасно на

багатьох серверах. Fabric може автоматично передавати змінені файли конфігурації або виконувати скрипт на одночасно на кількох віддалених серверах:

```
from fabric import Connection
servers = ["server1", "server2", "server3"]
for server in servers:
    conn = Connection(host=server, user="user")
    conn.put("config.yaml", "/etc/myapp/config.yaml")
    conn.run("python /path/to/script.py")
```

Таким чином, наведений вище код резервного копіювання та оновлення файлу забезпечує базову автоматизацію резервного копіювання та редагування файлу конфігурації. У поєднанні з DNSPython, Paramiko та Fabric цей процес можна розширити до масштабного управління мережевими серверами, конфігураціями та DNS-записами.

1.2.3 Інструменти DevOps

Для найбільш складних операцій з управління доменами використовуються спеціалізовані інструменти DevOps, такі як Ansible, Chef або Puppet, які дозволяють керувати великими середовищами та синхронізувати зміни конфігурації у масштабах всієї мережі [26-30].

Типовим прикладом використання програмного засобу Ansible є автоматизоване налаштування вебсервера Apache на декількох серверах. Ansible використовує спеціальні YAML-файли (playbooks) для автоматизації завдань. Прикладом може слугувати наступний playbook для автоматизованого встановлення вебсерверу Apache:

```
- name: Встановлення та запуск Apache на серверах
  hosts: webservers
  become: yes
  tasks:
    - name: Встановити Apache
      apt:
        name: apache2
        state: present
    - name: Запустити та увімкнути Apache
      service:
        name: apache2
        state: started
        enabled: true
```

Цей playbook встановлює вебсервер Apache та запускає його на всіх серверах у групі webservers. Запуск процесу виконується через термінал за допомогою команди:

```
ansible-playbook -i inventory.ini apache_setup.yml
```

Типове застосування програмного засобу Chef – це автоматизоване управління конфігурацією баз даних. Для конфігурації серверів баз даних Chef використовує спеціальні файли, так звані «рецепти» (recipes). Прикладом рецепту Chef є скрипт, що автоматично встановлює сервер баз даних MySQL, запускає його та налаштовує базову конфігурацію на сервері (мовою ruby):

```
package 'mysql-server' do
  action :install
end

service 'mysql' do
  action [:enable, :start]
end

execute 'mysql_secure_installation' do
  command 'mysql_secure_installation'
  action :run
end
```

Рецепт Chef – це набір інструкцій для налаштування певної програми або служби на сервері. Для виконання більш складних завдань, рецепти Chef об'єднуються у ролі (role) – логічні групи рецептів або налаштувань. Ролі у Chef визначають конфігурацію певного типу серверів. Нижче подано приклад використання рецепту mysql_setup у ролі database_server:

```
name "database_server"
description "Роль для налаштування серверів баз даних"
run_list "recipe[mysql_setup]"
```

Типовим застосуванням Puppet є автоматизоване налаштування конфігурації вебсерверів. Для опису конфігурації вебсерверів Puppet використовує спеціальні фали, так звані «маніфести» (manifests). У наступному прикладі подано маніфест Puppet, що автоматично встановлює вебсервер Nginx, забезпечує його запуск та оновлює конфігураційний файл за допомогою шаблону:

```
class nginx_setup {
  package { 'nginx':
    ensure => installed,
  }

  service { 'nginx':
    ensure => running,
    enable => true,
    require => Package['nginx'],
  }

  file { '/etc/nginx/nginx.conf':
    ensure  => file,
    mode    => '0644',
    content => template('nginx/nginx.conf.erb'),
    require => Package['nginx'],
    notify  => Service['nginx'],
  }
}

include nginx_setup
```

Виконання маніфесту Puppet здійснюється командою:

```
puppet apply nginx_setup.pp
```

Переваги та обмеження розглянутих методів автоматизації міграції доменів узагальнені у таблиці 1.2.

Таблиця 1.2 – Переваги та обмеження основних методів автоматизації міграції доменів

Метод	Опис	Переваги	Обмеження
Сценарії (Bash, PowerShell)	Просте налаштування, мінімум залежностей	Швидке налаштування, зручність для простих задач	Не підходить для великих мереж
Скрипти Python	Гнучкий інструмент для комплексних налаштувань	Широкі можливості, зручність для мережевих операцій	Вимагає знань Python
Інструменти DevOps	Комплексне управління конфігураціями в масштабах мережі	Висока масштабованість, централізоване управління	Вимагає налаштування інфраструктури

Виходячи зі сказаного вище, можна зауважити, що розглянуті інструменти DevOps дозволяють автоматизувати та стандартизувати зміни у конфігураціях на

різних серверах, полегшуючи керування великими середовищами. При цьому, Ansible добре підходить для простих одноразових завдань автоматизації, Chef підходить для складних завдань з управління конфігураціями, особливо в середовищах з великою кількістю серверів, Puppet найкраще використовувати для постійного контролю стану конфігурацій та автоматичного виправлення помилок у великих інфраструктурах [31-35].

Отже, основні методи автоматизації міграції доменів мають свої переваги та обмеження. Сценарії (Bash, PowerShell) забезпечують швидке налаштування для простих завдань, але не дуже добре підходять для великих мереж. Скрипти Python забезпечують гнучкість і широкі можливості, але вимагають специфічних знань цієї мови програмування. Інструменти DevOps характеризуються високою масштабованістю у застосуванні, але є доволі складними у налаштуванні. Таким чином, використання скриптів Python для автоматизації міграції доменів виявляється рішенням, яке задовольняє переважну більшість потреб.

1.3 Інтеграція Python-скриптів у корпоративне середовище

Інтеграція систем автоматизації, таких як скрипти Python, у корпоративне середовище є корисною для забезпечення ефективної міграції доменів і зниження обсягу ручної роботи. Інтеграція скриптів Python вимагає дотримання певних правил та принципів, які забезпечують стабільність, безпеку та масштабованість процесу міграції, а саме: модульність і повторне використання коду, безпека, сумісність, логування і моніторинг, масштабованість [36-45].

Розробка скриптів із використанням модульного підходу дозволяє легко повторно використовувати компоненти, що знижує витрати на підтримку та розвиток. Наприклад, скрипти для роботи з DNS-записами доцільно розділити на окремі модулі (для створення, видалення, зміни записів).

Принцип безпеки означає, що Python-скрипти, які взаємодіють із мережевими ресурсами, повинні мати обмежені права доступу і виконуватися в

захищених середовищах (наприклад, за допомогою ізоляції на віртуальних машинах або контейнерах).

Сумісність означає, що скрипти повинні бути кросплатформенними і адаптованими для виконання на різних операційних системах. Це можна забезпечити, використовуючи бібліотеки, які одночасно підтримують Windows, Linux та macOS.

Для відстеження стану виконання скриптів важливо реалізувати функції логування. Дотримання цього принципу дозволяє швидко знаходити причини помилок і забезпечувати стабільність роботи.

Python-скрипти повинні бути здатними працювати в умовах великих мережевих навантажень і мати можливість масштабуватися з мінімальними змінами.

Розглянутий нижче приклад коду Python-скрипту, ілюструє автоматизацію міграції DNS-записів з логуванням результатів міграції:

```
import logging

# Налаштування логування
logging.basicConfig(filename='migration.log',
level=logging.INFO)

def migrate_dns_record(domain, new_ip):
    logging.info("Початок міграції для домену %s",
domain)
    # Код для міграції DNS-запису
    try:
        # Симуляція успішного оновлення запису
        logging.info("DNS-запис для %s успішно
оновлено на %s", domain, new_ip)
    except Exception as e:
        logging.error("Помилка міграції для %s: %s",
domain, str(e))

migrate_dns_record("example.com", "192.168.1.1")
```

Цей код реалізує логіку міграції DNS-записів, фіксує події у файлі migration.log та обробляє можливі помилки. Реалізація міграції лише симулюється, але структура коду дозволяє розширити його функціонал, додавши

реальні операції з DNS. Структуру коду ілюструє блок-схема представлена на рисунку 1.3.

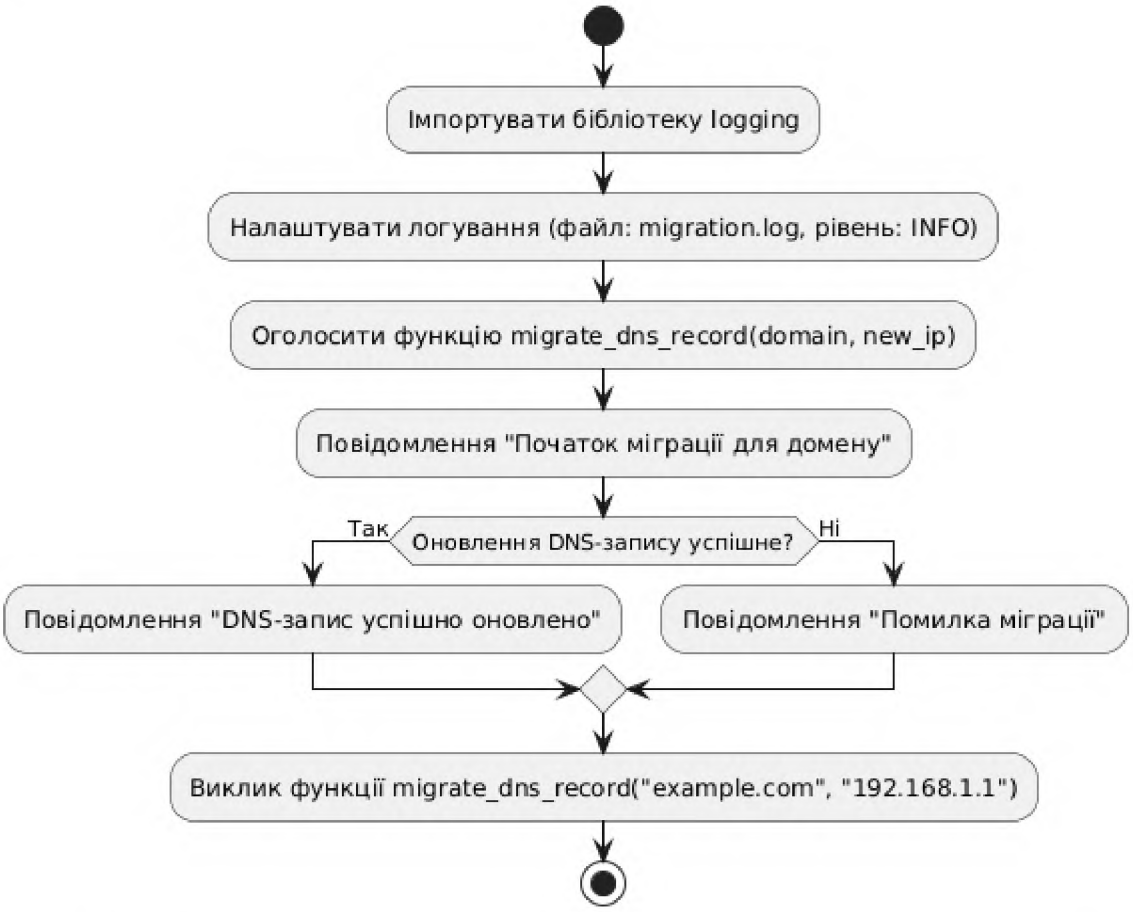

Рисунок 1.3 – Алгоритм виконання Python-скрипту для автоматизації міграції DNS-записів з логуванням результатів

На початку коду імпортується бібліотека logging, яка дозволяє створювати журнали подій (логування) для відстеження роботи програми:

```
import logging
```

Відразу після цього виконується налаштування логування:

```
logging.basicConfig(filename='migration.log',
level=logging.INFO)
```

Значення змінної filename='migration.log' вказує, що всі записи логів будуть зберігатися у файлі migration.log. Значення level=logging.INFO встановлює рівень логування: у цьому випадку до лог-файлу migration.log будуть записуватись події рівня INFO та вище (наприклад, ERROR).

У наступних рядках описана функція для міграції DNS-записів:

```
def migrate_dns_record(domain, new_ip):
    logging.info("Початок міграції для домену %s", domain)
```

Функція migrate_dns_record приймає два параметри:

domain – доменне ім'я для міграції (наприклад, example.com);

new_ip – нова IP-адреса для оновлення DNS-запису;

Метод logging.info записує повідомлення у лог-файл про те, що міграція для домену розпочалася.

Основна логіка міграції описується у блоці try: використання try вказує на те, при виконанні міграції може статися помилка, яку потрібно обробити. У наведеному коді фактичне оновлення DNS-запису не реалізоване, використовується лише симуляція успішного оновлення запису: метод logging.info робить запис про успішне оновлення DNS-запису для домену у файл логів.

Обробку помилок здійснює наступний код:

```
except Exception as e:
    logging.error("Помилка міграції для %s: %s", domain, str(e))
```

Блок except виконується, якщо під час оновлення DNS-запису виникає помилка: метод logging.error записує повідомлення про помилку у файл логів; цей запис включає домен та описання помилки.

В останньому рядку викликається описана вище функція migrate_dns_record для домену example.com з новою IP-адресою 192.168.1.1.

```
migrate_dns_record("example.com", "192.168.1.1")
```

Розглянутий програмний код ілюструє дотримання принципу логування та моніторингу під час використання скриптів Python для автоматизованої міграції доменів.

1.4 Переваги використання Python у міграції доменів

Завдяки своїм широким можливостям і доступності, Python є одним із найбільш популярних інструментів для автоматизації мережевих операцій, зокрема міграції доменів [46-49]. Переваги Python для автоматизації міграції доменів:

- відкритий код та безкоштовне використання, тому він є доступним для спеціалістів та компаній різних рівнів, незалежно від бюджету;
- має значну кількість спеціалізованих бібліотек, таких як DNSPython, Paramiko та Fabric, які забезпечують автоматизацію DNS-запитів, SSH-з'єднань та управління серверами;
- працює на більшості операційних систем (Windows, macOS, Linux), тобто є універсальним для різних мережевих середовищ;
- має зрозумілий синтаксис, що значно полегшує написання скриптів та їх обслуговування;
- дозволяє легко змінювати та налаштовувати скрипти, щоб адаптуватися до вимог конкретного середовища міграції.

У таблиці 1.3 представлене порівняння характеристик таких інструментів автоматизації, як Python, Bash та Ansible з аргументацією на користь того чи іншого інструменту.

Таблиця 1.3 – Порівняння Python з Bash та Ansible

Критерій	Python	Bash	Ansible
Швидкість	Швидке виконання для середніх завдань	Висока швидкість для простих задач	Швидке розгортання для великих інфраструктур
Масштабованість	Легко інтегрується у більші проєкти	Обмежена через складність налаштування	Добре підходить для масштабованих середовищ
Гнучкість	Дуже гнучкий та розширюваний	Менш гнучкий через простоту синтаксису	Підтримує декларативний стиль, обмежений у налаштуванні
Кросплатформеність	Працює на більшості платформ	Переважно Linux	Підтримка всіх основних платформ
Складність	Помірна (вимагає знання Python)	Низька для базових задач	Помірна, зручна для адміністраторів

У таблиці 1.4 представлені умовні оцінки інструментів автоматизації Python, Bash та Ansible за 10-бальною шкалою, де 0 – означає найнижчий рівень оцінки інструменту за відповідним критерієм, а 10 – відповідно, найвищий рівень. Дана таблиця дозволяє порівняти інструменти за основними критеріями для вибору найкращого рішення залежно від завдань.

Таблиця 1.4 – Умовні оцінки Python, Bash та Ansible за шкалою від 0 до 10

Критерій	Python	Bash	Ansible
Швидкість	8	9	7
Масштабованість	8	5	10
Гнучкість	10	6	8
Кросплатформеність	9	7	9
Складність	7	5	8

Виходячи з даних таблиці 1.6 можна зробити висновок про те, що Bash-скрипти автоматизації мають найвищу швидкість виконання (для простих задач). Використання Python забезпечує високу швидкість (для середніх задач), програмний засіб Ansible є дещо повільнішим за конкурентів через його декларативний стиль описання та виконання процесів автоматизації і суттєву залежність від інфраструктури.

Можна відмітити, що Python досить добре інтегрується у більші проєкти, на відміну від Bash, що має обмежену масштабованість через складність обслуговування великих скриптів. У той же час, Ansible найкраще з усіх трьох інструментів підходить для великих інфраструктур. Python з усіх трьох є найбільш гнучким і розширюваним інструментом. Bash менш гнучкий через його простий синтаксис. Ansible підтримує декларативний стиль, але його налаштування обмежені. Python і Ansible працюють на більшості платформ, можуть використовуватись у переважній більшості середовищ. Щодо Bash, то він найкраще функціонує у середовищі Linux.

Bash є найпростішим у використанні для базових завдань. Python має помірну складність, оскільки вимагає знання мови. Ansible також помірно складний, але при цьому є зручним для системних адміністраторів.

У таблиці 1.5 представлене порівняння інструментів Python з можливостями програмного засобу Ansible.

Таблиця 1.5 – Порівняння Python з можливостями Ansible

Параметр	Python (DNSPython, Paramiko, Fabric)	Ansible
Простота використання	Вимагає знань Python, налаштування середовища	Простий у використанні для великих інфраструктур
Масштабованість	Підходить для невеликих середовищ	Висока масштабованість, підходить для великих мереж
Автоматизація	Можливість створювати власні скрипти для автоматизації	Використовує декларативний підхід, що знижує помилки
Інтеграція	Може бути налаштовано під конкретні потреби	Легко інтегрується з іншими DevOps-інструментами
Управління	Менше функціональності для управління	Повноцінне управління конфігураціями та інвентаризація

Загалом, у підсумку, Python виявляється таким інструментом, що переважає інші інструменти автоматизації за сукупністю розглянутих критеріїв.

Отже, можна вважати, що вибір Python як основного інструмента для автоматизації міграції доменів є оптимальним рішенням, враховуючи його гнучкість, широкий набір різноманітних бібліотек, кросплатформенність, а також активну Python-спільноти. Переваги Python для автоматизації міграції доменів узагальнені у таблиці 1.6.

Таблиця 1.6 – Переваги Python в автоматизації міграції доменів

Переваги	Опис
Доступність	Python є безкоштовним і доступним для широкого використання
Кросплатформенність	Працює на Windows, macOS, Linux
Швидкість розробки	Простий синтаксис і швидка розробка скриптів
Широкий набір бібліотек	Наявність бібліотек для мережевих операцій: DNSPython, Paramiko, Fabric
Гнучкість	Легко адаптується до специфічних вимог
Масштабованість	Інтегрується з іншими інструментами DevOps для роботи у великих середовищах
Підтримка спільноти	Велика кількість документації та ресурсів для навчання, що спрощує процес впровадження Python у середовище автоматизації

Таким чином, Python є універсальним і гнучким інструментом для автоматизації міграції доменів. І хоча Bash краще підходить для простих завдань

але Python забезпечує більше можливостей для налаштування, інтеграції та обробки складних процесів. Водночас, Ansible є потужним інструментом для великих інфраструктур, але може бути менш гнучким для індивідуальних налаштувань у порівнянні з Python.

Висновки до розділу 1

Проведений огляд існуючих методів автоматизації міграції доменів дозволяє узагальнити основні переваги та недоліки кожного з існуючих підходів.

Традиційні методи міграції доменів, що включають ручні налаштування та скрипти на основі Bash, забезпечують базову функціональність але мають значні обмеження у масштабованості та залежать від людського фактору, що підвищує ризик помилок.

Інструменти DevOps, такі як Ansible, добре підходять для масштабованих середовищ, але їх декларативний підхід обмежує можливості індивідуальних налаштувань.

Вибір Python як основного інструмента для автоматизації міграції доменів є оптимальним рішенням завдяки його гнучкості, широкому набору бібліотек, кросплатформенності та активній підтримці спільноти. Python поєднує переваги високої швидкості розробки та можливість глибокого налаштування для конкретних потреб, що особливо важливо в корпоративних середовищах, де точність і надійність мають вирішальне значення. Завдяки таким бібліотекам, як DNSPython, Paramiko та Fabric, Python дозволяє здійснювати автоматизацію всіх етапів міграції доменів – від налаштування DNS-записів до безпечного SSH-з'єднання з віддаленими серверами. Інтеграція Python-скриптів у корпоративне середовище з дотриманням принципів модульності, безпеки та логування здата забезпечити надійну та гнучку основу для автоматизованої міграції доменів.

РОЗДІЛ 2
ПРОЦЕСИ ТА АЛГОРИТМИ АВТОМАТИЗОВАНОЇ МІГРАЦІЇ ДОМЕНІВ

2.1 Вимоги до автоматизованої міграції доменів

Для забезпечення ефективної автоматизованої міграції доменів важливо визначити вимоги до цього процесу. Ці вимоги охоплюють надійність, ефективність, гнучкість, а також включають показники, які дозволяють оцінити ефективність виконання міграції. Крім того, ці вимоги мають враховувати складність існуючої інфраструктури та забезпечувати масштабованість рішень, що дозволить застосовувати методику для великих проєктів або організацій із численними доменами [50-57].

Надійність. Автоматизований процес міграції повинен гарантувати цілісність даних та безперебійну роботу всіх мережевих ресурсів після переходу. Важливо уникати будь-яких помилок, що можуть порушити доступність домену або вплинути на його роботу. Надійність досягається через автоматичне резервне копіювання DNS-записів перед міграцією, а також перевірку цілісності налаштувань після завершення процесу. Окрім цього, інтеграція систем моніторингу може допомогти вчасно виявляти проблєми та зменшити ризик порушення доступності під час міграції, забезпечуючи оперативне реагування.

Ефективність. Процес автоматизації повинен скорочувати час, необхідний для міграції, з мінімальним втручанням адміністратора. Це включає швидке виконання всіх етапів міграції та зменшення часу простою. Ефективність забезпечується використанням скриптів, які можуть швидко та автоматично налаштувати потрібні параметри, обійшовши ручні дії, які вимагають більше часу. До того ж, ефективність може бути підвищена завдяки використанню паралельного виконання завдань для масштабних інфраструктур або сценаріїв, які дозволяють налаштовувати кілька доменів одночасно.

Гнучкість. Автоматизована система повинна бути достатньо гнучкою для адаптації до різних інфраструктурних налаштувань та специфічних вимог

організації, таких як налаштування різних типів DNS-записів (A, CNAME, MX тощо). Використання Python-скриптів дозволяє налаштовувати параметри відповідно до специфічних потреб, в той час як Python забезпечує можливість роботи у різних операційних системах. Більше того, розширення функціоналу за допомогою додаткових бібліотек, таких як dnspython або paramiko, дозволяє автоматизувати не тільки зміни DNS-записів, а й управління серверами, що робить процес універсальним для організацій будь-якого масштабу.

Такий підхід до визначення вимог забезпечує всебічну адаптацію процесу міграції до конкретних умов і дозволяє гарантувати стабільність, продуктивність і гнучкість автоматизованої системи.

Основні вимоги до автоматизації міграції доменів узагальнені у таблиці 2.1.

Таблиця 2.1 – Основні вимоги до автоматизації міграції доменів

Вимога	Опис	Реалізація
Надійність	Захист даних та безперебійна робота після міграції	Резервне копіювання та перевірка цілісності записів
Ефективність	Скорочення часу та мінімальне втручання адміністратора	Швидке виконання скриптів та автоматичне налаштування
Гнучкість	Можливість адаптації до різних інфраструктур і вимог організації	Налаштування параметрів для різних типів DNS-записів, кросплатформенність

Для оцінки ефективності автоматизованої міграції доменів використовують показники ефективності, які дозволяють точно виміряти продуктивність і якість процесу: час виконання міграції, відсоток помилок, час простою домену, гнучкість налаштувань.

Час виконання міграції (Migration Time) – загальний час, необхідний для завершення процесу міграції. Менший час свідчить про високу ефективність автоматизованої системи. Мета: мінімізувати час, необхідний для виконання всіх етапів міграції, без шкоди для надійності.

Відсоток помилок (Error Rate) – відсоток помилок, що виникають під час міграції, які можуть призвести до збоїв або проблєм з доступністю домену. Мета: мінімізувати кількість помилок, які можуть порушити роботу домену після завершення міграції.

Час простою домену (Downtime) – період, протягом якого домен недоступний для користувачів. Цей показник повинен бути мінімізований для забезпечення стабільності мережі. Мета: забезпечити безперервний доступ до домену з мінімальним простоєм.

Гнучкість налаштувань (Configuration Flexibility) – здатність системи налаштовувати різні типи DNS-записів та адаптуватися до вимог різних платформ. Мета: максимально підвищити можливість налаштування, щоб система могла обробляти складні інфраструктури.

Характеристики показників ефективності автоматизованої міграції доменів узагальнені у таблиці 2.2.

Таблиця 2.2 – Показники ефективності автоматизованої міграції доменів

Показник	Опис	Мета
Час виконання міграції	Загальний час завершення процесу міграції	Мінімізувати
Відсоток помилок	Відсоток помилок, що виникають під час міграції	Знизити до мінімуму
Час простою домену	Період недоступності домену під час міграції	Забезпечити мінімальний простій
Гнучкість налаштувань	Можливість налаштовувати параметри та адаптуватися до різних інфраструктур	Максимальна гнучкість

Показники, представлені у таблиці 2.2, дозволяють оцінити ефективність автоматизованої міграції доменів та забезпечити стабільну, безперебійну роботу домену після завершення процесу.

2.2 Оцінювання ефективності міграції доменів

Ефективність автоматизованої міграції доменів можна оцінити через аналіз її впливу на основні показники, такі як час виконання, ризик помилок та економія ресурсів. Автоматизація може зменшити час виконання міграції, знизити кількість помилок і сприяти оптимізації використання ресурсів [58-61].

Автоматизація значно впливає на зменшення часу, необхідного для міграції доменів, а також знижує ризик помилок, які часто виникають при ручному налаштуванні, забезпечує стабільність процесу міграції.

Дійсно, автоматизовані скрипти виконують налаштування DNS-записів та інших параметрів швидше, ніж це можливо вручну, особливо при великих обсягах доменів. Автоматизація виключає потребу в ручному введенні даних, що зменшує ризик допущення помилок, таких як некоректні DNS-записи або неправильні конфігурації. Автоматизовані процеси більш послідовні та передбачувані, що дозволяє зменшити варіативність результатів та забезпечити однаковий рівень якості на кожному етапі міграції. Таблиця 2.3 ілюструє вплив автоматизації на показники міграції доменів.

Таблиця 2.3 – Вплив автоматизації на показники міграції доменів

Показник	Вплив автоматизації
Час виконання	Скорочення загального часу міграції за рахунок швидкого виконання скриптів
Ризик помилок	Зниження кількості помилок завдяки автоматичному виконанню та зниженню людського фактора
Стабільність	Послідовність та передбачуваність процесу, що забезпечує однакову якість міграції

Для об’єктивної оцінки ефективності автоматизованої міграції доменів використовуються кілька критеріїв, які дозволяють виміряти продуктивність процесу. Основними з них є швидкість, точність та економія ресурсів.

Для розрахунку швидкості вимірюється час, необхідний для завершення міграції: чим швидше відбувається міграція, тим ефективніший процес. Час виконання міграції вимірюється в секундах або хвилинах. Цей показник застосовується для визначення середнього часу виконання скриптів та оптимізація швидкості.

Точність показує кількість помилок, які виникають під час виконання автоматизації. Висока точність забезпечує коректність налаштувань DNS та інших параметрів. Метрикою точності є відсоток помилок (наприклад, кількість помилок

на 1000 DNS-записів). Показник точності використовується для виявлення та виправлення некоректних конфігурацій у скриптах для підвищення точності.

Економія ресурсів означає зменшення потреби в ресурсах, таких як час адміністратора, завдяки застосуванню автоматизації. Економія ресурсів дозволяє фахівцям займатися іншими завданнями. Мірою цього показника є кількість зекономлених годин роботи, зменшення витрат на обслуговування. Цей показник застосовується для порівняння витрат часу на ручну міграцію з автоматизованою, визначення економії.

Основні відомості про критерії оцінки ефективності міграції доменів представлені у таблиці 2.4.

Таблиця 2.4 – Основні показники міграції доменів та їх застосування

Критерій	Опис	Метрика	Застосування
Швидкість	Час виконання міграції	Час виконання міграції	Визначення середнього часу та оптимізація
Точність	Коректність налаштувань, мінімізація помилок	Відсоток помилок	Виявлення некоректних конфігурацій
Економія ресурсів	Зменшення потреби в робочих ресурсах завдяки автоматизації	Години роботи адміністратора	Порівняння витрат часу між ручною та автоматизованою міграцією

Ця таблиця показує основні критерії оцінки ефективності автоматизованої міграції, які допомагають визначити вплив автоматизації на продуктивність, точність і економію ресурсів, що дозволяє забезпечити стабільну роботу домену після міграції.

2.3 Процеси міграції доменів у корпоративному середовищі

Для описання процесу автоматизованої міграції доменів у корпоративному середовищі розглядається модельний приклад компанії «TechSolutions», яка використовує декілька доменів для підтримки своїх внутрішніх та зовнішніх сервісів. Компанія планує перенести свої домени від одного хостинг-провайдера

до іншого, одночасно змінивши інфраструктуру DNS на більш захищене і масштабоване середовище.

Характеристики корпоративної IT-інфраструктури компанії TechSolutions включають відомості про доменну структуру, DNS-записи, середовище для хостингу доменів та сервери для тестування.

Доменна структура. Компанія має три основні домени:

techsolutions.com – основний сайт компанії;

intranet.techsolutions.com – внутрішня мережа для співробітників;

api.techsolutions.com – API для зовнішніх партнерів.

DNS-записи. Для кожного домену існують різні записи (A, CNAME, MX та TXT), які обслуговують як внутрішні сервіси (напр. пошта), так і зовнішні ресурси (наприклад, доступ до API).

Середовище для хостингу доменів. Наразі домени прив'язані до хостингу провайдера OldHost, але компанія планує перенести їх до NewHost для того, щоб покращити масштабованість та підвищити захист даних.

Сервери для тестування. Використовуються підготовлені тестові сервери для перевірки коректності DNS-записів перед повною міграцією.

Завданням автоматизації є міграція DNS-записів усіх доменів з OldHost на NewHost, з одночасною перевіркою коректності кожного запису та створенням резервних копій налаштувань.

Для проведення експериментів налаштовується тестова інфраструктура, яка дозволяє перевірити процес міграції доменів без впливу на основні сервіси компанії.

Етапи налаштування тестової інфраструктури [62-64]:

- створення середовища для тестування;
- розробка Python-скриптів для автоматизації міграції;
- тестування коректності міграції;
- використання логування та звітів.

Створення середовища для тестування передбачає встановлення локального DNS-сервера, який симулює роботу нового хостинг-провайдера NewHost та

налаштування тестових записів для кожного домену (techsolutions.com, intranet.techsolutions.com, api.techsolutions.com) на цьому сервері для перевірки коректності міграції.

Блок-схема алгоритму реалізації міграції DNS-записів, представлена на рисунку 2.1.

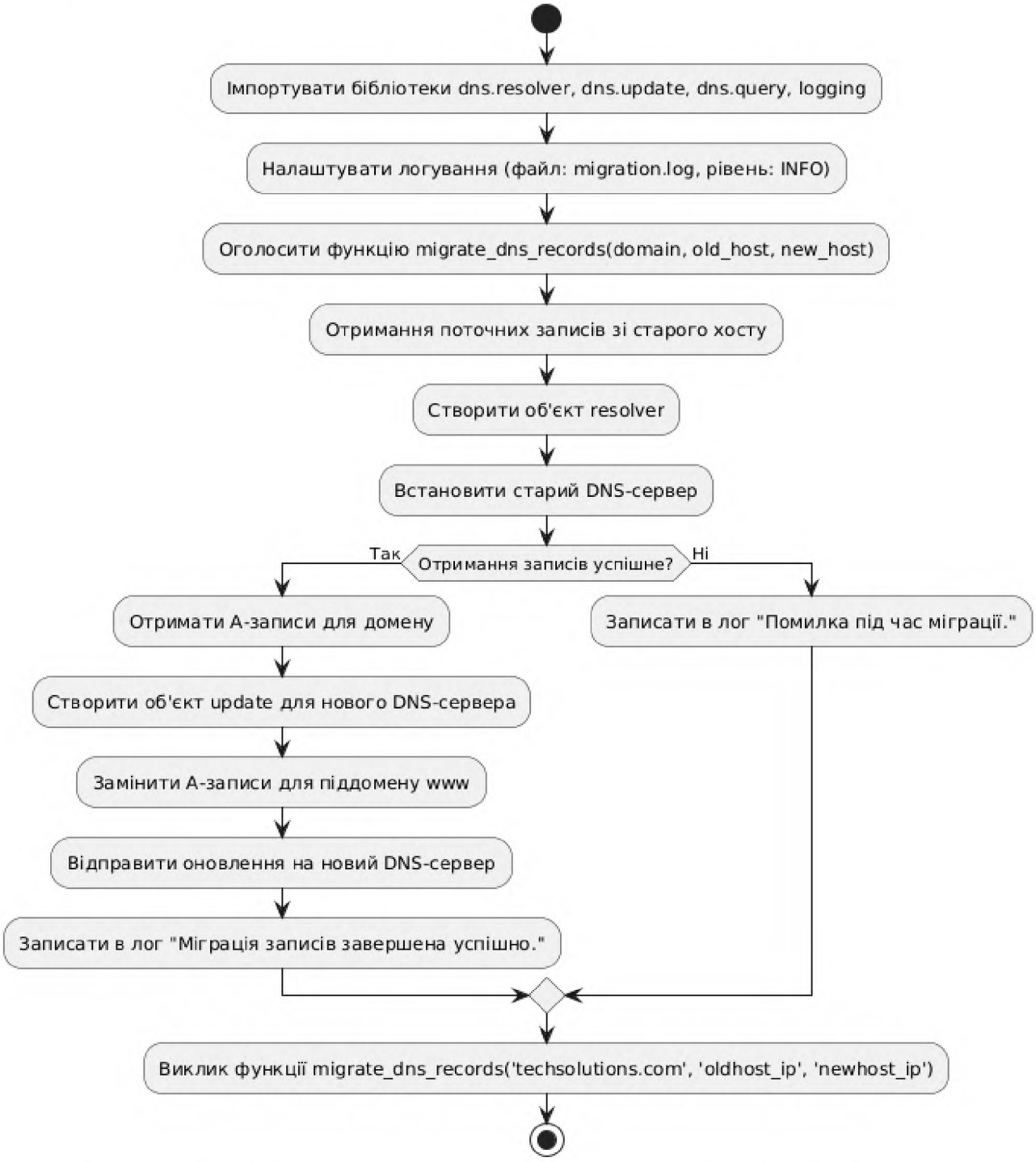

Рисунок 2.1 – Блок-схема алгоритму реалізації міграції DNS-записів

Розробка Python-скриптів для автоматизації міграції включає розробку скриптів для автоматичного зчитування DNS-записів з OldHost і копіювання їх до середовища NewHost, а також додавання функцій резервного копіювання, щоб зберігати початкові налаштування на випадок помилок [65-67].

Тестування коректності міграції відбувається таким чином: після перенесення кожного запису на NewHost виконується автоматизоване тестування за допомогою скриптів для перевірки доступності кожного домену. Після цього виконується виявлення можливих розбіжностей у налаштуваннях та автоматичне виправлення їх у скрипті.

Використання логування та звітів включає логування всіх операцій перенесення та перевірку кожного DNS-запису з наступним формуванням звітів про статус міграції для подальшого аналізу.

Нижче представлена реалізація Python-скрипта для міграції DNS-записів:

```
import dns.resolver
import dns.update
import dns.query
import logging

# Налаштування логування
logging.basicConfig(filename='migration.log',
level=logging.INFO)

# Функція для перенесення DNS-записів
def migrate_dns_records(domain, old_host, new_host):
    try:
        # Отримання поточних записів із старого хосту
        resolver = dns.resolver.Resolver()
        resolver.nameservers = [old_host]
        records = resolver.resolve(domain, 'A')

        # Налаштування нового DNS-сервера
        update = dns.update.Update(domain)
        for record in records:
            update.replace('www', 300, 'A',
record.to_text())

        # Відправка на новий хост
        dns.query.tcp(update, new_host)
```

```
            logging.info(f"Міграція записів для {domain}
завершена успішно.")

        except Exception as e:
            logging.error(f"Помилка під час міграції
{domain}: {e}")

# Виконання міграції для домену techsolutions.com
migrate_dns_records('techsolutions.com', 'oldhost_ip',
'newhost_ip')
```

Застосування такого скрипту дозволяє автоматизувати процес перенесення DNS-записів між різними DNS-серверами, що є корисним у багатьох випадках:

1. У разі зміни провайдера хостингу або серверного обладнання (міграція інфраструктури), коли необхідно перенести всі DNS-записи з одного сервера на інший, скрипт автоматизації зменшує ризик помилок, які можуть виникати під час ручного оновлення DNS-записів;

2. Якщо мережа розширюється (масштабування мережі), такий скрипт дозволяє швидко перенести DNS-записи на нові сервери, забезпечуючи безперервність роботи послуг, що особливо важливо для великих компаній з численними піддоменами, наприклад, www, api, mail тощо;

3. Скрипт автоматизації зручно використовувати для відновлення DNS-записів у разі аварійного перемикання на інший сервер (резервне копіювання та відновлення); це забезпечує безперервність роботи у разі виходу з ладу основного DNS-сервера;

4. Скрипт дозволяє адміністраторам здійснювати централізоване управління DNS-записами (централізовано оновлювати записи), зменшуючи час, витрачений на ручне налаштування, наприклад, для перенесення десятків доменів на новий сервер;

5. Скрипт легко інтегрується з іншими інструментами DevOps (інтеграція у DevOps-процеси), такими, як Ansible або Jenkins, для виконанння завдань автоматизованого розгортання, що дозволяє виконувати міграцію DNS-записів у рамках CI/CD процесів.

Головні переваги від застосування цього скрипту: швидкість – скрипт може обробляти багато доменів і записів за короткий час; точність – використання автоматизації мінімізує ризик людських помилок; гнучкість – скрипт легко адаптується до різних типів записів (A, CNAME, MX) та мережевих конфігурацій; масштабованість – можна застосовувати як для невеликих мереж, так і для великих корпоративних інфраструктур. Загалом, скрипт є зручним рішенням для адміністраторів, які керують мережею, оскільки він автоматизує рутинні задачі, підвищує надійність перенесення DNS-записів і забезпечує гнучкість для масштабованих мережевих рішень.

Налаштоване таким чином тестове середовище та відповідні Python-скрипти дозволяють безпечно виконати міграцію доменів, зберігаючи всі конфігураційні дані та забезпечуючи безперебійну роботу. Цей спосіб дозволяє мінімізувати ризики та полегшує процес міграції в реальній корпоративній інфраструктурі.

2.4 Оцінка результатів автоматизованої міграції доменів

Після завершення автоматизованої міграції доменів важливо провести оцінку її результатів, щоб визначити, наскільки ефективним був цей процес [68-72].

Для порівняння отриманих результатів використовуються такі показники:

- час виконання міграції (показує різницю в продуктивності між ручною і автоматизованою міграціє);
- відсоток помилок (показує кількість помилок, які виникають під час міграції; автоматизація має зменшити цей показник);
- час простою домену – тривалість періоду, коли домен недоступний під час міграції;
- економія робочого часу адміністратора (показує зниження потреби у втручанні адміністратора завдяки автоматизації).

Кількісні результати оцінки результатів автоматизованої міграції доменів у модельному середовищі представлені у таблиці 2.5.

Таблиця 2.5 – Порівняння результатів до і після автоматизації: кількісні та якісні показники

Показник	До автоматизації	Після автоматизації	Відмінність
Час виконання міграції	2-3 години	15-30 хвилин	Скорочення на 75-85%
Відсоток помилок	5-10%	1-2%	Зниження на 80%
Час простою домену	до 30 хвилин	до 5 хвилин	Зниження на 83%
Економія робочого часу	Інтенсивне втручання	Мінімальне втручання	Зниження навантаження

На основі отриманих даних щодо часу, відсотка помилок і простою, можна зробити висновки щодо ефективності автоматизованої міграції доменів.

На рисунку 2.2 представлена діаграма порівняння показників виконання міграції до та після автоматизації.

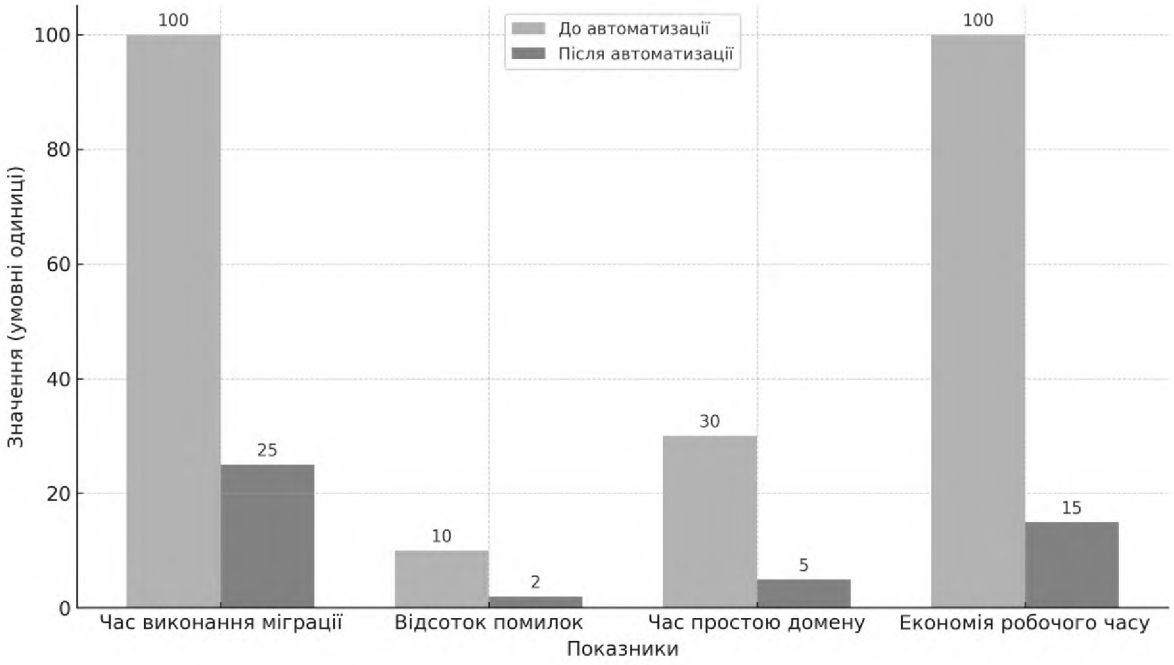

Рисунок 2.2 – Порівняння показників виконання міграції до та після автоматизації

Аналіз цих показників дозволяє відмітити наступне:

- час виконання міграції значно скоротився, що свідчить про те, що автоматизація значно підвищує ефективність процесу. Замість кількох годин на

міграцію кожного домену, автоматизований процес виконується за лічені хвилини;

- відсоток помилок суттєво зменшився, що вказує на надійність автоматизованого підходу. Завдяки автоматизації виключається багато ризиків, пов'язаних із людським фактором;

- набагато зменшився час недоступності домену, що позитивно впливає на користувачів і загальну стабільність сервісу;

- велика економія робочого часу адміністратора показує, що автоматизація зменшила навантаження на адміністраторів. Це вивільняє їх робочий час для виконання інших завдань та сприяє підвищенню загальної продуктивності IT-відділу компанії.

Загалом, представлені дані демонструють перевагу ефективності автоматизованої міграції доменів: скорочення часу виконання, зменшення відсотку помилок та часу простою свідчать про успішну реалізацію автоматизації процесу міграції доменів.

2.5 Аналіз використаних технологій

Аналіз технологій, застосованих для автоматизації міграції доменів, допомагає оцінити їхню ефективність та виявити обмеження, які можуть вплинути на продуктивність і надійність процесу [73]. Далі аналізуються можливості Python-інструментів для міграції доменів та визначаються потенційні вдосконалення, що можуть підвищити ефективність автоматизованого підходу.

Основними інструментами Python, що використовуються для автоматизації міграції доменів, є Python-бібліотеки DNSPython, Paramiko і Fabric, що мають як спільні риси, так і відмінності у використанні. DNSPython, Paramiko та Fabric – це потужні інструменти, які дозволяють автоматизувати різноманітні завдання, пов'язані з мережевою інфраструктурою та управлінням серверами [74-79]. Спільним для них є те, що:

- всі ці бібліотеки написані на мові Python, що дозволяє легко інтегрувати їх в інші Python-проєкти;

- всі вони дозволяють взаємодіяти з мережевими пристроями, такими як сервери, через різні протоколи;

- головна мета цих бібліотек – автоматизувати рутинні завдання, щоб підвищити ефективність роботи.

Базова інформація про Python-бібліотеки, що використовуються для автоматизації міграції доменів представлена у таблиці 2.6.

Таблиця 2.6 – Базова інформація про основні інструменти Python для автоматизації міграції доменів

Інструмент	Переваги	Недоліки
DNSPython	Підтримка різних типів записів, простота	Обмежена інтеграція, відсутність моніторингу
Paramiko	Безпечне SSH-з'єднання, можливість віддалених команд	Складне налаштування, обмежена підтримка асинхронних завдань
Fabric	Легкість налаштування, зручний синтаксис	Менш ефективний для складних завдань, відсутність моніторингу

DNSPython забезпечує доступ до DNS-запитів і дозволяє змінювати DNS-записи. Це робить його корисним для роботи з різними типами записів (A, CNAME, MX, TXT). Її перевагами є простота використання, підтримка різних DNS-запитів, надійність. Недоліками DNSPython є обмежена підтримка функцій для інтеграції з великими корпоративними середовищами, а також відсутність вбудованих засобів для моніторингу.

Paramiko дозволяє встановлювати SSH-з'єднання для управління віддаленими серверами, що є корисним для автоматизації дій під час перенесення налаштувань доменів. Перевагами є безпечне з'єднання та можливість віддаленого виконання команд. До недоліків можна віднести та, що автоматизація з використанням цієї бібліотеки може бути складною у налаштуванні при великій кількості серверів. Також, у Paramiko обмежена підтримка асинхронних завдань.

Fabric спрощує виконання команд на віддалених серверах через SSH, дозволяючи об'єднати їх у сценарії для автоматизації. Головними перевагами

Fabric є легкість налаштування для стандартних завдань та зручний синтаксис. Як недоліки відмітимо, що використання Fabric може бути не таким ефективним при роботі з великою кількістю складних команд, а також відсутність вбудованого моніторингу стану серверів.

При всіх перевагах розглянутих вище Python-інструментів, у деяких питаннях, пов'язаних автоматизацією міграції доменів, вони можуть бути недостатньо ефективними. Зокрема, це стосується управління великою кількістю серверів через потребу у послідовному виконанні команд, що знижує загальну продуктивність процесу. Друга проблєма, на яку слід звернути увагу, є обмежена інтеграція з існуючими корпоративними системами, оскільки DNSPython і Fabric можуть підтримувати не всі типи корпоративних DNS-налаштувань, і це може вимагати додаткового налаштування їх сумісності. Крім того, жоден із цих інструментів не має засобів вбудованого моніторингу, що ускладнює відстеження стану серверів і DNS-записів під час міграції [73].

Таким чином, основними обмеженнями використання інструментів Python у контексті автоматизації міграції доменів є:

- відсутність інструментів масштабованості для великих інфраструктур;
- обмежені можливості інтеграції з існуючими корпоративними системами;
- відсутність вбудованих засобів моніторингу.

Можливими рішеннями щодо подолання цих обмежень та покращення результатів використання інструментів Python для автоматизації доменів можуть бути:

- впровадження паралельного виконання завдань при роботі з великою кількістю серверів;
- інтеграція з системами моніторингу;
- розширення бібліотек для підтримки складних корпоративних середовищ.

Зокрема, одним з можливих рішень щодо впровадження паралельного виконання завдань, є використання асинхронних бібліотек Python (наприклад, AsyncIO) або комбінування Python-скриптів з інструментами, що підтримують паралельне виконання команд (наприклад, Ansible).

Для покращення контрольованості та стабільності процесу міграції можна використовувати інтеграцію Python-скриптів із системами моніторингу (наприклад, Prometheus). Це дозволить відстежувати стан DNS-записів і доступність доменів у реальному часі.

У випадку автоматизованої міграції складних та нестандартних корпоративних середовищ, можливо, що буде необхідно самостійно розробити відповідні додаткові модулі для DNSPython або інтегрувати Python-скрипти з іншими засобами, які підтримують складніші конфігурації DNS.

Основні відомості про інструментами Python, що використовуються для автоматизації міграції доменів, узагальнені у таблиці 2.7.

Таблиця 2.7 – Узагальнені відомості про бібліотеки DNSPython, Paramiko і Fabric

Бібліотека	Відмінності	Переваги	Обмеження	Використання
DNSPython	Спеціалізується на роботі з DNS. Запити до DNS-серверів, оновлення записів DNS.	Спеціалізований інструмент для DNS. Простота використання	Обмежений функціонал (тільки DNS).	Для автоматизації DNS-завдань, таких як оновлення DNS-записів чи отримання інформації про DNS.
Paramiko	Низькорівнева взаємодія через SSH. Виконання команд, передача файлів, SFTP-з'єднання.	Гнучкий інструмент. Широкий спектр завдань, пов'язаних із SSH.	Потребує більше коду через низький рівень абстракції.	Для низькорівневої взаємодії з SSH, написання власних інструментів управління конфігурацією.
Fabric	Побудована на Paramiko. Високий рівень абстракції. Автоматизація розгортання та керування конфігурацією	Зручна для розгортання додатків. Виконання команд на декількох серверах одночасно.	Надмірність для простих завдань. Залежить від Paramiko	Для автоматизації розгортання додатків та керування конфігурацією на багатьох серверах одночасно.

Дана таблиця демонструє, в яких випадках і як найкраще застосовувати кожну з розглянутих бібліотек Python, демонструє їх основні особливості, переваги та недоліки.

Таким чином, проведений аналіз представлених інструментів показує, що, незважаючи на їх ефективність для стандартних завдань, Python-інструменти автоматизації міграції доменів мають певні обмеження, перш за все, для великих корпоративних середовищ. Пропоновані покращення, такі як використання паралельного виконання скриптів та інтеграція з сторонніми моніторинговими системами, дозволить підвищити ефективність і стабільність автоматизованого процесу міграції.

Висновки до розділу 2

Проведений аналіз автоматизованої міграції доменів свідчать на користь гіпотези про ефективність автоматизації цього процесу. Результати проведеного аналізу показують, що автоматизація суттєво скорочує час міграції, зменшує кількість помилок, знижує час простою та оптимізує робочі ресурси. При цьому, використання спеціалізованих Python-інструментів, призначених для управління доменами, таких як бібліотеки DNSPython, Paramiko та Fabric, дозволяє автоматизувати всі етапи міграції, забезпечити послідовність, надійність та контрольованість процесу.

Загалом, автоматизована міграція доменів за допомогою скриптів Python здатна забезпечити значне покращення основних показників цього процесу, а саме: скорочення часу виконання на 75-85%, що підтверджує високу швидкість роботи автоматизованих скриптів порівняно з ручними методами; зниження відсотку помилок у процесі міграції на 80% завдяки мінімізації впливу людського фактору та забезпеченню стабільної роботи процесу; зменшення часу простою домену внаслідок міграції на 83%, що підвищує загальну доступність сервісів для кінцевих користувачів; економію робочого часу адміністратора, що дозволяє скоротити навантаження на IT-відділ компанії і зосередитись на інших завданнях.

Однак, у той же час, проведений аналіз також виявив деякі суттєві обмеження у застосуванні інструментів Python для автоматизації міграції доменів

у великих корпоративних середовищах, такі як недостатня підтримка масштабованості та відсутність засобів моніторингу процесів. Тому, для підвищення ефективності цих інструментів запропоновано використання засобів паралельного виконання завдань та інтеграції зі сторонніми моніторинговими системами.

В цілому, можна зробити висновок про те, що застосування інструментів Python дозволяє отримати високу ефективність і покращити загальну продуктивність процесів автоматизації міграції доменів, за рахунок їх оптимізації та мінімізації ризиків під час міграції.

РОЗДІЛ 3
ПРОЄКТУВАННЯ АВТОМАТИЗОВАНОЇ МІГРАЦІЇ ДОМЕНІВ

3.1 Розроблення скриптів для автоматизації процесу міграції доменів

Розроблення Python-скриптів для автоматизації процесу міграції доменів включає етапи управління DNS-записами, резервного копіювання даних і перевірка коректності міграції, кожен з яких важливий для забезпечення безпеки, стабільності та ефективності процесу [80].

Управління DNS-записами. Завданнями цього етапу є автоматичне створення, оновлення та видалення DNS-записів (A, CNAME, MX тощо) для налаштування доменів на новому сервері. Реалізація цих завдань здійснюється за допомогою бібліотеки DNSPython, за допомогою якої розробляються функції, що автоматично зчитують існуючі DNS-записи зі старого сервера та налаштовують їх на новому.

Резервне копіювання даних. Головним завданням цього етапу є: перед початком перенесення DNS-записів зберегти резервну копію початкових налаштувань, щоб у разі помилок можна було повернутися до попередньої конфігурації. Реалізація цього завдання: у скрипті автоматизації додається функція резервного копіювання, яка зберігає всі налаштування у файл JSON, забезпечуючи їх швидке відновлення у разі потреби.

Перевірка міграції. На цьому етапі виконується автоматична перевірка коректності перенесення DNS-записів на новий сервер, а також тестування доступності домену та перевірка відповідності DNS-записів. Реалізація: за допомогою функцій перевірки у скрипті автоматизації здійснюється запит DNS-записів на новому сервері та порівняння їх з резервними даними.

Узагальнений опис етапів розробки скриптів для управління DNS-записами, резервного копіювання даних, перевірки міграції, представлений у таблиці 3.1.

Таблиця 3.1 – Етапи розробки Python-скриптів для автоматизації процесу міграції доменів

Етап	Завдання	Реалізація
Управління DNS-записами	Створення та налаштування DNS-записів на новому сервері	DNSPython для автоматичного налаштування записів
Резервне копіювання	Збереження початкових налаштувань для відновлення	Збереження у файл JSON
Перевірка міграції	Перевірка коректності міграції та відповідності DNS-записів	Порівняння нових записів із резервною копією

Діаграма на рисунку 3.1 описує процес автоматизації міграції DNS-записів.

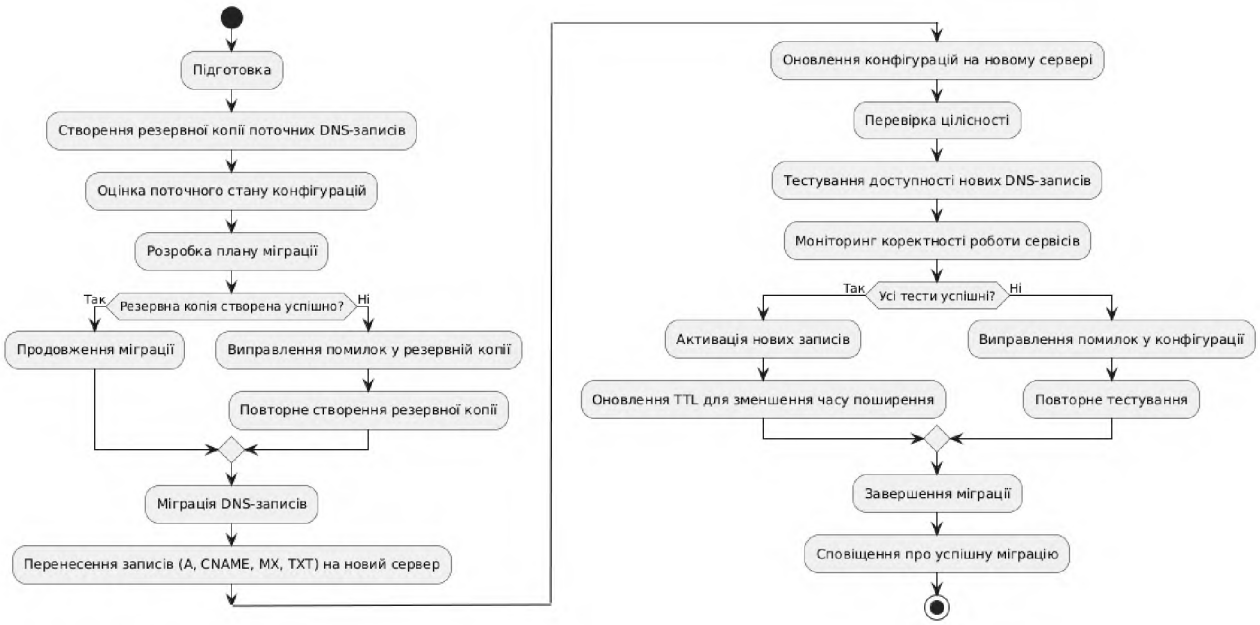

Рисунок 3.1 – Діаграма процесу автоматизації міграції DNS-записів

Дана діаграма структурує процес міграції DNS-записів, передбачає контрольні точки для перевірки та можливість повернення на попередні етапи для усунення ймовірних помилок. Етап підготовки включає створення резервної копії поточних DNS-записів, оцінку поточного стану конфігурації та планування міграції. Міграція включає перенесення DNS-записів на новий сервер і оновлення налаштувань. На етапі перевірки цілісності конфігурації відбувається тестування доступності та коректності роботи нових DNS-записів. Активація включає оновлення TTL і введення нових записів у дію. На етапі завершення передбачене сповіщення про завершення процесу.

У додатку Б представлений скрипт Python для автоматизації міграції DNS-записів, де реалізовані основні функції для зчитування DNS-записів, створення резервної копії, перенесення на новий сервер та перевірки коректності міграції.

Опис функціоналу скрипту:

a) функція backup_dns_records зчитує поточні DNS-записи з початкового сервера і створює резервну копію у форматі JSON, щоб зберегти всі необхідні налаштування та у разі потреби мати можливість повернутися до них;

b) функція migrate_dns_records переносить збережені DNS-записи на новий сервер: зчитує дані з резервної копії, створеної на попередньому етапі, і додає або змінює записи на новому DNS-сервері;

c) функція verify_migration перевіряє коректність міграції, запитуючи DNS-записи на новому сервері та порівнюючи їх з даними з резервної копії. У випадку успішної перевірки записується позитивне повідомлення у лог-файл, інакше робиться запис про помилки.

Таким чином, запропонований скрипт Python забезпечує автоматизацію основних етапів міграції DNS-записів, включаючи створення резервної копії, перенесення записів та перевірку їхньої коректності.

3.2 Обґрунтування оновлення параметру TTL

Окремо розглянемо TTL (Time To Live) – параметр, який визначає час життя DNS-запису в кеші DNS-серверів. Він вказується в секундах і повідомляє кешуючим серверам, як довго вони можуть зберігати цей запис перед тим, як звернутися до авторитетного DNS-сервера для отримання оновленої інформації.

Оновлення TTL є критично важливим під час міграції доменів, щоб мінімізувати простої та забезпечити швидке оновлення інформації на всіх DNS-серверах.

Коли вебклієнт запитує IP-адресу домену (наприклад, example.com), то DNS-сервер зберігає у кеші відповідний запис із заданим параметром TTL, і

протягом часу, визначеного TTL, обслуговує цього клієнта із кешу, не звертаючись до авторитетного сервера. Такий механізм дозволяє прискорити процес обслуговування запитів клієнта з боку DNS-серверу. Після закінчення часу, встановленого у TTL, кешуючий сервер видаляє запис із кешу, і для наступного запиту сервер звертається до авторитетного DNS-сервера за актуальною інформацією.

Оновлення TTL під час міграції доменів має важливе значення. Під час міграції DNS-записів важливо, щоб зміни (наприклад, нова IP-адреса) стали видимими для користувачів якомога швидше. Якщо TTL встановлений на тривалий час (наприклад, 86400 секунд або 24 години), користувачі можуть отримувати застарілу інформацію із кешу протягом цього періоду. Тому, під час міграції потрібна зміна TTL на менше значення (наприклад, 300 секунд або 5 хвилин), щоб забезпечити швидше оновлення кешу на всіх DNS-серверах. Після завершення міграції TTL можна повернути до звичайного значення, щоб зменшити навантаження на авторитетний сервер (оскільки короткий TTL збільшує кількість запитів до нього).

Оновлення значення TTL виконується наступним чином. Заздалегідь (зазвичай, за кілька годин до початку міграції) потрібно змінити TTL на менше значення, наприклад, встановити TTL = 300 (5 хвилин). Після перевірки успішності міграції повернути TTL до стандартного значення TTL = 86400 (24 години).

Приклад реалізації оновлення значення TTL на Python:

```
import dns.update
import dns.query

domain = "example.com"
dns_server = "192.168.1.1"

# Зміна TTL
def update_ttl(domain, ttl):
    update = dns.update.Update(domain)
    update.replace("www", ttl, "A", "192.168.1.100")
    dns.query.tcp(update, dns_server)
```

```
    print(f"TTL для {domain} оновлено до {ttl}
секунд.")

# Зменшення TTL перед міграцією
update_ttl(domain, 300)

# Повернення до стандартного TTL після міграції
update_ttl(domain, 86400)
```

Таким чином, оновлення TTL дозволяє ефективно керувати доступністю вебресурсів і уникати проблєм, пов'язаних із застарілими даними в кеші.

3.3 Інтеграція розроблених рішень у середовище корпоративної мережі

3.3.1 Впровадження скриптів автоматизації міграції

Для інтеграції Python-скриптів, призначених для автоматизації міграції доменів, у корпоративну мережу необхідно виконати низку налаштувань для забезпечення сумісності з існуючими системами та мережевими конфігураціями.

Перш за все, виконуються попередні налаштування – це перший крок для впровадження розроблених скриптів у робоче середовище, а саме: встановити Python та всі необхідні бібліотеки. Перед цим, потрібно переконатись, що Python встановлений у робочому середовищі, наприклад:

```
python --version
```

Після встановлення Python для інсталяції всіх необхідних бібліотек використовується пакетний менеджер pip:

```
pip install dnspython paramiko fabric
```

На цьому ж кроці виконується конфігурація мережевих прав доступу, а саме: потрібно дозволити скрипту автоматизації міграції доступ до DNS-серверів та SSH-з'єднань, щоб мережевий трафік між сервером і середовищем, де виконуватиметься скрипт, був дозволений.

Наступним кроком є налаштування середовища виконання: тут необхідно створити конфігураційний файл (наприклад, config.json), де вказуються усі

важливі параметри для міграції, зокрема IP-адреси старого та нового DNS-серверів, домени та типи записів. Приклад конфігураційного файлу config.json:

```
{
  "domain": "example.com",
  "old_dns_server": "192.168.1.1",
  "new_dns_server": "192.168.2.1",
  "log_file": "migration.log",
  "record_types": ["A", "CNAME", "MX", "TXT"]
}
```

Також, потрібно виконати налаштувати логування – зберігання записів про всі дії скрипту у лог-файлі, що дозволить відслідковувати процес міграції та, при потребі, швидко знаходити помилки.

Третім кроком є тестування та перевірка. Перед повноцінною міграцією рекомендується запустити скрипт автоматизації міграції у тестовому режимі для невеликого набору DNS-записів. Цей тестовий запуск потрібен, щоб виявити можливі помилки або неточності в налаштуваннях. Після завершення тестового запуску слід переконатись, що отримані результати відповідають очікуванням: це робиться ручною перевіркою DNS-записів, щоб упевнитись у правильності налаштування міграції.

Заключний, четвертий крок – реалізація повноцінної міграції, коли виконується запуск скрипту для міграції всіх DNS-записів у робочому середовищі. На цьому етапі для моніторингу процесу міграції використовуються завчасно налаштовані лог-файли.

3.3.2 Налаштування міграції доменів для різних інфраструктур

У контексті мережевих технологій та автоматизації DNS-систем, «інфраструктура» означає сукупність апаратних, програмних і мережевих ресурсів, які використовуються для забезпечення роботи IT-сервісів, зокрема, для керування доменами та DNS-записами.

Інфраструктура у контексті даної роботи визначає загальний обсяг та складність IT-системи, яка використовується для обслуговування доменів і DNS-записів. Залежно від масштабів інфраструктури вибираються інструменти, методи

автоматизації та рівень управління. Існуючі категорії інфраструктур можна класифікувати за їх масштабом: мала, середня і велика інфраструктура.

Мала інфраструктура включає невелику кількість серверів та DNS-записів, що обслуговують локальні або внутрішні потреби організації, як правило, використовується одне чи кілька доменних імен з кількома піддоменами. Типові характеристики малої інфраструктури: кількість серверів 1-5; кількість DNS-записів – до 50 записів (A, CNAME, MX тощо). Зазвичай, таку інфраструктуру використовують малі компанії, стартапи та локальні мережі в офісі. Наприклад, невелика IT-компанія з одним доменом example.com, який має A-записи для www, mail, ftp, один сервер електронної пошти та вебсервер. Для автоматизованого налаштування малої інфраструктури можна використовувати базові інструменти, такі як Python-скрипти, що виконуються локально на сервері без складних механізмів паралельного виконання.

Середня інфраструктура – це мережі середнього масштабу, що обслуговують більшу кількість серверів та піддоменів. Зазвичай вони розміщуються у хмарних середовищах для забезпечення масштабованості та доступності. Типові характеристики середньої інфраструктури: кількість серверів 5-50; DNS-записів: 50-500. Такою інфраструктурою володіють середні компанії, інтернет-магазини, регіональні офіси. Наприклад, інтернет-магазин, який має домен store.com із такими A-записами для піддоменів www, api, images, mail, кілька серверів для обробки запитів, зберігання зображень та бази даних. Для автоматизованого управління середньою інфраструктурою рекомендується розгортати скрипти автоматизації у хмарному середовищі (наприклад, AWS, Azure), використовувати інструменти моніторингу (наприклад, Prometheus, Grafana) для відстеження статусу DNS-записів.

Велика інфраструктура включає складні розподілені системи з великою кількістю серверів, піддоменів та DNS-записів. Зазвичай це глобальні корпорації чи онлайн-платформи, які обслуговують користувачів у різних регіонах. Характерні параметри великої інфраструктури: кількість серверів понад 50, DNS-записів понад 500. Великими інфраструктурами володіють, зазвичай, глобальні

корпорації, хмарні провайдери, стримінгові сервіси тощо. Наприклад, стримінгова платформа Netflix має домен netflix.com із тисячами піддоменів, A-записи для регіональних серверів us.netflix.com, eu.netflix.com, asia.netflix.com, MX-записи для корпоративної пошти, велику кількість серверів для доставки контенту. Автоматизоване налаштування великої інфраструктури здійснюється з використанням інструментів паралельного виконання (наприклад, Celery, multiprocessing у Python) для одночасної обробки великих обсягів даних, що інфегрувані з автоматизованими CI/CD процесами для швидкого оновлення конфігурацій. Розглянута класифікація допомагає вибрати оптимальні підходи до налаштування залежно від масштабів мережі. Основні відомості про існуючі категорії інфраструктур узагальнені у таблиці 3.2.

Таблиця 3.2 – Порівняння характеристик різних інфраструктур

Категорія	Кількість серверів	DNS-записи	Приклад
Мала інфраструктура	1-5	До 50	Офісна мережа або локальний бізнес.
Середня інфраструктура	5-50	50-500	Інтернет-магазин, регіональні офіси.
Велика інфраструктура	50+	500+	Глобальна корпорація, стримінгова платформа.

Інтеграція скриптів автоматизації міграції доменів у різні типи інфраструктур може потребувати додаткових налаштувань залежно від масштабів і специфіки мережі. Для великої інфраструктури пропонуються такі додаткові рекомендації з налаштування робочого середовища:

- використовувати можливості паралельного виконання, асинхронного або багатопотокового виконання, щоб прискорити обробку записів для великої кількості доменів;

- для відстеження стану міграції та доступності доменів доцільною є інтеграція зі сторонніми системами моніторингу, такими як Prometheus або Nagios;

- передбачити автоматичне створення резервних копій усіх конфігураційних файлів, щоб забезпечити можливість їх відновлення у разі помилок.

Варіанти налаштувань залежно від масштабу корпоративної мережі представлені у таблиці 3.3.

Таблиця 3.3 – Налаштування робочого середовища залежно від масштабів мережі

Тип інфраструктури	Пропозиції з налаштування
Мала інфраструктура	Використовувати базові налаштування без необхідності паралельного виконання. Запускати скрипт автоматизації локально або на одному сервері.
Середня інфраструктура	Розгорнути скрипт у хмарному середовищі для забезпечення доступності, використовувати інструменти моніторингу для відстеження статусу.
Велика інфраструктура	Використовувати інструменти паралельного виконання (наприклад, Celery або multiprocessing) для одночасної обробки великих обсягів записів.

Процес інтеграції скриптів автоматизації міграції доменів у корпоративну інфраструктуру відображений на діаграмі (рисунок 3.2).

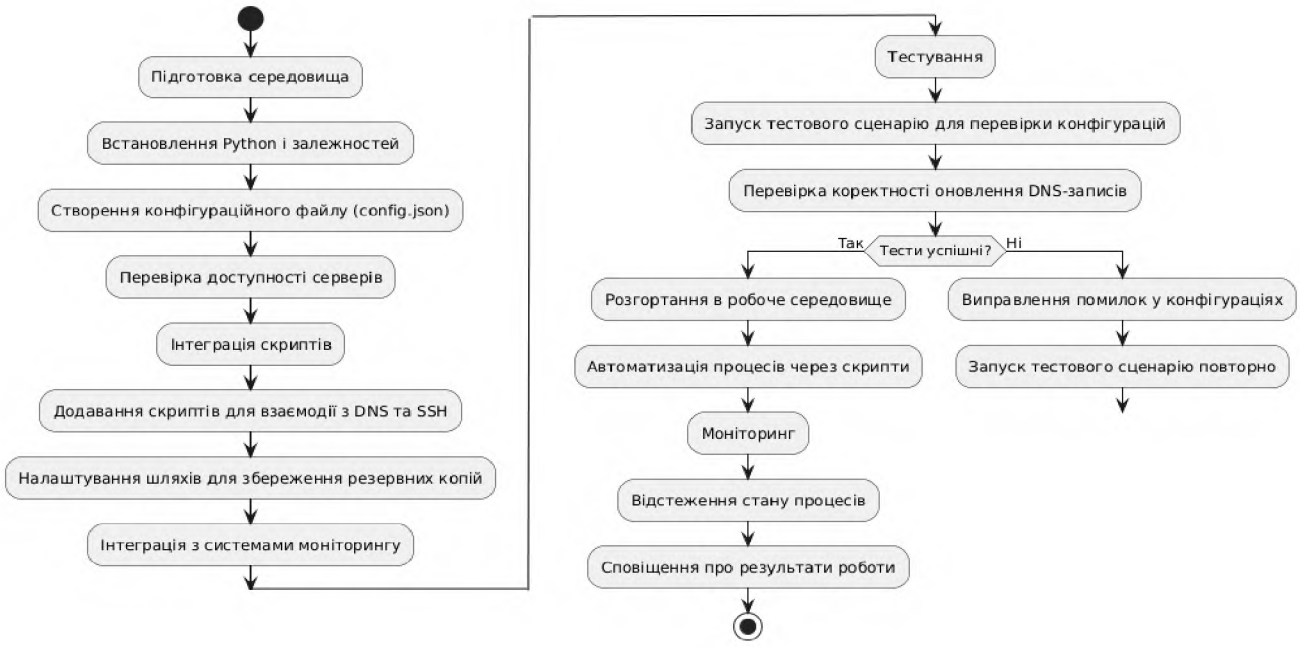

Рисунок 3.2 – Діаграма інтеграції скриптів автоматизації міграції доменів

Отже, впровадження розроблених скриптів у корпоративну мережу потребує належної підготовки та налаштувань для адаптації до різної інфраструктури. Забезпечення правильного логування, резервного копіювання та моніторингу значно підвищує надійність процесу та дозволяє уникнути можливих помилок під час міграції доменів.

3.4 Алгоритм автоматизації міграції доменів з використанням Python

Автоматизована міграція доменів включає кілька послідовних етапів, які забезпечують безпечне й ефективне перенесення DNS-записів між серверами. Діаграма загального алгоритму автоматизованої міграції доменів з використанням інструментів Python представлена на рисунку 3.3.

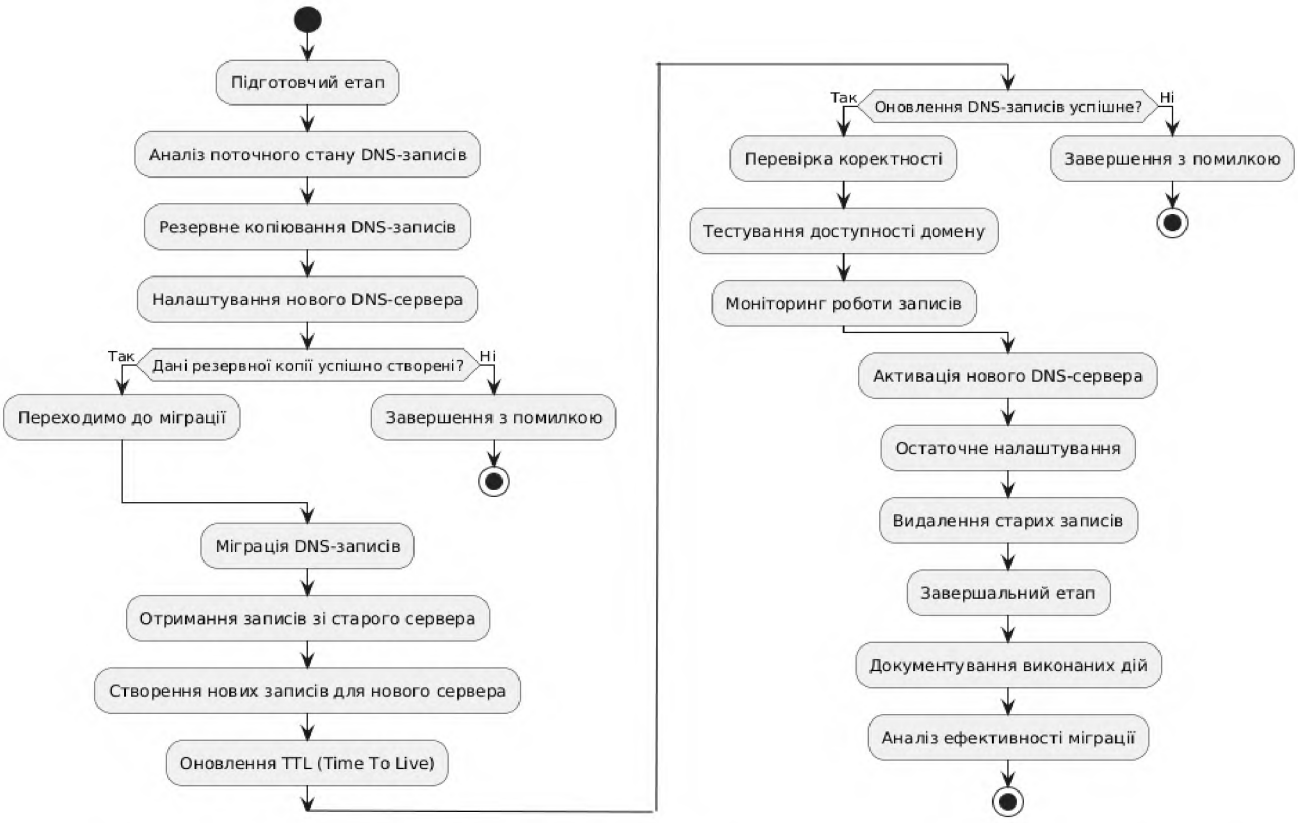

Рисунок 3.3 – Блок-схема алгоритму автоматизованої міграції доменів з використанням інструментів Python

Як видно з представленої діаграми (рисунок 3.3), процес автоматизованої міграції доменів включає декілька послідовних етапів.

На підготовчому етапі виконується аналіз існуючих DNS-записів та поточного стану інфраструктури, резервне копіювання даних для забезпечення можливості відновлення та налаштування нового DNS-сервера.

Після перевірки резервного копіювання, якщо дані успішно збережені, процес переходить до міграції; у разі помилки – відбувається завершення з повідомленням про невдачу.

Наступним етапом є міграція DNS-записів, де відбувається отримання існуючих записів зі старого сервера, створення нових записів на новому сервері та оновлення TTL для швидшого оновлення кешу.

Важливим етапом є перевірка оновлення записів: якщо оновлення успішне, виконується тестування доступності та моніторинг, у разі помилки процес завершується з логуванням проблєми.

На наступному етапі після успішної міграції активується новий сервер, а старі записи видаляються.

На завершальному етапі виконуються такі завдання: всі дії документуються для аналізу та виконується оцінка ефективності міграції.

Застосування Python та його бібліотек DNSPython, Paramiko і Fabric, спрощує реалізацію процесу автоматизованої міграції доменів.

Розглянемо поетапно практичну реалізацію цього алгоритму.

1. Підготовчий етап (етап 1).

Аналіз поточного стану:

- перевірка структури DNS-записів для домену (A, CNAME, MX тощо);
- ідентифікація активного DNS-сервера;
- збір інформації про всі піддомени та їхні записи.

Реалізація у Python:

```
import dns.resolver

domain = "example.com"
resolver = dns.resolver.Resolver()
```

```
records = resolver.resolve(domain, "A")
for record in records:
    print(f"A-запис для {domain}: {record.to_text()}")
```

Резервне копіювання: збереження поточних DNS-записів у вигляді файлу або бази даних для забезпечення можливості відновлення конфігурації у разі помилок.

Реалізація у Python:

```
with open("backup_dns.txt", "w") as file:
    for record in records:
        file.write(f"{record.to_text()}\n")
```

Налаштування нового DNS-сервера:

- підготовка IP-адреси нового сервера;
- забезпечення доступу до сервера через SSH (наприклад, із використанням Paramiko).

2. Міграція DNS-записів (етап 2).

Перенесення записів: створення нових записів на новому сервері із заміною старих IP-адрес на нові.

Реалізація у Python з використанням DNSPython:

```
import dns.update
import dns.query

domain = "example.com"
new_ip = "192.168.1.100"
dns_server = "192.168.1.1"

update = dns.update.Update(domain)
update.replace("www", 300, "A", new_ip)
dns.query.tcp(update, dns_server)
```

Оновлення TTL (Time To Live): зменшення TTL перед перенесенням для швидшого оновлення кешованих записів; після завершення процесу TTL повертається до нормального значення.

3. Перевірка коректності (етап 3).

Тестування доступності домену:

- використання інструментів для перевірки доступності піддоменів;
- перевірка відповідності нових IP-адрес.

Реалізація у Python:

```
import requests

url = "http://example.com"
response = requests.get(url)
if response.status_code == 200:
    print(f"{url} доступний.")
else:
    print(f"{url} недоступний.")
```

Моніторинг:

- відстеження стану записів після перенесення;
- виявлення потенційних проблєм через інструменти моніторингу, такі як Prometheus або Grafana.

4. Активація нового DNS-сервера (етап 4).

Остаточне налаштування:

- переведення домену на новий сервер;
- повідомлення всіх зацікавлених сторін (адміністраторів, користувачів) про завершення міграції.

Видалення старих записів: після перевірки коректної роботи нових записів видаляються дані з попереднього сервера.

5. Завершальний етап (етап 5).

Документування:

- запис дій, виконаних під час міграції, включаючи зміни та час виконання;
- збереження резервних копій.

Оцінка ефективності:

- аналіз часу виконання, помилок і впливу на роботу сервісів;
- використання результатів для покращення майбутніх міграцій.

Представлений алгоритм забезпечує системний підхід до автоматизованої міграції доменів. Застосування Python та його бібліотек дозволяє автоматизувати рутинні задачі, підвищити точність і знизити ризики, пов’язані з людськими помилками, що робить цей процес надійним і ефективним.

3.5 Перспективи розвитку автоматизації міграції доменів

Автоматизована міграція доменів демонструє свою ефективність та економічну доцільність у корпоративному середовищі. Подальший розвиток цього підходу передбачає вдосконалення існуючих рішень та інтеграцію нових функцій для масштабування системи, особливо для великих інфраструктур.

Пропозиції з подальшого вдосконалення автоматизованої міграції доменів за допомогою інструментів Python узагальнені у таблиці 3.4.

Таблиця 3.4 – Пропозиції з подальшого вдосконалення автоматизованої міграції доменів за допомогою інструментів Python

Пропозиція	Опис	Переваги
Паралельне виконання завдань	Використання AsyncIO для одночасної обробки записів	Збільшення швидкості, особливо у великих мережах
Інтеграція з хмарними сервісами	Використання AWS, Google Cloud для DNS-управління	Масштабованість, стабільність
Розширене логування та моніторинг	Інтеграція з Prometheus, Grafana для відстеження	Реальний час моніторингу, швидке виправлення

З метою підвищення продуктивності і функціональності Python-скриптів автоматизованої міграції доменів, пропонуються наступні вдосконалення:

a) запровадити паралельне виконання завдань автоматизації, що передбачає використання асинхронних бібліотек (наприклад, AsyncIO) або багатопотоковості для паралельного виконання операцій, яке забезпечить підвищення швидкості міграції, насамперед, у великих інфраструктурах, а також скорочення загального часу міграції;

b) використання хмарних рішень (наприклад, AWS Route 53 або Google Cloud DNS) для автоматичного керування DNS-записами, що забезпечить стабільне та швидке керування DNS-записами, а також можливість розширення корпоративної IT-інфраструктури без додаткових фізичних серверів;

c) впровадження розширеного логування та моніторингу процесів міграції за рахунок інтеграції зі сторонніми системами моніторингу, такими як Prometheus або Grafana, для збору даних про міграцію та стан DNS-записів, що забезпечить

можливість відстежувати стан міграції доменів у реальному часі, а також швидке виявлення та виправлення помилок.

Інтеграція нових функцій також дозволить підвищити функціональність механізму автоматизованої міграції доменів та зробити його більш ефективним для великих корпоративних мереж:

a) застосування контейнеризації (Docker) або кластеризації (Kubernetes) для виконання скриптів на великій кількості серверів одночасно, може допомогти вирішити проблєму обмеженого масштабування системи автоматизації міграції доменів для великих інфраструктур. Перевагою такого підходу є забезпечення масштабованості, можливість автоматично розподіляти навантаження між серверами;

b) інтеграція системи автоматизації міграції доменів з CI/CD-процесами, тобто автоматизація міграції у рамках DevOps-процесів для постійного управління доменами та оновлення DNS-записів. Це забезпечить можливість безперервної інтеграції та доставки, спрощення оновлення та підтримки DNS;

c) впровадження автоматичних перевірок на вразливості та функцій відновлення конфігурацій у разі виникнення помилок у процесі автоматизованої міграції доменів. Це забезпечить підвищений рівень безпеки та надійності, а також швидке відновлення після збоїв, у разі потреби.

Пропозиції щодо інтеграції нових функцій, масштабування системи автоматизації міграції доменів для великих інфраструктур за допомогою інструментів Python узагальнені у таблиці 3.6.

Таблиця 3.6 – Пропозиції щодо перспектив удосконалення автоматизації міграції доменів для великих інфраструктур

Напрямок розвитку	Пропозиція	Переваги
Масштабування для великих інфраструктур	Контейнеризація (Docker) або кластеризація (Kubernetes)	Масштабованість, ефективний розподіл
Інтеграція з CI/CD	Включення до DevOps-процесів	Постійне оновлення, полегшення підтримки
Автоматична перевірка безпеки	Перевірка на вразливості, функції відновлення	Підвищення надійності, захист від помилок

Таким чином, подальший розвиток автоматизованої міграції доменів може значно підвищити ефективність та надійність цього методу у корпоративних середовищах. Впровадження паралельного виконання завдань, інтеграція з хмарними сервісами, розширене логування та масштабування для великих мереж є перспективними напрямками вдосконалення автоматизації міграції доменів для забезпечення ефективного управління великими корпоративними інфраструктурами.

3.6 Техніко-економічне обґрунтування автоматизації міграції доменів

Очікуваними ефектами від впровадження автоматизації процесу міграції доменів є значне зменшення часу простою, підвищення продуктивності роботи та оптимізація витрат.

У цьому пункті проведено оцінку ефективності автоматизації, порівняння витрат на її впровадження з традиційними методами, а також складено кошторис і розраховано окупність інвестицій.

Розглянемо такі показники ефективності, як зменшення часу простою і підвищення продуктивності.

Як відомо, традиційні методи міграції доменів призводять до значного простою сервісів під час переналаштування DNS. Як було зазначено вище, автоматизація дозволяє знизити цей показник на 80-90%, що особливо важливо для компаній, залежних від безперервного доступу до мережевих ресурсів.

Також, автоматизація процесу міграції дозволяє виконувати завдання міграції без значного втручання та постійного контролю адміністратора, що звільняє ресурси IT-відділу компанії для інших важливих завдань і за рахунок цього підвищує загальну продуктивність та ефективність роботи IT- команди.

У таблиці 3.5 представлене порівняння показників ефективності процесу міграції доменів з використанням традиційних та автоматизованих методів.

Таблиця 3.5 – Порівняння показників ефективності процесу міграції доменів з використанням традиційних та автоматизованих методів

Показник	Традиційні методи	Автоматизований процес	Результат впровадження автоматизації
Час простою	до 30 хвилин	до 5 хвилин	Скорочення на 83%
Час, витрачений на міграцію	2-3 години	15-30 хвилин	Скорочення на 75-85%
Залучення адміністратора	Інтенсивне	Мінімальне	Підвищення продуктивності

Таким чином, автоматизація процесу міграції доменів дозволяє помітно скоротити час виконання завдань, знизити ризик помилок та зменшити час простою, що сприяє підвищенню загальної продуктивності IT-відділу та ефективнішому використанню робочих ресурсів.

Порівняємо тепер розмір витрат на автоматизацію із затратами на міграцію доменів традиційними методами.

У таблиці 3.6 проведене порівняння витрат на міграцію доменів з використанням традиційних методів та інструментів автоматизації [81, 82].

Таблиця 3.6 – Порівняння витрат на міграцію доменів

Вид витрат	Традиційні методи	Автоматизований процес
Робочі години адміністратора	3 години/міграція	0,5 години/міграція
Вартість робочої години	$40	$40
Загальна вартість міграції одного домену	$120	$20
Витрати на розробку та підтримку скриптів	Немає	$500 (одноразові витрати)

При використанні традиційних методів витрати робочого часу, пов’язані з міграцією доменів, включають більше робочих годин, що пов’язане із залученням адміністратора для ручного налаштування DNS-записів. Автоматизація дозволяє суттєво зменшити ці витрати, що призводить до економії на робочій силі та зниження ризиків фінансових втрат через тривалий простій.

Розглянемо кошторис витрат на автоматизацію, і виконаємо розрахунок окупності інвестицій.

Одноразові витрати (оцінка):

- розробка Python-скриптів для автоматизації – $500;

- навчання персоналу для роботи з автоматизованими скриптами – $200.

Всього: $700.

Постійні витрати:

- супровід і підтримка автоматизованих процесів – $100 на місяць.

Річні постійні витрати: $1200.

Наведені дані дозволяють виконати розрахунок окупності інвестицій у автоматизацію міграції доменів (ROI). Для розрахунку ROI використаємо наступну формулу:

$$ROI = \frac{\text{Річна економія} - \text{Одноразові витрати}}{\text{Одноразові витрати} + \text{Річні витрати}} \times 100\% \quad (1)$$

Завдяки автоматизації можна заощадити $100 на кожній міграції (традиційні витрати $120 – автоматизовані витрати $20) (таблиця 3.5). Тому, якщо за рік відбуватиметься міграція 20 доменів, то річна економія витрат становитиме $2000.

Підставивши вихідні дані до формули (1), обчислимо:

$$ROI = \frac{\$2000 - \$700}{\$700 + \$1200} \times 100\% = \frac{\$1300}{\$1900} \times 100\% = 68{,}42\%$$

Отримане позитивне значення окупності інвестицій, рівне 68, 42% свідчить про достатньо високий рівень рентабельності інвестицій у автоматизацію процесу міграції доменів. Це означає, що кошти, інвестовані у автоматизацію міграції доменів, досить швидко повернуться за рахунок зменшення витрат та економії ресурсів.

Для розрахунку терміну окупності вкладених коштів використаємо наступну формулу:

$$T = \frac{\text{Початкові витрати}}{\text{Річний прибуток}} \quad (2)$$

Підставивши вихідні дані до формули (2), отримаємо:

$$T = \frac{\$700}{\$2000} = 0{,}35 \text{ р.} = 4{,}2 \text{ міс.}$$

Розраховане значення терміну окупності показує, що впровадження автоматизації міграції доменів з використанням інструментів Python досить швидко окупається за рахунок економії на витратах і підвищення продуктивності роботи персоналу. Термін окупності інвестицій у автоматизацію міграції доменів, згідно виконаних розрахунків, становить 4,2 місяці.

Таким чином, проведений аналіз підтверджує доцільність використання інструментів Python для автоматизації міграції доменів у корпоративному середовищі. Значна економія витрат і швидка окупність інвестицій демонструють переваги автоматизації міграції доменів в корпоративному середовищі

Висновки до розділу 3

У цьому розділі було розглянуто розробку Python-скриптів для автоматизації процесу міграції доменів, а також можливості інтеграції цих рішень у корпоративне середовище.

Оцінка перспектив розвитку автоматизованих рішень дозволила сформулювати низку рекомендацій для покращення процесу міграції, а саме:

- паралельне виконання завдань – використання асинхронних або багатопотокових підходів підвищує швидкість виконання міграції для великих інфраструктур та суттєво знижує загальний час обробки;

- інтеграція з хмарними сервісами – використання хмарних платформ для керування DNS-записами (AWS Route 53, Google Cloud DNS) забезпечує більшу стабільність і можливість масштабування, що особливо важливо для середніх та великих корпоративних мереж;

- розширене логування та моніторинг – інтеграція з системами моніторингу, такими як Prometheus або Grafana, дозволяє контролювати стан

міграції у реальному часі, швидко виявляти помилки та відслідковувати загальну ефективність процесу;

- масштабування для великих інфраструктур – застосування технологій контейнеризації (Docker) або кластеризації (Kubernetes) дозволяє розподіляти навантаження між серверами, що підвищує надійність системи та забезпечує гнучкість;

інтеграція з CI/CD – автоматизація міграції у рамках CI/CD-процесів спрощує управління DNS-записами та дозволяє оперативно реагувати на зміни у мережевій інфраструктурі.

Запропоновані покращення є виправданими з точки зору продуктивності, надійності та гнучкості. Використання паралельного виконання завдань та інтеграція з хмарними сервісами забезпечують ефективне виконання міграції навіть для великих доменних структур. Розширене логування та моніторинг допомагають швидко ідентифікувати та вирішувати проблєми, що знижує ризики простоїв. Контейнеризація і кластеризація підвищують масштабованість, а інтеграція з CI/CD-процесами дозволяє впровадити безперервне управління DNS.

Проведене техніко-економічне обґрунтування виявило високу ефективність впровадження автоматизації міграції доменів, яка забезпечує значне зменшення часу простою, підвищення продуктивності та оптимізацію витрат на управління DNS-записами.

ВИСНОВКИ

За результатами виконаної роботи можна зробити висновки, що охоплюють основні результати, отримані у кожному розділі, та підтвердити досягнення поставленої мети.

Метою дослідження було розробити ефективний підхід до автоматизації міграції доменів у корпоративних мережах, що дозволить скоротити витрати, зменшити ризики та забезпечити стабільність процесу. З урахуванням отриманих результатів і запропонованих інновацій, можна стверджувати, що мету дослідження досягнуто. Запропонована автоматизації міграції доменів на основі використання скриптів Python, здатна забезпечити високу ефективність, забезпечуючи значне скорочення часу міграції, зниження кількості помилок та оптимізацію робочих ресурсів.

Розділ 1 було присвячено аналізу існуючих методів автоматизації міграції доменів. Було виявлено ключові переваги Python як основного інструмента для автоматизації: гнучкість, масштабованість і широкі можливості для інтеграції з корпоративним середовищем. Дослідження підтвердило, що автоматизація на основі Python є оптимальним підходом для ефективного перенесення DNS-записів, що забезпечує надійність та економію часу.

У розділі 2 проведено оцінку ефективності автоматизованого підходу на основі зібраних кількісних і якісних показників. Встановлено, що автоматизація значно скорочує час міграції доменів, знижує ризик помилок і зменшує час простою доменів. Підтверджено економічну доцільність автоматизації, що демонструє суттєву економію робочих ресурсів у порівнянні з традиційними методами міграції доменів.

У розділі 3 розглянуто технічні аспекти розробки та впровадження Python-скриптів для автоматизації міграції доменів. Визначено конкретні етапи розробки скриптів, впровадження рішення з резервного копіювання та перевірки коректності перенесення записів, що забезпечує безпеку процесу. Проведене техніко-економічне обґрунтування показало швидку окупність інвестицій в

автоматизацію процесу міграції доменів. У межах цього розділу також були сформульовані перспективи розвитку автоматизації міграції доменів, включаючи рекомендації щодо масштабування та інтеграції з хмарними та DevOps інструментами.

Практичний результат роботи полягає у проєктуванні рішення для автоматизованої міграції доменів, яке може бути інтегроване у різні корпоративні середовища та адаптоване до їх специфічних вимог. Дане рішення має практичну цінність для підприємств, які потребують забезпечення безперервної роботи своїх мережевих ресурсів і мінімізації витрат на адміністрування корпоративних доменів.

Елементи наукової новизни виявляються у розробці нових підходів до автоматизації міграції доменів з використанням Python, що може бути основою для подальших удосконалень у цьому напрямку. Запропоновані у роботі рекомендації з масштабування та вдосконалення автоматизованих рішень міграції доменів можуть бути використані для подальшого розвитку у сфері управління інфраструктурою корпоративних мереж.