DOI: https://doi.org/10.36910/6775-2524-0560-2026-63-16
УДК 004.72
Багнюк Наталія Володимирівна, к.т.н., доцент
https://orcid.org/0000-0002-7120-5455
Мельник Катерина Вікторівна, к.т.н., доцент
https://orcid.org/0000-0002-9991-582X
Бортник Катерина Яківна, к.т.н., доцент
https://orcid.org/0000-0001-5282-099X
Міскевич Оксана Іванівна, старший викладач
https://orcid.org/0000-0002-5009-2391
Карасійчук Дмитро Ярославович, студент
Луцький національний технічний університет, м.Луцьк, Україна

АВТОМАТИЗАЦІЯ КЕРУВАННЯ КОНФІГУРАЦІЯМИ КОМП'ЮТЕРНИХ СИСТЕМ ЗА ДОПОМОГОЮ ANSIBLE ТА ПІДХОДУ «INFRASTRUCTURE AS CODE»

Багнюк Н. В., Мельник К. В., Бортник К. Я., Міскевич О. І., Карасійчук Д. Я. Автоматизація керування конфігураціями комп'ютерних систем за допомогою Ansible та підходу «Infrastructure as code». У статті розглянуто застосування інструменту Ansible та концепції Infrastructure as Code (IaC) для автоматизації керування конфігураціями комп'ютерних систем у сучасних IT-інфраструктурах підприємств. Особливу увагу приділено технічним аспектам впровадження системи автоматизації, включаючи створення playbook'ів, ролей та інвентаря серверів. Проаналізовано переваги декларативного підходу до управління інфраструктурою, що забезпечує відтворюваність конфігурацій, версійний контроль та зменшення людських помилок. Наведено практичний приклад впровадження Ansible для автоматизації розгортання веб-серверів, налаштування безпеки та моніторингу систем. Розглянуто типові проблеми, що виникають під час міграції від ручного управління до автоматизованого, включаючи питання ідемпотентності операцій, управління секретами та масштабування інфраструктури. Запропоновано ефективні практики організації коду Infrastructure as Code, включаючи структуру проєкту, використання змінних та шаблонів Jinja2, інтеграцію з системами контролю версій. Результати дослідження показують значне скорочення часу на розгортання та налаштування серверів, підвищення стабільності системи та можливість швидкого відновлення після збоїв.
Ключові слова: Ansible, Infrastructure as Code, автоматизація, конфігураційне управління, DevOps, playbook, ідемпотентність.

Bahniuk N., Melnyk K., Bortnyk K., Miskevych O., Karasiichuk D. Automation of computer systems configuration management using ansible and «the infrastructure as code» approach. The article discusses the application of the Ansible tool and the Infrastructure as Code (IaC) concept for automating configuration management of computer systems in modern enterprise IT infrastructures. Particular attention is paid to the technical aspects of implementing an automation system, including the creation of playbooks, roles, and server inventory. The advantages of a declarative approach to infrastructure management are analyzed, which ensures configuration reproducibility, version control, and reduction of human errors. A practical example of implementing Ansible for automating web server deployment, security configuration, and system monitoring is presented. Typical problems that arise during migration from manual to automated management are considered, including issues of operation idempotency, secrets management, and infrastructure scaling. Effective practices for organizing Infrastructure as Code are proposed, including project structure, use of variables and Jinja2 templates, and integration with version control systems. The research results show a significant reduction in server deployment and configuration time, increased system stability, and the ability to quickly recover from failures.
Keywords: Ansible, Infrastructure as Code, automation, configuration management, DevOps, playbook, idempotency.

Постановка наукової проблеми. У зв'язку з тим, що сучасні підприємства стикаються з необхідністю управління великою кількістю серверів та мережевих пристроїв, що в свою чергу робить ручне адміністрування неефективним та схильним до помилок. Традиційні підходи до управління IT-інфраструктурою, в ситуації, коли адміністратори вручну налаштовують кожен сервер, призводять до проблем консистентності, відсутності документації змін та складності масштабування. Концепція Infrastructure as Code (IaC) пропонує вирішення цих проблем через опис інфраструктури у вигляді коду, який дозволяє застосовувати практики розробки програмного забезпечення до управління інфраструктурою.

Ansible є одним із провідних інструментів автоматизації, який використовує агентлесну архітектуру та декларативний синтаксис на базі YAML. На відміну від інших рішень, Ansible не вимагає встановлення спеціальних агентів на керованих вузлах, використовуючи стандартні протоколи SSH та WinRM. Однак впровадження Ansible та підходу IaC потребує ретельного планування архітектури, розуміння принципів ідемпотентності операцій та організації коду для забезпечення масштабованості та підтримуваності рішення.

Аналіз останніх досліджень і публікацій. Останні дослідження у сфері Infrastructure as Code зосереджені на порівнянні різних інструментів автоматизації. У роботі [1] проведено комплексний

аналіз Ansible, Puppet, Chef та Salt, де виявлено, що Ansible має найнижчий поріг входу завдяки відсутності необхідності у спеціалізованих агентах та використанню простого синтаксису YAML. Дослідження показують, що Ansible особливо ефективний для середніх та великих інфраструктур, де важлива швидкість впровадження та простота підтримки.

Актуальним напрямком є інтеграція Ansible з контейнерними технологіями [2]. Дослідження демонструють ефективність використання Ansible для оркестрації Docker-контейнерів та управління Kubernetes-кластерами, що дозволяє створювати гібридні інфраструктури з традиційними віртуальними машинами та контейнерами. Важливим аспектом є автоматизація процесів CI/CD через інтеграцію Ansible з Jenkins, GitLab CI та іншими системами безперервної інтеграції.

У сфері безпеки [3] активно досліджуються методи управління секретами в Ansible через Ansible Vault, HashiCorp Vault та інші системи управління секретами. Розроблено підходи до реалізації принципів Zero Trust у автоматизованих інфраструктурах, включаючи динамічне управління доступом та автоматичну ротацію облікових даних. Особливу увагу приділено аудиту змін та забезпеченню compliance через автоматизовані перевірки конфігурацій.

Дослідження в області оптимізації продуктивності [4] показують ефективність використання стратегій паралельного виконання завдань, кешування фактів та оптимізації SSH-з'єднань. Для великих інфраструктур рекомендується використання Ansible Tower (AWX) для централізованого управління, планування завдань та RBAC (Role-Based Access Control).

Виділення невирішених раніше частин загальної проблеми. Попри широке впровадження Ansible та концепції IaC, залишаються невирішеними питання стандартизації організації великих проєктів, ефективного управління складними залежностями між ролями, оптимізації часу виконання playbook'ів для інфраструктур з тисячами вузлів. Недостатньо досліджені підходи до тестування Infrastructure as Code, включаючи модульне тестування ролей, інтеграційне тестування та валідацію конфігурацій перед застосуванням у продуктивному середовищі.

Мета дослідження: основною метою дослідження є розробка та впровадження ефективної системи автоматизації керування конфігураціями комп'ютерних систем з використанням Ansible та підходу Infrastructure as Code, що включає створення структури проєкту, розробку повторно використовуваних ролей, налаштування безпечного управління секретами та інтеграцію з системами моніторингу та оповіщення.

Завдання дослідження. Для досягнення поставленої мети визначено наступні завдання: проаналізувати архітектуру існуючої IT-інфраструктури та визначити об'єкти автоматизації, розробити структуру Ansible-проєкту з використанням ролей та групових змінних, створити playbook'и для автоматизації типових завдань адміністрування, впровадити систему управління секретами через Ansible Vault, протестувати розроблене рішення на тестовому та продуктивному середовищі.

Основна частина дослідження. З метою забезпечення експериментальної перевірки підходу Infrastructure as Code у роботі сформовано ізольоване тестове середовище, яке імітує реальну багаторівневу серверну інфраструктуру (табл. 1).

Таблиця 1. Структура тестового середовища

Контейнер	Роль	SSH-порт
server-web1	Веб-сервер 1 (Nginx)	2210
server-web2	Веб-сервер 2 (Nginx)	2211
server-db1	База даних 1 (PostgreSQL)	2220
server-db2	База даних 2 (PostgreSQL)	2221
server-mon	Моніторинг	2230

В якості технологічної основи використано платформу контейнеризації Docker, що дозволяє швидко створювати, масштабувати та ізолювати середовища виконання без необхідності розгортання повноцінних віртуальних машин. Контейнери функціонують на базі операційної системи Ubuntu 22.04, яка забезпечує стабільне та передбачуване середовище для встановлення серверного програмного забезпечення (рис.1).

Створене тестове середовище відображає типовий сценарій організації сучасної веб-інфраструктури, де функціональні ролі розподілені між окремими вузлами. У межах дослідження реалізовано п'ять контейнерів, які виконують ролі веб-серверів, серверів баз даних та вузла

моніторингу. Такий підхід дозволяє відтворити принципи мікросервісної або багаторівневої архітектури, де кожен компонент системи ізольований та виконує чітко визначені функції.

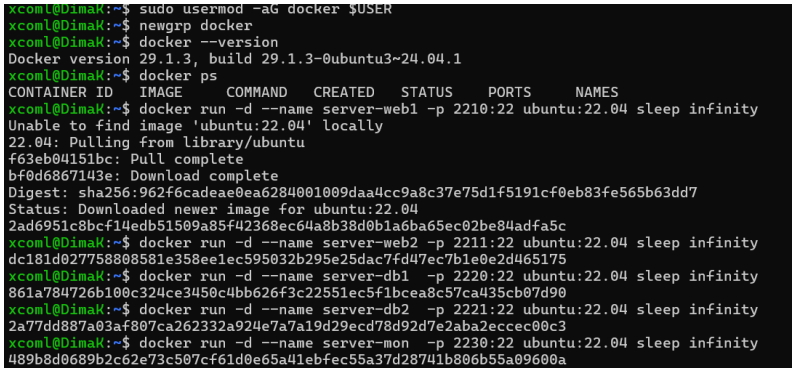

Рис. 1. Створення структури тестового середовища

Запуск контейнерів здійснюється у фоновому режимі з використанням механізму пробросу портів, що забезпечує доступ до кожного вузла через унікальний SSH-порт на локальній машині. Це дозволяє імітувати розподілену мережеву інфраструктуру навіть у межах одного фізичного хоста.

Особливістю такого підходу є використання команди запуску контейнера з нескінченним процесом очікування, що гарантує безперервну роботу контейнера після ініціалізації. Кожному контейнеру присвоюється унікальне ім'я та порт, що дозволяє однозначно ідентифікувати його у системі та забезпечити коректну маршрутизацію мережевих запитів. В результаті формується логічна структура тестового середовища, яка включає два веб-сервери для обробки HTTP-запитів, два сервери баз даних для зберігання інформації та окремий вузол, що може використовуватися для моніторингу або допоміжних сервісів.

Важливим етапом підготовки середовища є організація безпечного віддаленого доступу до кожного контейнера. Для цього у всіх вузлах встановлюється сервер віддаленого доступу, що реалізує протокол SSH, після чого виконується базова конфігурація автентифікації, включаючи встановлення облікових даних адміністратора та налаштування дозволу на підключення. Однак використання парольної автентифікації у системах автоматизації є неефективним та потенційно небезпечним, тому наступним кроком є впровадження механізму безпарольного доступу на основі криптографічних ключів.

З цією метою генерується пара ключів типу Ed25519, яка характеризується високим рівнем криптографічної стійкості та ефективністю (рис. 2).

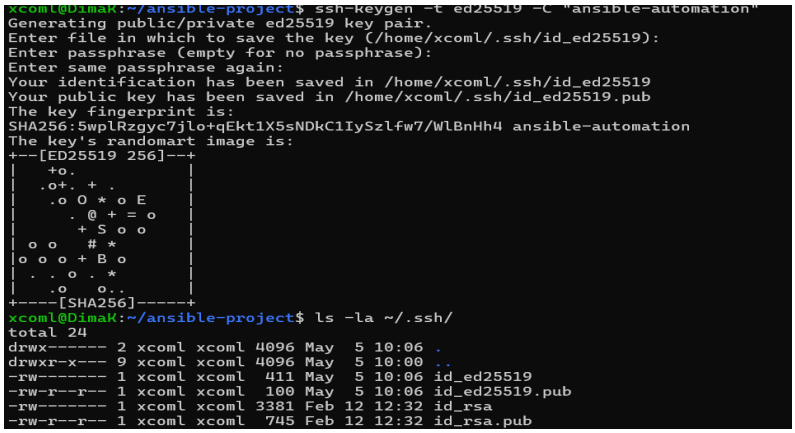

Рис. 2. Генерування SSH-ключів

Відкритий ключ розповсюджується на всі керовані вузли, що дозволяє організувати довірені SSH-з'єднання без необхідності введення пароля при кожному підключенні. Такий підхід є критично важливим для коректної роботи Ansible, оскільки цей інструмент використовує SSH як основний механізм комунікації з віддаленими вузлами.

Після формування тестового середовища наступним етапом є організація структури проєкту автоматизації, що реалізується засобами Ansible (рис. 3). Раціональна побудова структури проєкту відіграє ключову роль у забезпеченні масштабованості, підтримуваності та повторного використання конфігурацій, особливо в умовах складних розподілених інфраструктур. У даному дослідженні застосовано підхід, що відповідає рекомендаціям офіційної документації Ansible та передбачає розділення логіки конфігурації на окремі компоненти, зокрема інвентар, ролі, змінні та playbook-и.

```
ansible-project/
├── ansible.cfg
├── inventory/
│   └── docker/
│       ├── hosts
│       └── group_vars/
│           └── databases/vars.yml
├── roles/
│   ├── common/
│   ├── webserver/
│   └── database/
└── playbooks/
    └── site.yml
```

Рис. 3. Структура Ansible проєкту

Центральним елементом структури є інвентар, який містить опис усіх керованих вузлів та їх логічне групування відповідно до функціонального призначення (рис. 4). У межах розробленої інфраструктури вузли поділяються на три основні групи: веб-сервери, сервери баз даних та вузол моніторингу. Для кожного вузла визначаються параметри підключення, включаючи адресу хоста, порт SSH-з'єднання та облікові дані користувача. Така організація дозволяє ефективно застосовувати різні конфігурації до відповідних груп серверів, що є особливо важливим у випадках, коли різні компоненти системи мають відмінні вимоги до програмного забезпечення та налаштувань.

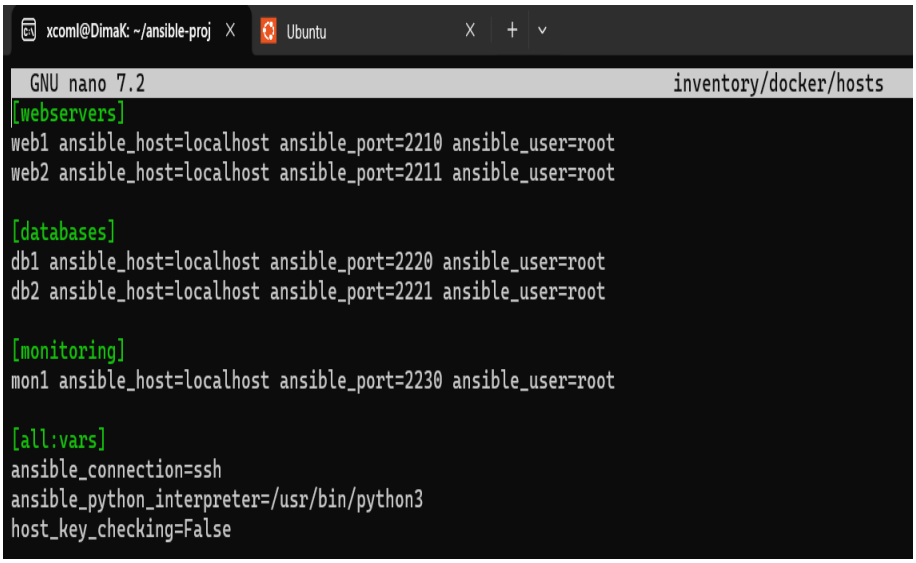

Рис. 4. Файл інвентарю

Окрему роль відіграє механізм змінних, який реалізується через структуру group_vars. Це дозволяє визначати параметри конфігурації для цілих груп вузлів, зокрема налаштування баз даних, облікові дані або інші параметри, що можуть змінюватися залежно від середовища. Такий підхід підвищує гнучкість конфігурації та спрощує її модифікацію без необхідності зміни основного коду playbook-ів.

Ключовим компонентом архітектури проєкту є ролі, які забезпечують модульність та логічне групування задач. Кожна роль інкапсулює певний набір функцій, що відповідає конкретному аспекту конфігурації системи. Це дозволяє розділити складний процес налаштування на незалежні частини, які можуть бути використані повторно у різних проєктах або середовищах. Крім того, ролі сприяють кращій структуризації коду, що значно полегшує його супровід та розширення.

Playbook-и виступають у ролі сценаріїв виконання, які визначають послідовність застосування ролей до відповідних груп вузлів. Вони описують бажаний стан інфраструктури у декларативній формі, що дозволяє Ansible самостійно визначати необхідні кроки для досягнення цього стану. Завдяки цьому значно зменшується складність управління конфігураціями та мінімізується ризик виникнення помилок.

Для перевірки коректності налаштування інвентарю та доступності вузлів використовується базовий механізм тестування з'єднання, який дозволяє виконати просту операцію на кожному вузлі та отримати відповідь про її успішність (рис. 5). Отримання позитивного результату з усіх вузлів свідчить про правильність налаштування мережевої взаємодії, що є необхідною умовою для подальшого виконання автоматизованих сценаріїв.

```
xcoml@DimaK:~/ansible-project$ ansible all -i inventory/docker/hosts -m ping
web2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
web1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
mon1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
xcoml@DimaK:~/ansible-project$
```

Рис. 5. Перевірка доступності всіх вузлів

Наступним етапом реалізації автоматизованого управління інфраструктурою є впровадження рольового підходу до організації конфігурацій [5]. У середовищі Ansible ролі виступають як основні структурні одиниці, що дозволяють групувати пов'язані задачі, змінні, шаблони та обробники у логічно завершені модулі. Такий підхід значно підвищує рівень абстракції, забезпечує повторне використання коду та сприяє стандартизації процесів налаштування.

У межах дослідження реалізовано декілька ролей, кожна з яких відповідає окремому функціональному аспекту інфраструктури (табл. 2).

Таблиця 2. Ролі Ansible та їхні функції

Роль	Задачі	Пакети/Сервіси
common	Базове налаштування всіх вузлів	curl, sudo, UFW
webserver	Встановлення та конфігурація Nginx	nginx
database	PostgreSQL, БД, користувач	postgresql

Базова роль common призначена для початкової підготовки всіх вузлів системи (рис. 6). Вона включає оновлення пакетного менеджера, встановлення необхідного програмного забезпечення та створення службового користувача, який використовується для подальшої автоматизації. Це дозволяє сформувати уніфіковане середовище, яке є однаковим для всіх компонентів системи.

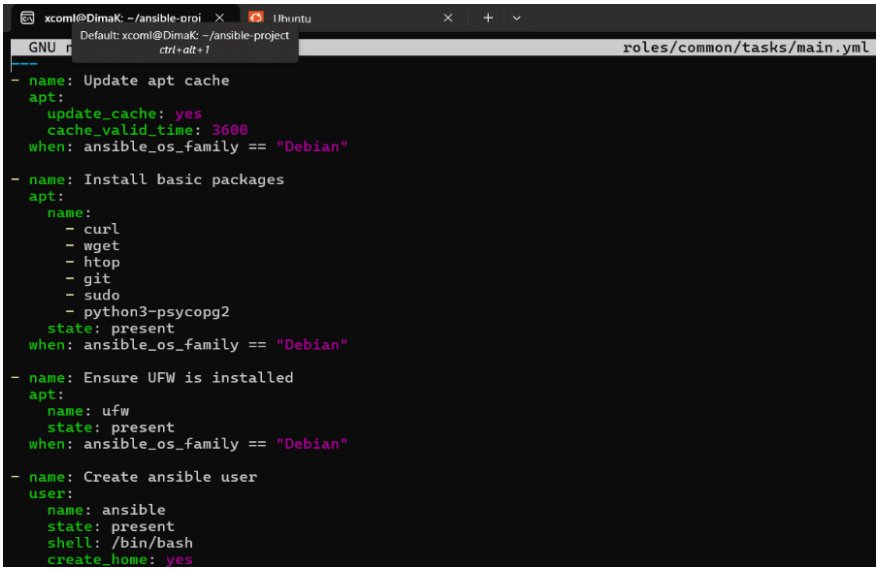

Рис. 6 Роль common

Роль, що відповідає за налаштування веб-серверів, webserver, передбачає встановлення та конфігурацію Nginx (рис. 7).

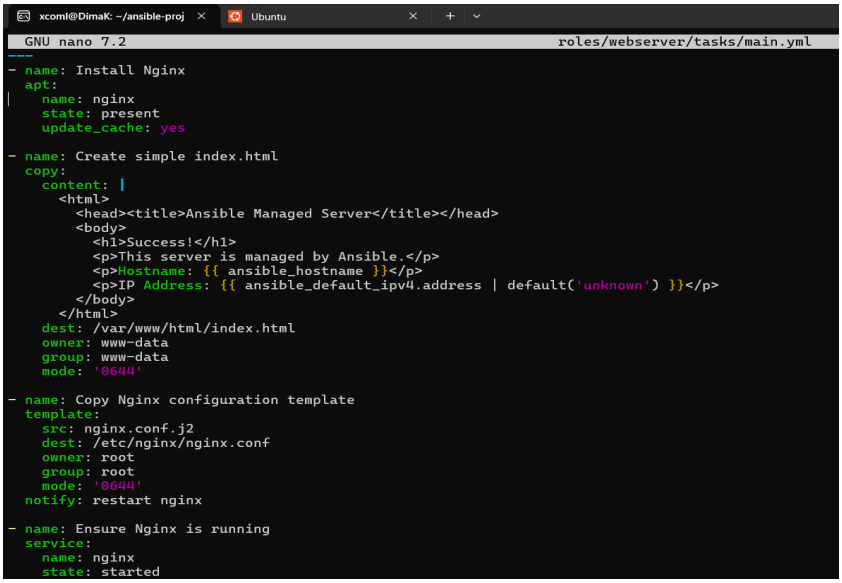

Рис. 7. Роль webserver

Важливим аспектом є використання шаблонів, які дозволяють динамічно формувати конфігураційні файли залежно від параметрів конкретного вузла. Це забезпечує гнучкість та адаптивність системи до змін у середовищі виконання. Крім того, реалізовано механізм обробників, який дозволяє перезапускати сервіс лише у випадку зміни конфігурації, що сприяє оптимізації ресурсів та підвищенню стабільності системи.

Окрема роль призначена для розгортання системи керування базами даних PostgreSQL – database (рис. 8). Вона включає встановлення необхідних пакетів, запуск служби бази даних, створення бази даних та налаштування користувачів із відповідними правами доступу. У процесі реалізації враховано специфіку різних версій програмного забезпечення, що потребує динамічного визначення шляхів до конфігураційних файлів. Це дозволяє забезпечити універсальність ролі та її коректну роботу в різних середовищах.

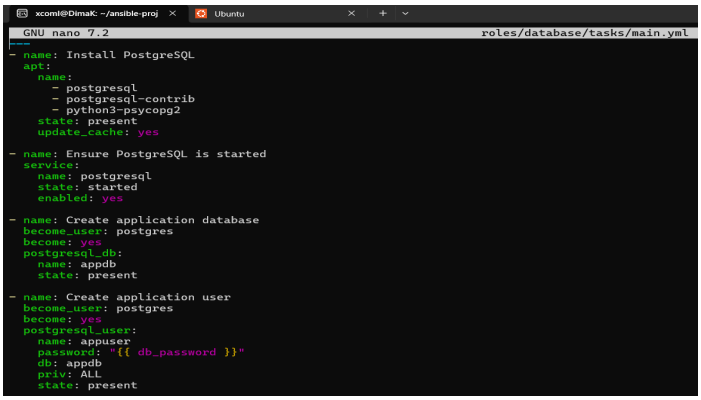

Рис. 8. Роль database

Завершальним етапом практичної реалізації автоматизованого управління інфраструктурою є виконання сценаріїв конфігурації, які в середовищі Ansible представлені у вигляді playbook-файлів. Саме playbook виступає центральним механізмом, що визначає узгоджену послідовність застосування ролей до відповідних груп вузлів, описаних в інвентарі [6]. У даному дослідженні використовується головний сценарій, який об'єднує всі попередньо розроблені ролі та забезпечує комплексне налаштування тестового середовища відповідно до заданих вимог.

Безпосередній запуск процесу конфігурації здійснюється за допомогою стандартної команди виконання playbook – ansible-playbook -i inventory/docker/hosts playbooks/site.yml, яка ініціює підключення до всіх вузлів та послідовне виконання визначених задач відповідно до їх ролей (рис. 9). У ході виконання Ansible аналізує поточний стан кожного вузла та визначає необхідність внесення змін, що дозволяє уникнути надлишкових операцій і забезпечує ефективність процесу.

Рис. 9. Запуск playbook

Важливим етапом перед фактичним застосуванням конфігурацій є проведення попередньої перевірки у так званому режимі імітації, або dry-run (рис. 10). Для цього використовується спеціальний параметр виконання, який дозволяє змоделювати процес застосування playbook без внесення реальних змін у систему. У такому режимі система відображає, які саме задачі були б виконані та які зміни

відбулися б у випадку повноцінного запуску. Це дає змогу виявити потенційні помилки, конфлікти конфігурацій або некоректні залежності ще до моменту їх фактичного впровадження.

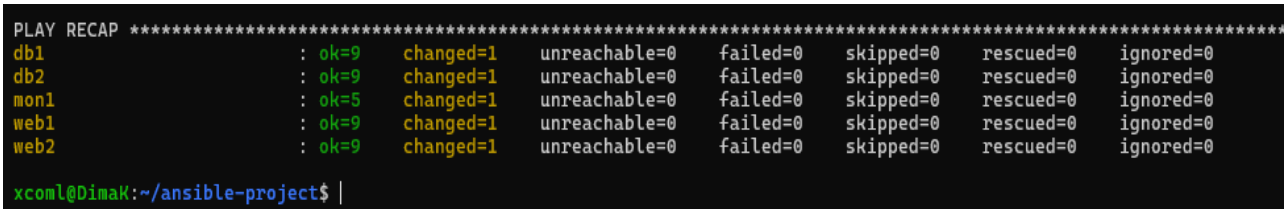

Рис. 10. Перевірка без реальних змін (dry-run):

Для оцінювання коректності роботи автоматизованих сценаріїв та підтвердження принципів ідемпотентності проведено порівняльний аналіз результатів виконання playbook у двох режимах: стандартному режимі застосування конфігурацій та режимі попередньої симуляції dry-run (табл. 3).

Таблиця 3. Порівняльні результати виконання playbook у різних режимах

Хост	Ok	Changed	Failed
web1	9	0	0
web2	9	0	0
db1	9	0	0
db2	9	0	0
mon1	5	0	0
web1	9	1	0
web2	9	1	0
db1	9	1	0
db2	9	1	0
mon1	5	0	0

Результати порівняння демонструють те, що у звичайному режимі значення параметра changed дорівнює нулю для всіх вузлів, що свідчить про досягнення цільового стану інфраструктури без необхідності внесення додаткових змін. У режимі dry-run зафіксовано умовні зміни на рівні одного елемента конфігурації для кожного вузла, що підтверджує коректність описаних задач та їхню застосовність до середовища виконання. А також в обох випадках відсутні помилки, що підтверджує стабільність та передбачуваність автоматизованого процесу розгортання.

Висновки та перспективи подальшого дослідження. У ході дослідження реалізовано підхід до автоматизації керування конфігураціями комп'ютерних систем із використанням Ansible та концепції Infrastructure as Code. Створено тестове середовище на основі контейнеризації, що дозволило змоделювати розподілену інфраструктуру з веб-серверами, серверами баз даних і вузлом моніторингу. Розроблено структурований Ansible-проєкт із використанням інвентарю, ролей і playbook-ів, що забезпечило модульність та повторне використання конфігурацій.

Результати виконання playbook підтвердили коректність реалізованого рішення. У стандартному режимі система не потребувала додаткових змін, що свідчить про досягнення цільового стану, а режим dry-run дозволив попередньо перевірити конфігурації без їх застосування. Це підтверджує ідемпотентність та передбачуваність роботи автоматизованої системи.

Перспективи подальших досліджень полягають у розширенні системи шляхом інтеграції з CI/CD-процесами, впровадженні централізованого керування секретами та масштабуванні рішення на хмарні та великі розподілені інфраструктури, а також у розробці підходів до автоматизованого тестування Ansible-конфігурацій.

Список бібліографічного опису:

1. Ansible Documentation. Official Ansible Documentation. URL: https://docs.ansible.com/ (дата звернення: 15.01.2026).
2. Infrastructure as Code: Managing Servers in the Cloud. Kief Morris. O'Reilly Media, 2022. 358 p.
3. Ansible for DevOps: Server and configuration management for humans. Jeff Geerling. Leanpub, 2021. 425 p.
4. HashiCorp Vault Documentation. Secrets Management. URL: https://www.vaultproject.io/docs (дата звернення: 18.03.2026).

5. Molecule Documentation. Ansible Role Testing. URL: https://molecule.readthedocs.io/ (дата звернення: 20.03.2026).
6. GitOps and Kubernetes: Continuous Deployment with Argo CD, Jenkins X, and Flux. Billy Yuen, Alexander Matyushentsev. Manning Publications, 2021. 328 p.

References:

1. Ansible Documentation. Official Ansible Documentation. URL: https://docs.ansible.com/ (accessed: 15.01.2026).
2. Infrastructure as Code: Managing Servers in the Cloud. Kief Morris. O'Reilly Media, 2022. 358 p.
3. Ansible for DevOps: Server and configuration management for humans. Jeff Geerling. Leanpub, 2021. 425 p.
4. HashiCorp Vault Documentation. Secrets Management. URL: https://www.vaultproject.io/docs (accessed: 18.03.2026).
5. Molecule Documentation. Ansible Role Testing. URL: https://molecule.readthedocs.io/ (accessed: 20.03.2026).
6. GitOps and Kubernetes: Continuous Deployment with Argo CD, Jenkins X, and Flux. Billy Yuen, Alexander Matyushentsev. Manning Publications, 2021. 328 p.

Історія статті:
Отримано: 18.05.2026 Доопрацьовано: 19.05.2026 Прийнято до друку: 23.05.2026 Опубліковано: 29.05.2026