

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

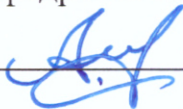
ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра
на тему «Методи та технології автоматизації управління
корпоративними серверами на базі Linux»

Виконав: студент 2-го курсу групи КІм-24-1
спеціальності 123 «Комп'ютерна інженерія»


Олександр МАКАРОВ

Керівник: доцент кафедри КІ та КБ, к.т.н.


Олексій ШЕЛУХА

Рецензент: доцент кафедри КІ та КБ, к.пед.н., доцент

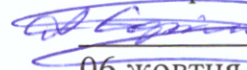

Олена ГОЛОВНЯ

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко
06 жовтня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Макаров Олександр Вікторович

Тема роботи: «Методи та технології автоматизації управління корпоративними серверами на базі Linux»

затверджена Наказом університету від 06 жовтня 2025 р. № 566/с

Термін здачі студентом закінченої роботи 24 грудня 2025 р.

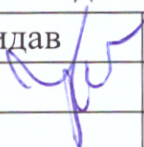
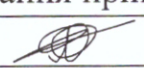
Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

Віртуальні середовища, сервер, віртуальні приватні мережі, процес автоматизації управління корпоративними серверами на базі Linux;

Обладнання та програмне забезпечення:

1. Віртуальні середовища VirtualBox;
2. Віртуальні машини на базі Ubuntu та CentOS;
3. Інструменти автоматизації: Packer, Ansible, Kubernetes
4. Інструмент контейнеризації: Docker
5. Інструменти керування: SSH

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Фальковський І.Г.		

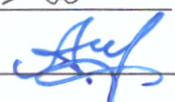
Студент  Олександр МАКАРОВ

Керівник  Олексій ШЕЛУХА

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	06.10.2025 – 10.10.2025	Виконано
2	Аналіз предметної області. Огляд існуючих рішень автоматизації (IaC), порівняння ОС Linux та інструментів оркестрації.	11.10.2025 – 19.10.2025	Виконано
3	Написання вступної частини кваліфікаційної роботи	21.10.2025 – 28.10.2025	Виконано
4	Проектування архітектури системи автоматизації.	02.11.2025 – 07.11.2025	Виконано
5	Реалізація архітектури.	08.11.2025 – 17.11.2025	Виконано
10	Перевірка роботи розробленої архітектури.	18.11.2025 – 23.11.2025	Виконано
10	Оформлення кваліфікаційної роботи	23.11.2025 – 06.12.2025	Виконано
11	Підготовка презентації до захисту	08.12.2025 – 24.12.2025	Виконано

Студент  Олександр МАКАРОВ

Керівник  Олексій ШЕЛУХА

АНОТАЦІЯ

Пояснювальна записка містить: 83 сторінок текстової частини, 24 рисунків, 2 таблиць та 16 літературних джерел.

Метою роботи є розробка та впровадження автоматизованої системи управління корпоративними серверами на базі ОС Linux з використанням підходів «Інфраструктура як код» (IaC).

В кваліфікаційній роботі розглядається процес побудови сучасної серверної інфраструктури з використанням методів віртуалізації, контейнеризації та оркестрації. В першому розділі проведено аналітичний огляд серверних дистрибутивів Linux, проаналізовано принципи роботи технології IaC та порівняно сучасні інструменти автоматизації (Terraform, Ansible, Chef), а також визначено технічне завдання для теми «Методи та технології автоматизації управління корпоративними серверами на базі Linux». В другому розділі, відповідно до поставлених завдань, проведено проєктування архітектури системи на базі гіпервізора VirtualBox, обґрунтовано вибір зв'язки інструментів Packer та Ansible, а також розроблено сценарії автоматизованого розгортання для трьох типів середовищ: Bare Metal, Docker та Kubernetes. В третьому розділі, згідно з проєктуванням у другому розділі, висвітлено практичну реалізацію автоматизованого створення образів віртуальних машин, налаштування серверів конфігурації через Ansible-плейбуки, а також розгортання та тестування гібридного кластера оркестрації контейнерів K3s.

Ключові слова: SERVER, LINUX, АВТОМАТИЗАЦІЯ, IAC, PACKER, ANSIBLE, DOCKER, KUBERNETES.

ANNOTATION

The explanatory note contains: 83 pages of text, 24 figures, 2 tables, and 16 references.

The purpose of the work is the development and implementation of an automated corporate server management system based on Linux OS using "Infrastructure as Code" (IaC) approaches.

The qualification work examines the process of building a modern server infrastructure using virtualization, containerization, and orchestration methods. The first section provides an analytical review of Linux server distributions, analyzes the operating principles of IaC technology, compares modern automation tools (Terraform, Ansible, Chef), and defines the terms of reference for the topic "Methods and Technologies for Automating Corporate Server Management based on Linux". In the second section, in accordance with the set tasks, the system architecture was designed based on the VirtualBox hypervisor, the choice of the Packer and Ansible toolset was justified, and automated deployment scenarios were developed for three types of environments: Bare Metal, Docker, and Kubernetes. The third section, according to the design in the second section, covers the practical implementation of automated virtual machine image creation, configuration server setup via Ansible playbooks, as well as the deployment and testing of a hybrid K3s container orchestration cluster.

Keywords: SERVER, LINUX, AUTOMATION, IAC, PACKER, ANSIBLE, DOCKER, KUBERNETES.

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ПРО СЕРВЕРА ТА ТЕХНОЛОГІЙ АВТОМАТИЗАЦІЇ..	10
1.1. Принцип роботи сервера та його види	10
1.2. Аналіз операційних систем для серверів	22
1.3. Аналіз технологій автоматизованого управління серверами.....	30
1.4. Постановка завдання дослідження	35
Висновок до другого розділу:	36
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ КОРПОРАТИВНОЇ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ	38
2.1. Архітектура топології та середовище віртуалізації	38
2.1.1. Вибір та обґрунтування платформи віртуалізації	38
2.1.2. Проєктування топології мережі та схема адресації	39
2.1.3. Характеристики та ролі віртуальних вузлів	39
2.2. Обґрунтування та аналіз інструментарію автоматизації.....	40
2.2.1. Packer як інструмент оркестрації ресурсів.....	40
2.2.2. Ansible як засіб управління конфігураціями	42
2.2.3. Модель інтеграції та розмежування зон відповідальності	42
2.3. Моделювання сценаріїв автоматичного розгортання та конфігурації.....	43
Висновок до другого розділу:	47
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО УПРАВЛІННЯ	48
3.1. Реалізація сценарію автоматизації класичного серверного середовища	48
3.1.1. Підготовка образу операційної системи засобами Packer.....	48
3.1.2. Налаштування конфігурації сервера веб-додатка за допомогою Ansible	50
3.1.3. Перевірка працездатності та відмовостійкості розгорнутого рішення.....	52
3.2. Розгортання контейнеризованої інфраструктури на базі Docker	53
3.2.1. Підготовка образу операційної системи засобами Packer.....	54
3.2.2. Побудова та запуск контейнерів додатку через Ansible-плейбуки	55
3.2.3. Перевірка роботи контейнерів	56
3.3. Організація та оркестрація кластера Kubernetes	58
3.3.1. Підготовка образу операційної системи засобами Packer.....	59
3.3.2. Ініціалізація та налаштування кластера	60
3.3.3. Перевірка роботи кластеру	62
3.4. Порівняльний аналіз реалізованих підходів до автоматизації.....	65
Висновок до третього розділу:.....	67
ВИСНОВКИ.....	69
ДОДАТКИ.....	73

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – Операційна Система

ПЗ – Програмне Забезпечення

API – Application Programming Interface

BS – Backup Server

DHCP – Dynamic Host Configuration Protocol

DNS – Domain Name System

NVM – Node Version Manager

RHEL – Red Hat Enterprise Linux

SSH – Secure Shell

UDP – User Datagram Protocol

VE – Virtual Environment

VM – Virtual Machine

YAML – YAML Ain't Markup Language

ВСТУП

Актуальність теми. У сучасному ІТ-середовищі, що характеризується стрімким зростанням обсягів даних та складності інфраструктурних рішень, традиційні методи ручного адміністрування серверів стають неефективними та вразливими до помилок. Корпоративні мережі потребують високої швидкості розгортання, масштабованості та стабільності, що робить перехід до автоматизації процесів критично важливим. Ключовим аспектом забезпечення безперервності бізнес-процесів є впровадження підходів «Інфраструктура як код» (IaC) та використання спеціалізованих інструментів для управління конфігураціями серверів на базі операційної системи Linux, яка є стандартом для корпоративних рішень.

Метою роботи є дослідження підходів до автоматизації управління серверами, що базуються на операційній системі Linux, та розробка системи автоматизованого розгортання, налаштування та управління корпоративною серверною інфраструктурою з використанням сучасних засобів оркестрації та управління конфігураціями.

Визначена мета обумовлює наступні завдання:

- Провести аналіз принципів роботи серверів, класифікації їх видів, операційних систем, зокрема Linux, та сучасних технологій, методів та засобів автоматизації управління серверами.
- Розробити модель автоматизованої системи управління корпоративними серверами на базі Linux.
- Виконати розгортання моделі та тестування роботи розробленої системи, провести аналіз її функціональних можливостей.

Об'єктом дослідження є процес автоматизації управління та адміністрування корпоративної серверної інфраструктури.

Предметом дослідження є операційні системи сімейства Linux (CentOS/Ubuntu), методологія Infrastructure as Code (IaC), система управління конфігураціями Ansible, інструмент створення образів Packer та платформа оркестрації контейнерів K3s.

У процесі виконання кваліфікаційної роботи було розроблено комплексне рішення для автоматизації управління серверами, реалізовано автоматичне створення шаблонних образів системи за допомогою Packer, налаштовано управління конфігураціями через Ansible та розгорнуто відмовостійкий кластер K3s для обслуговування корпоративних сервісів.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ПРО СЕРВЕРА ТА ТЕХНОЛОГІЙ АВТОМАТИЗАЦІЇ

1.1. Принцип роботи сервера та його види

Сервер — це комп'ютер, який надсилає дані іншим комп'ютерам, підключеним до нього мережею, і керує мережевими ресурсами. Сервери з'єднують різні різні пристрої та дозволяють їм обмінюватися інформацією та доступом до програмного забезпечення. Ви можете використовувати сервер, щоб забезпечити потужність обробки, зберігання даних, послуги резервного копіювання даних або створити центральне розташування для керування даними. Сервери зазвичай мають великий об'єм пам'яті та потужні процесори. Вони призначені для цілодобової роботи та виняткової надійності, тому що вони необхідні для забезпечення доступності IT-послуг.

Часто вони розміщуються централізовано та обслуговуються системним адміністратором. На відміну від цього, на столах співробітників є простіші машини, які називаються клієнтами, за допомогою яких вони отримують доступ до віддалених даних, наприклад, для включення в електронні таблиці, які вони створюють. Клієнтська та серверна машини з'єднані мережею, як показано на рис.1.1. Зверніть увагу, що ми показали мережу у вигляді простого овалу без будь-яких деталей. Ми будемо використовувати цю форму, коли маємо на увазі мережу в найбільш абстрактному сенсі.

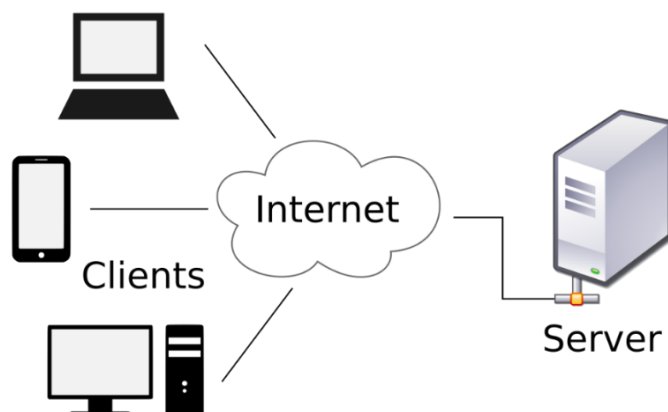


Рисунок 1.1 – Мережа з трьома клієнтами та одним сервером

Якщо ми детально розглянемо клієнт-серверну модель, то побачимо, що задіяні два процеси: один на клієнтській машині, а інший на серверній. Комунікація приймає форму клієнтського процесу, який надсилає повідомлення через мережу серверному процесу. Потім процес клієнта чекає на повідомлення відповіді. Коли серверний процес отримує запит, він виконує потрібну роботу або шукає запитані дані та надсилає відповідь[1]. Ці повідомлення показано на рис.1.2.



11

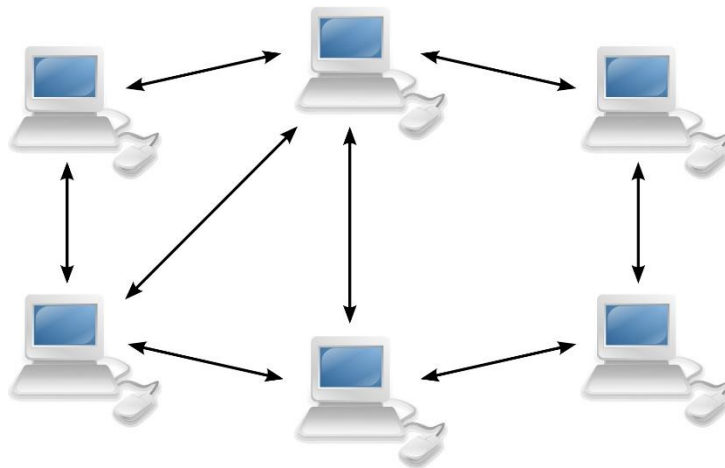


Рисунок 1.3 – Модель peer-to-peer зв'язка

Багато peer-to-peer система, наприклад BitTorrent, не мають центральної бази даних вмісту. Замість цього кожен користувач підтримує власну базу даних локально та надає список інших людей поблизу, які є членами системи. Потім новий користувач може перейти до будь-якого існуючого учасника, щоб побачити, що він має, і отримати імена інших учасників для перевірки додаткового вмісту та інших імен. Цей процес пошуку можна повторювати нескінченно довго, щоб створити велику локальну базу даних про те, що існує. Це заняття, яке може стати втомливим для людей, але комп'ютери в цьому справляються.

Peer-to-peer зв'язок часто використовується для обміну музикою та відео. У 2000 році це стало справжнім успіхом із службою обміну музикою під назвою Napster, яку було закрито після, мабуть, найбільшої справи про порушення авторських прав за всю історію. Існують також легальні програми для однорангового спілкування. До них належать шанувальники, які діляться загальнодоступною музикою, або сім'ї, які діляться фотографіями та фільмами, а також користувачі, які завантажують публічні пакети програмного забезпечення. Фактично, одна з найпопулярніших інтернет-додатків, електронна пошта, за своєю суттю є peer-to-peer.

Існують багато видів серверів[2], но ми розглянемо тільки ті що можуть знадобитись для корпоративної, такі як:

1. Веб-сервер: Сервер, який відповідає за публікацію веб-сайту в Інтернеті, називається веб-сервером. Сервер, який надає хостинг, який також називається "хостинг", через Інтернет-протокол, називається веб-сервером. Оренда місця, необхідного для публікації веб-сторінок в Інтернеті, називається «хостинг». Хостинг означає розміщення сторінок, зображень або документів, призначених для веб-сайту, на комп'ютері, який користувачі Інтернету використовують для доступу до них. Apache, Microsoft Internet Information Server (IIS) і Nginx є найпопулярнішими веб-серверами в Інтернеті. Дані для кожного сайту, який ви бачите в Інтернеті, зберігаються на іншому хості та програмному забезпеченні під назвою Веб-сервер. Веб-сервер використовується для ефективного використання цього комп'ютера. Веб-сервери зазвичай складаються з численних програмних пакетів. Кожен із цих програмних пакетів допомагає керувати різними функціями. Наприклад, пакет протоколу передачі файлів (FTP), який постачається з програмним забезпеченням веб-сервера, допомагає серверу виконувати завдання FTP. Подібним чином доступні численні пакети для виконання різноманітних завдань, таких як обслуговування електронних листів, виконання запитів на завантаження та публікація веб-сторінок через HTTP.

Найпоширеніші веб-сервери:

- HTTP-сервер Apache: Apache Software Foundation створила найпопулярніший у світі веб-сервер. Apache — це веб-сервер із відкритим вихідним кодом, який можна встановити на Linux, Unix, Windows, FreeBSD, Mac OS X та багатьох інших операційних системах. Веб-сервер Apache використовується понад 60% комп'ютерів веб-серверів.

- Internet Information Server: інформаційний сервер Інтернету (IIS) від Microsoft — це високопродуктивний веб-сервер.

- Lighttpd: Lighttpd, вимовляється як легкий, — це безкоштовний веб-сервер, який постачається з операційною системою FreeBSD. Цей веб-сервер із відкритим вихідним кодом швидкий, безпечний і споживає набагато менше потужності ЦП. Lighttpd також можна запускати в Windows, Mac OS X, Linux і Solaris.

- Nginx: веб-сервер Nginx є критично важливим компонентом обчислювального середовища клієнт-сервер, який надає клієнтським системам важливу для бізнесу інформацію. Як і Lighttpd, це веб-сервер із відкритим вихідним кодом, який відомий своєю швидкістю та потребує небагато ресурсів і часу для налаштування. Здебільшого він використовується для кешування, потокового мультимедіа, балансування навантаження, керування статичними файлами, автоматичного індексування та інших подібних завдань. Nginx обробляє запити в одному потоці, використовуючи асинхронну техніку, а не генеруючи нові процеси для кожного запиту, зробленого користувачем.

- Веб-сервер Sun Java System: веб-сервер Sun Microsystems призначений для середніх і великих веб-сайтів. Сервер є безкоштовним, але не має відкритого коду. З іншого боку, він працює на платформах Windows, Linux і Unix. Веб-сервер Sun Java System підтримує широкий спектр мов, сценаріїв і технологій Web 2.0, включаючи JSP, Java Servlets, PHP, Perl, Python, Ruby on Rails, ASP і Coldfusion.

2. Сервер бази даних: Сервер бази даних керує базою даних і надає послуги бази даних клієнтам. Сервер керує доступом до даних і пошуком, а також виконанням запитів клієнта. Сервер бази даних — це комп'ютер, на якому працює програмне забезпечення бази даних і призначений для надання послуг бази даних. Сервер бази даних складається з апаратного та програмного забезпечення, яке використовується для запуску бази даних.

Сервери баз даних мають різноманітні програми. Серед них:

- Регулярна робота з великими обсягами даних. Сервери баз даних сяють у клієнт-серверних архітектурах, де клієнти часто обробляють дані.

- Управління відновленням і безпекою СУБД (системи управління базами даних). Сервери баз даних забезпечують виконання обмежень, визначених у СУБД. Сервер керує всіма підключеними до нього клієнтами, а також усіма запитами доступу до бази даних і керування ними.

- Забезпечується одночасний контроль доступу. Сервери баз даних забезпечують багатокористувацьке середовище, в якому кілька користувачів

можуть отримати доступ до бази даних одночасно, зберігаючи безпеку та приховуючи СУБД від клієнтів.

- Сховище файлів програм і не баз даних. Сервери баз даних є набагато ефективнішим рішенням для деяких організацій, ніж файлові сервери.

Більшість серверів баз даних використовують модель клієнт-сервер. Вони отримують запити від клієнтських пристроїв та їхніх користувачів, а потім надсилають відповідь, яку запитували. Сервери баз даних — це потужні машини, які можуть підключатися до кількох баз даних і обслуговувати ресурси клієнтам, іноді за допомогою серверів додатків і веб-серверів, які діють як посередники. Бази даних, наприклад, можуть надавати ресурси клієнтам за відсутності веб-сервера або сервера додатків.

Найпоширеніші сервери баз даних:

- Oracle: Oracle — це база даних, яка найчастіше використовується для адміністрування об'єктно-реляційної бази даних, а остання версія — 12c (12 Cloud Computing). Він підтримує різні версії Linux, Windows і UNIX.

- IBM DB2: IBM DB2 було розроблено на мовах C, C++ і Assembly та вперше було випущено в 1983 році. Його налаштування та процедури встановлення прості, як і доступ до даних. Як результат, це допомагає зберігати великі обсяги даних, аж до петабайт.

- Microsoft SQL Server: цей сервер вперше був представлений у 1989 році. Він написаний кількома мовами, зокрема асемблером, C, Linux і C++. Він може працювати з Linux і Windows. Це дозволяє кільком користувачам використовувати ту саму базу даних одночасно.

- MySQL: MySQL стає все більш популярним для багатьох веб-додатків. Вона доступна як безкоштовна, так і преміальна версія.

- SAP HANA: створено компанією SAP SE і може керувати даними як SAP, так і не SAP. Він здатний підтримувати OLTP, OLAP і SQL. Він може спілкуватися з широким спектром різних програм.

- MS Access: він сумісний лише з Microsoft Windows. Оскільки ця система керування базами даних є недорогою, її зазвичай використовують для веб-сайтів електронної комерції.

3. DNS-сервер: Система доменних імен (DNS) — це телефонний довідник Інтернету. DNS відповідає за пошук правильної IP-адреси для веб-сайтів, коли користувачі вводять свої доменні імена, наприклад «google.com» або «nytimes.com», у веб-браузерах.

Потім ці адреси використовуються браузерами для зв'язку з початковими серверами або крайовими серверами CDN для доступу до інформації веб-сайту. Усе це можливо завдяки DNS-серверам, які є спеціалізованими машинами для відповідей на запити DNS.

Для розпізнавання імен система DNS має системи розпізнавання. Резолвери імен використовуються для пошуку IP-адрес, пов'язаних із доменними іменами. DNS-клієнти — це люди, які використовують розпізнавальні пристрої. Система DNS може мати багато розпізнавачів імен. У результаті, якщо один з них стає недієздатним, інші беруться за це і забезпечують, щоб зв'язок не порушувався.

Коли ви вводите доменне ім'я у свій браузер і входите в систему, ваш браузер спочатку запитує вашу операційну систему про доменне ім'я, яка не відповідає. Потім він запитує ваш модем і продовжує запитувати, доки не досягне Інтернет-провайдера та основних серверів DNS.

Якщо відповідна база даних DNS-сервера містить еквівалент цього доменного імені, вона відповість на IP-адресу, і ваш браузер отримає цю відповідь за частки секунди та зв'яжеться з сервером, на якому розміщено відповідне доменне ім'я. Потім розпочнеться передача даних, і потрібна веб-сторінка відобразиться у вашому браузері.

4. Файловий сервер: Файловий сервер — це центральний сервер у комп'ютерній мережі, який обслуговує файлові системи або частини файлових систем для клієнтів, підключених до мережі. Як наслідок, файлові сервери надають

користувачам центральне місце зберігання файлів на внутрішніх носіях даних, доступне всім авторизованим клієнтам. Адміністратор сервера встановлює суворі вказівки щодо того, які користувачі мають які права доступу: наприклад, конфігурація або авторизація файлів окремої файлової системи дозволяє адміністратору вказати, до яких файлів певний користувач або група користувачів може мати доступ і відкривати, а також чи дані можна лише побачити або додатково додати, змінити чи видалити.

Користувачі можуть отримувати доступ до файлів через локальну мережу, а також віддалено, якщо файлові сервери підключено до Інтернету та налаштовано належним чином. Це дозволяє користувачам переглядати та зберігати файли на файловому сервері, коли вони перебувають у русі. Усі поточні операційні системи, такі як Windows, Linux або macOS, можна використовувати на файловому сервері, але мережеві пристрої мають бути сумісні з операційною системою. Однак файлові сервери використовуються не тільки для зберігання та адміністрування файлів. Вони також часто використовуються як резервний сервер і як сховище для програм, які мають бути доступні численним учасникам мережі.

Клієнти можуть отримати доступ до віддаленої файлової системи лише через файлові сервери. Вони можуть зберігати будь-який матеріал, наприклад виконувати файли, документи, зображення або фільми. Вони зазвичай зберігають дані у вигляді двійкових блоків або файлів. Це означає, що вони не виконують подальшої індексації чи обробки файлів, які зберігають. Однак можуть існувати інші плагіни або серверні операції, які можуть надавати додаткові функції. Файловий сервер не має вбудованих методів для взаємодії з даними, і для цього покладається на клієнта. Оскільки бази даних обробляють лише структуровані дані, отримані за допомогою запиту, вони не вважаються файловими серверами.

Файлові сервери зазвичай містять додаткові можливості, які дозволяють кільком користувачам використовувати їх одночасно: Керування дозволами використовується для контролю того, хто має доступ до певних файлів і хто має повноваження змінювати або видаляти їх. Блокування файлу не дозволяє кільком

людям одночасно редагувати той самий файл. Вирішення конфліктів; збереження цілісності даних у разі перезапису файлу. Дублюючи дані на численні сервери в різних місцях, розподілена файлова система може зробити дані надлишковими та доступними.

Для багатьох підприємств використання файлового сервера є доцільним з різних причин. Першою перевагою є, очевидно, центральність, яка гарантує, що будь-який авторизований учасник мережі має доступ до збережених даних.

Це дає змогу спільно працювати над певними файлами. Конфлікти між кількома версіями документа можуть бути практично усунені, оскільки певні операції, наприклад редагування або видалення, недоступні для інших користувачів, щойно ви отримуєте доступ до файлу. Якщо користувачам доводиться ділитися необхідними файлами у своїй системі або надсилати їх через знімний носій, це займе значно більше часу та незручностей, і це майже напевно призведе до появи різних версій файлів.

Іншою значною перевагою використання файлових серверів є те, що це зменшує навантаження на клієнтські ресурси. За винятком особистих документів, майже всі корпоративні дані та резервні копії можна зберігати на файловому сервері, залежно від того, як організація збирається використовувати сховище файлів. А за належного розташування (каталоги, папки тощо) користувачі отримують набагато кращу перспективу повної інвентаризації файлів.

5. DHCP-сервер: Сервер DHCP — це мережевий сервер, який автоматично надає та призначає IP-адреси, шлюзи за замовчуванням та іншу мережеву інформацію клієнтським пристроям. Щоб відповісти на широкомовні запити клієнта, він використовує стандартний протокол, відомий як Dynamic Host Configuration Protocol або DHCP.

Сервер DHCP автоматично надсилає мережеві параметри, необхідні клієнтам для успішного спілкування в мережі. Без цього адміністратор мережі повинен вручну налаштувати кожного клієнта, який підключається до мережі, що може зайняти багато часу, особливо у великих мережах.

Сервери DHCP зазвичай призначають унікальну динамічну IP-адресу кожному клієнту, яка змінюється, коли закінчується термін оренди цієї IP-адреси.

Кожному пристрою, підключеному до Інтернету в IP-мережі, має бути призначена унікальна IP-адреса. DHCP дозволяє менеджерам мережі централізовано контролювати та призначати IP-адреси. Коли комп'ютер переміщується, він може автоматично надати йому нову IP-адресу. DHCP автоматизує процес розподілу IP-адрес, скорочуючи як час, необхідний для налаштування та розгортання пристрою, так і ймовірність помилок конфігурації. Сервер DHCP також може керувати налаштуваннями кількох сегментів мережі. Коли змінюється конфігурація сегмента мережі, адміністратору потрібно лише оновити відповідні параметри на сервері DHCP.

Конфігурація надійної IP-адреси. DHCP зменшує проблеми конфігурації IP-адреси вручну, наприклад друкарські помилки та конфлікти адрес, викликані призначенням IP-адреси декільком комп'ютерам одночасно. Адміністрування мережі спрощено. Щоб зменшити адміністрування мережі, DHCP підтримує такі функції:

- Конфігурація TCP/IP централізована та автоматизована. Можливість визначати налаштування TCP/IP з однієї точки контакту.

- Можливість використовувати DHCP для призначення різноманітних додаткових параметрів конфігурації TCP/IP. Ефективне керування змінами IP-адрес для клієнтів, які потрібно часто оновлювати, наприклад портативних пристроїв, які переміщуються по бездротовій мережі.

6. NTP-сервер: Протокол мережевого часу (NTP) — це Інтернет-протокол, який використовується для синхронізації з джерелами часу годинника комп'ютера в мережі. Він належить і є одним із найперших компонентів набору TCP/IP. Слово NTP стосується як протоколу, так і клієнт-серверних програм, які працюють на комп'ютерах. Він призначений для надзвичайної відмовостійкості та масштабованості, а також дозволяє тимчасову синхронізацію.

Після синхронізації клієнт оновлює годинник приблизно кожні 10 хвилин, потребуючи лише одного обміну повідомленнями на додаток до синхронізації клієнт-сервер. Ця транзакція відбувається на порту 123 протоколу дейтаграм користувача (UDP). NTP також забезпечує синхронізацію трансляції годинника комп'ютера.

Точний час для всіх пристроїв у комп'ютерній мережі є критичним з кількох причин; навіть частка секунди може спричинити проблеми. Нижче наведено деякі приклади використання NTP:

- Для розподілених процедур потрібен скоординований час, щоб гарантувати дотримання належної послідовності.
- Послідовний відлік часу в мережі потрібен для процедур безпеки.
- Зміни файлової системи, які виконуються на кількох машинах, залежать від синхронізованого часу.
- Щоб контролювати продуктивність і виправляти проблеми, системи прискорення мережі та керування мережею покладаються на точність часових позначок.

7. Сервер Syslog: Протокол системного журналювання (Syslog) — це стандартний формат повідомлень, який використовується мережевими пристроями для підключення до сервера журналювання. Він був створений, перш за все, щоб зробити моніторинг мережних пристроїв простим. Пристрої можуть використовувати агент Syslog для надсилання сповіщень за різними сценаріями.

Ці повідомлення журналу містять позначку часу, рейтинг серйозності, ідентифікатор пристрою (включно з IP-адресою) та інформацію про подію. Хоча він має недоліки, протокол Syslog широко використовується, оскільки він простий у створенні та дуже відкритий, що дозволяє використовувати різноманітні власні реалізації та, отже, можливість контролювати практично будь-який підключений пристрій.

Стандарт Syslog визначає три рівні:

- Syslog Content Layer: це вміст повідомлення про подію. Він містить кілька елементів даних, таких як коди закладів і рейтинги серйозності.
- Syslog Application Layer: повідомлення генерується, інтерпретується, маршрутизується та зберігається на цьому рівні.
- Syslog Transport Layer: цей рівень відповідає за надсилання повідомлень через мережу.

Під час усунення труднощів інформація журналу є досить корисною. Наприклад, якщо деякі користувачі повідомляють про збій у роботі мережі, як-от нещодавні збої у роботі Facebook, WhatsApp та Instagram, ми можемо просто просканувати всі дані журналу, щоб перевірити, чи не було проблем. Зберігання даних є ще однією перевагою централізованого зберігання інформації журналу. Він може надати тимчасову інформацію, необхідну для відновлення попереднього стану системи після збою.

Повідомлення системного журналу передаються через порт 514 протоколу дейтаграм користувача (UDP). Оскільки UDP є протоколом без підключення, повідомлення не підтверджуються та не гарантуються, що вони надходять. Це може бути недоліком, але це також робить систему базовою та легкою в управлінні.

Повідомлення системного журналу часто мають формат, зрозумілий людині, але це не обов'язково. Кожне повідомлення має рівень пріоритету в заголовку, який є поєднанням коду для процесу створення пристроєм повідомлення та рівня серйозності. Коди процесу, відомі як "об'єкти", засновані на UNIX. Рівні серйозності варіюються від 0 (надзвичайна ситуація) до 1 (потрібна термінова допомога), а інформаційні та налагоджувальні повідомлення знаходяться десь посередині. У поєднанні ці два коди дозволяють швидко класифікувати повідомлення Syslog.

8. Фізичний сервер: Деякі сервери використовуються виключно для певних функцій. Наприклад, сервер додатків просто розміщує веб-сторінку. Фізичні

сервери легко використовувати для широкого спектру мережесих завдань завдяки їх програмному та апаратному забезпеченню. Деякі з цих операцій:

- Оновлення операційної системи
- Програмне забезпечення для захисту від спаму
- Антивірусне програмне забезпечення
- Захист від DDoS атак
- Хостинг DNS
- Виявлення вторгнень
- Управління SNMP
- Адміністрування бази даних
- Резервне копіювання та відновлення

Отже, застосування різнопланових серверів є невід'ємною практикою функціонування різноманітних інформаційних систем, тому вибір оптимальних операційних систем для таких серверів та оптимізація їх взаємодії є актуальною задачею, що впливає на стабільність, продуктивність та безпеку роботи всієї системи. Іншою важливою проблемою є вплив людського фактору, який може суттєво спливати як на роботу окремих серверів та інформаційної системи в цілому, тому, для вирішення цієї проблеми актуальним є застосування інструментів, які дозволять автоматизувати процес налаштування серверів та зменшити вплив людського фактору.

1.2. Аналіз операційних систем для серверів

Серверна операційна система (ОС) — це операційна система, яка встановлюється та використовується на серверах для керування апаратними та програмними ресурсами сервера та надання послуг багатьом програмам.

Серверні операційні системи оптимізовані для надання більш розширених можливостей і функцій для архітектури клієнт-сервер або корпоративних ІТ-середовищ. А серверні операційні системи націлені на мережу, а не на одного користувача, тому вони можуть працювати з декількома робочими столами та спрощують адміністрування та зменшують простой.

Більшість серверів можуть підтримувати широкий спектр операційних систем. Кожна серверна ОС має свої переваги і недоліки. Нижче наведено найпоширеніші серверні операційні системи на ринку:

- Windows Server: Microsoft розробила сімейство операційних систем Windows для повсякденного особистого використання та професійного використання на серверах. ОС Windows Server підтримує керування на корпоративному рівні, зберігання даних і широкий спектр програм.

Windows Server підтримує керування віртуальною пам'яттю, повний графічний інтерфейс робочого столу, дозволяє багатозадачність і підтримує різні периферійні пристрої. Корпорація Майкрософт зазвичай надає 10 років підтримки Windows Server.

Перевагами серверної ОС Windows є інтуїтивно зрозумілий графічний інтерфейс користувача, підтримка симетричних багатопроцесорних систем, чудова підтримка програм сторонніх розробників і багато версій на вибір. Мінусами є потреба в ліцензуванні на основі користувачів і більше загроз безпеки від вірусів порівняно з іншими платформами.

- MacOS Server: MacOS Server — це UNIX-подібна серверна операційна система на основі macOS, розроблена Apple. ОС створена на основі macOS і додає функції сервера та інструменти системного адміністрування, а також інструменти для керування пристроями macOS та iOS.

MacOS Server — чудовий вибір, якщо ви використовуєте клієнти Mac у своїй мережі, враховуючи його здатність легко створювати функції для клієнтів Mac.

Перевагами macOS Server є просте адміністрування, інтуїтивно зрозумілий графічний інтерфейс, чудова підтримка та простий розподіл робочого навантаження між кількома машинами. Таким чином, легко збільшити потужність обробки. ОС поставляється з необмеженою ліцензією користувача.

Мінуси полягають у тому, що macOS Server працює лише на апаратному забезпеченні Apple, яке може бути дорогим, а сторонніх програм не так багато. Крім того, незважаючи на те, що Apple використовує програмне забезпечення з

відкритим кодом у своїй системі, є зміни, характерні для macOS, які вимагають вирішення деяких проблем, яких немає в Linux

- Linux: Linux — це сімейство UNIX-подібних операційних систем, які мають усі функції UNIX. Це відкритий вихідний код, доступний безкоштовно, що полегшує багатокористувацькі, багатопроцесні та багатопотокові операції. Однак сервери Linux вимагають більше технічних знань - від установки до обслуговування та виправлення помилок.

Одна з найпоширеніших платформ для розміщення веб-сайтів, стек LAMP, базується на Linux (Linux, Apache, MariaDB/MySQL, Perl/PHP/Python).

Перевагами серверної ОС Linux є висока безпека, широкий вибір дистрибутивів, інтегроване програмне забезпечення з відкритим вихідним кодом, включаючи компілятори мов високого рівня, і можливість керувати системою за допомогою графічного інтерфейсу користувача.

Мінусами є відсутність довгострокової підтримки деяких дистрибутивів і певних складних операцій, таких як оновлення системи.

Таблиця 1.1

Порівняння операційних систем для серверу

Критерій	Windows Server	Linux	MacOS
Стабільність	Досить стабільна, але вимагає регулярних оновлень	Висока стабільність, довгі цикли підтримки	Досить стабільний, але обмежений у функціоналі
Безпека	Потужні засоби безпеки, але вимагає постійної уваги та оновлень	Високий рівень безпеки завдяки відкритому коду та активній спільноті	Високий рівень захисту, але без гнучких налаштувань
Гнучкість	Досить гнучка, але може бути обмежена у деяких аспектах	Висока гнучкість, дозволяє налаштувати систему під будь-які потреби	Обмежена, підходить для простих задач
Підтримка спільноти	Велика спільнота, але менш активна, ніж у Linux	Велика та активна спільнота розробників та користувачів	Невелика спільнота, мінімальна активність

Довгострокова підтримка	Доступна за додаткову плату	Багато дистрибутивів мають довгі цикли підтримки без додаткових витрат	Регулярні оновлення, але обмежена перспектива
--------------------------------	-----------------------------	--	---

Існує багато дистрибутивів Linux, які підходять для серверних задач[3], серед них це:

1. Ubuntu Server: Ubuntu Server — це популярний серверний дистрибутив Linux, відомий своєю стабільністю, безпекою та гнучкістю. Побудований на базі ядра Linux, він забезпечує надійну роботу як у малих проєктах, так і в масштабних корпоративних інфраструктурах. Завдяки довготривалим версіям (LTS) система отримує регулярні оновлення та підтримку протягом п'яти років, що робить її особливо привабливою для довгострокових проєктів. Ubuntu Server має потужний рівень безпеки, використовуючи сучасні технології, такі як AppArmor та UFW, а також можливість аудиту коду через відкритість проєкту.

Гнучкість платформи дозволяє налаштовувати її під різноманітні задачі, включаючи веб-хостинг, управління базами даних, віртуалізацію та хмарні сервіси. Вона підтримує інструменти для автоматизації, такі як Ansible і Puppet, і інтегрується з популярними хмарними платформами, включаючи AWS, Azure і Google Cloud.

Активна спільнота користувачів і розробників сприяє швидкому вирішенню проблем та надає багату документацію, що особливо важливо для новачків. Ubuntu Server легко адаптується до різних середовищ і обладнання, а його відкритий код дозволяє налаштовувати систему відповідно до індивідуальних потреб. Це безкоштовна, продуктивна та проста у використанні платформа, яка залишається одним із найкращих виборів для серверних задач будь-якого рівня складності.

2. Debian Server: Debian Server — це надійний серверний дистрибутив Linux, відомий своєю стабільністю, універсальністю та акцентом на відкритість. Завдяки своїй репутації "еталонної" Linux-системи, Debian використовується як основа для багатьох інших дистрибутивів, включаючи Ubuntu. Стабільність системи

забезпечується через ретельне тестування всіх пакетів перед включенням у стабільну гілку, що робить Debian ідеальним вибором для критично важливих серверів. Його політика довгострокової підтримки (LTS) гарантує оновлення безпеки для стабільних релізів протягом щонайменше п'яти років, що дозволяє підтримувати стабільну та безпечну інфраструктуру.

Безпека є ключовим аспектом Debian Server — оновлення безпеки випускаються оперативно, а відкритий код дозволяє проводити аудит програмного забезпечення. Активна та глобальна спільнота користувачів і розробників забезпечує швидку підтримку, багату документацію та вирішення проблем. Debian також повністю безкоштовний, що робить його доступним для всіх без обмежень.

Ця система чудово підходить як для невеликих серверів, так і для масштабних корпоративних середовищ, де важливі стабільність і контроль. Завдяки своєму консервативному підходу до змін, Debian Server залишається одним із найнадійніших і найуніверсальніших рішень для серверних задач.

3. CentOS: CentOS Server — це стабільний і потужний серверний дистрибутив Linux, створений на основі Red Hat Enterprise Linux (RHEL). Його основною перевагою є висока стабільність і надійність, завдяки чому він довгий час використовувався у корпоративних середовищах та для критично важливих серверних задач. CentOS має тривалий життєвий цикл підтримки, що дозволяє компаніям створювати інфраструктури з мінімальною потребою в частих оновленнях і міграціях.

Система відома своєю безпекою, оскільки вона отримує регулярні оновлення та патчі, успадковані від RHEL. Інтегровані інструменти, такі як SELinux, забезпечують додатковий рівень контролю та захисту. CentOS пропонує широкі можливості для налаштування, завдяки підтримці інструментів автоматизації, віртуалізації та управління контейнерами, таких як Docker та Kubernetes.

4. CoreOS: CoreOS — це спеціалізований серверний дистрибутив Linux, оптимізований для контейнерних середовищ і масштабованих хмарних інфраструктур. Його основний фокус — забезпечення легкого розгортання,

автоматичного оновлення та ефективного управління контейнерами. CoreOS побудований для сучасних DevOps-процесів і є ідеальним вибором для платформ, які покладаються на оркестрацію контейнерів, таких як Kubernetes.

Система забезпечує високу стабільність завдяки своєму мінімалістичному підходу: вона включає лише необхідні компоненти, що зменшує потенційні точки відмови та підвищує продуктивність. CoreOS використовує атомарні оновлення, що дозволяють безпечно оновлювати систему без простоїв, а у випадку збоїв можна легко повернутися до попереднього стану.

Безпека CoreOS є одним із головних пріоритетів. Усі оновлення застосовуються автоматично, що мінімізує ризики, пов'язані з уразливостями. Крім того, система підтримує технології ізоляції контейнерів, які додають додатковий рівень захисту в мультиконтейнерних середовищах.

Гнучкість CoreOS досягається завдяки глибокій інтеграції з Docker і Kubernetes, що дозволяє створювати, масштабувати та управляти мікросервісами на будь-якій інфраструктурі. CoreOS працює як у локальних дата-центрах, так і в хмарних середовищах, таких як AWS, Azure і Google Cloud.

5. Fedora Server: Fedora Server — це серверний дистрибутив Linux, орієнтований на інновації, сучасні технології та гнучкість. Він є частиною екосистеми Fedora Project, що спонсорується Red Hat, і часто слугує тестовим майданчиком для функцій, які згодом інтегруються в Red Hat Enterprise Linux (RHEL). Fedora Server пропонує актуальні оновлення та підтримує широкий спектр сучасних серверних технологій, що робить його ідеальним для тих, хто прагне бути на передовій розвитку програмного забезпечення.

Стабільність Fedora Server є достатньою для багатьох задач, хоча через часті релізи (кожні 6 місяців) він більше підходить для експериментів або розробки, ніж для критично важливих середовищ. Новації включають модульність пакетів, що дозволяє обирати різні версії програмного забезпечення залежно від потреб.

Безпека системи забезпечується завдяки постійним оновленням, активній підтримці спільноти та інструментам, як-от SELinux, що інтегрований за

замовчуванням. Fedora Server також підтримує новітні технології шифрування і забезпечує швидке отримання патчів для вразливостей.

Гнучкість Fedora Server дозволяє його використовувати для різних задач, включаючи веб-хостинг, бази даних, контейнерні платформи та хмарні сервіси. Система підтримує сучасні технології, такі як Podman, Kubernetes та Cockpit, що забезпечує зручний веб-інтерфейс для адміністрування.

6. Red Hat Enterprise Linux: RHEL — це комерційний серверний дистрибутив Linux, орієнтований на корпоративні середовища. Він відомий своєю стабільністю, високим рівнем безпеки та професійною технічною підтримкою. RHEL є основою для багатьох серверних інфраструктур, пропонуючи довгострокову підтримку (до 10 років) і регулярні оновлення, що робить його ідеальним для критично важливих задач.

Стабільність системи забезпечується строгим тестуванням усіх компонентів і ретельним контролем якості перед кожним релізом. Завдяки цьому RHEL підходить для великих організацій, яким потрібне передбачуване та надійне середовище.

Безпека є однією з головних переваг RHEL. Він інтегрує такі інструменти, як SELinux і firewallD, забезпечуючи високий рівень захисту. Дистрибутив регулярно отримує оновлення безпеки, а сертифікації відповідності стандартам (наприклад, FIPS) роблять його придатним для використання в урядових і фінансових організаціях.

Гнучкість RHEL проявляється у його здатності працювати в різних середовищах: від локальних серверів до хмарних платформ і контейнерних оркестрацій, таких як Kubernetes. Він підтримує широкий спектр програмного забезпечення та інтегрується з інструментами автоматизації, як-от Ansible.

Підтримка спільноти через проекти Fedora і CentOS Stream забезпечує доступ до інновацій, які згодом включаються в RHEL. Комерційна підтримка Red Hat надає експертну допомогу, включаючи рекомендації щодо налаштування, оптимізації та усунення неполадок.

Порівняння дистрибутивів Linux [4] [5] [6] [7] [8] [9] [10]

Параметр	Ubuntu Server	Debian	CentOS	Fedora Server	CoreOS	RHEL
Призначення	Загального призначення, простота використання	Універсальний, стабільний	Серверне середовище, альтернатива RHEL	Нові технології, тестування інновацій	Контейнеризація, Kubernetes	Корпоративне використання, стабільність
Основна аудиторія	Початківці, середній бізнес, адміністратори	Адміністратори, орієнтовані на стабільність	Великі компанії, підприємства	Розробники, ентузіасти Linux	DevOps, контейнерна інфраструктура	Корпоративний сектор
Підтримка оновлень	До 5 років (LTS версії)	До 2–3 роки	До 10 років	До 1 рік	Регулярні оновлення	До 10 років
Стабільність	Висока (у LTS версіях)	Дуже висока	Дуже висока	Середня	Висока	Дуже висока
Складність налаштування	Низька	Висока	Висока	Середня	Висока	Висока
Використання в бізнесі	Середнє	Середнє	Високе	Низьке	Висока	Висока
Модель оновлень	LTS/регулярні оновлення	Повільні, орієнтовані на стабільність	Повільні, стабільні	Швидкі	Швидкі	Повільні, орієнтовані на стабільність
Ліцензія	Безкоштовна	Безкоштовна	Безкоштовна	Безкоштовна	Безкоштовна	Платна

Ubuntu Server ідеально підходить для малого та середнього бізнесу, а також для новачків у серверній адміністрації, завдяки простоті налаштування, активній спільноті та довготривалим оновленням у LTS-версіях. Він є універсальним і підтримує широкий спектр програмного забезпечення, що робить його популярним для веб-серверів, баз даних і хмарних рішень.

Debian орієнтований на тих, хто шукає максимальної стабільності та надійності. Завдяки строгому тестуванню пакетів і тривалому циклу підтримки, Debian ідеально підходить для критично важливих серверів, таких як сервери баз даних чи мережеві шлюзи, де важлива безперебійна робота.

CentOS є вибором для корпоративних користувачів, які потребують стабільної платформи для довготривалих проєктів. Це безкоштовна альтернатива RHEL, яка використовується для великих інфраструктур, таких як веб-хостинг або ERP-системи, завдяки високій сумісності з RHEL.

Fedora Server спрямована на розробників і ентузіастів, які бажають тестувати новітні технології. Завдяки швидкому циклу оновлень Fedora використовується для експериментів з новими версіями програмного забезпечення, інновацій у віртуалізації та контейнеризації.

CoreOS спеціалізується на контейнерних середовищах і орієнтований на DevOps-команди. Це ідеальний вибір для побудови кластерів Kubernetes або роботи з хмарними платформами, де потрібна легка і масштабована ОС.

RHEL є стандартом для корпоративного використання завдяки комерційній підтримці, тривалому циклу оновлень і стабільності. Він підходить для великих підприємств, де потрібна висока надійність, відповідність стандартам безпеки та повна підтримка виробника.

1.3 Аналіз технологій автоматизованого управління серверами.

Інфраструктура як код (IaC) - це метод визначення та управління інфраструктурою вашої системи за допомогою коду. Подібно до того, як програмний код визначає логіку та функціональність програми, IaC описує архітектуру системи, включаючи такі компоненти, як сервери, мережі, сховища та операційні системи[12].

За допомогою IaC можна керувати ресурсами інфраструктури у послідовний та організований спосіб. Він розглядає конфігураційні файли як вихідний код, що дозволяє використовувати контроль версій, відстежувати помилки та легко вносити оновлення. Ці конфігураційні файли можуть бути написані такими мовами, як Python або Java, і зазвичай розробляються в інтегрованих середовищах розробки (IDE), які допомагають виявляти помилки та оптимізувати процес кодування. Таким чином, за допомогою таких систем контролю версій можна відстежувати всі внесені зміни та ефективно співпрацювати з розробниками.

DevOps фокусується на тому, щоб команди розробників і операторів створювали програмне забезпечення швидше і якісніше. Інфраструктура як код відіграє важливу роль у цьому, автоматизуючи управління інфраструктурою разом з безперервною інтеграцією та безперервними конвеєрами доставки.

Додавання IaC до DevOps допоможе автоматично змінювати інфраструктуру залежно від змін у програмі. Ніяких помилок вручну, і зміни вносяться однаково в різних середовищах.

IaC має широкий спектр застосувань у таких сферах як:

- Хмарні обчислення: IaC широко використовується в хмарних обчисленнях, де його можна застосовувати для надання та налаштування хмарних ресурсів, таких як віртуальні машини, сховища та бази даних.

- DevOps: IaC є ключовим компонентом DevOps, де він використовується для автоматизації розгортання та управління інфраструктурою і додатками.

- Безперервна інтеграція та доставка (CI/CD): IaC використовується в конвеєрах CI/CD для автоматизації розгортання та конфігурації інфраструктури та додатків.

- Мережеві технології: IaC можна використовувати для автоматизації розгортання і управління мережами, включаючи створення і управління підмережами, групами безпеки і брандмауерами.

- Розгортання веб-додатків: IaC можна використовувати для автоматизації розгортання та управління веб-додатками, включаючи вказівку веб-сервера, сервера додатків і балансувальника навантаження.

- Розгортання баз даних: IaC можна використовувати для автоматизації розгортання та управління базами даних, включаючи вказівку механізму бази даних, створення таблиць і налаштування користувачів.

- Великі дані: IaC можна використовувати для автоматизації розгортання та управління інфраструктурою великих даних, включаючи створення кластерів і налаштування фреймворків обробки даних.

Щоб створити архітектуру, яка базується на IaC, потрібно мати інструментарій, який буде наведено нижче[13]:

1. Terraform: Terraform - один з найпопулярніших інструментів IaC на ринку. Це проект з відкритим вихідним кодом з неймовірною гнучкістю, що підтримує всі найвідоміші хмарні платформи.

Terraform підтримує незмінну інфраструктуру, дозволяючи замінювати ресурси, а не оновлювати їх при внесенні змін. Визначаючи інфраструктуру як код, Terraform гарантує, що будь-яка модифікація призводить до знищення та відновлення ресурсів, підтримуючи узгодженість та мінімізуючи дрейф конфігурації. Такий підхід підвищує надійність, передбачуваність і знижує ризики розгортання.

Від DNS, управління секретами та моніторингу до постачальників систем контролю версій - ви можете використовувати Terraform для попередньої конфігурації та надання інфраструктури з нуля. Розробники можуть задекларувати бажану конфігурацію інфраструктури за допомогою власного синтаксису HCL або вже знайомої їм мови програмування, наприклад, Typescript, Python, Java, C# або Go, за допомогою набору для розробки хмарних додатків (Cloud Development Kit)

Окрім вбудованої перевірки синтаксису, Terraform також дозволяє попередньо переглянути вплив змін у коді до того, як вони будуть застосовані до ресурсів інфраструктури. Таким чином, ви можете все перевірити ще раз, запобігти помилкам і, як наслідок, уникнути непотрібних витрат.

2. Ansible: Ansible - це інструмент оркестрування та конфігурації від Red Hat. Ansible був створений для простоти та автоматизації з самого початку. Надійна конфігурація за замовчуванням дозволяє використовувати його одразу, не потребуючи додаткової конфігурації.

Як інструмент IaC, Ansible використовує конфігураційні модулі під назвою «Playbooks», написані на YAML, де ви можете налаштувати бажаний кінцевий стан вашої інфраструктури. Якщо ви знайдете варіант використання, який не може бути вирішений за допомогою стандартних модулів, Ansible дозволяє вам написати власні модулі та плагіни.

Ansible покращує розробку, автоматизуючи багато повторюваних і складних завдань, заощаджуючи багато часу при встановленні пакунків або налаштуванні великої кількості серверів. Наприклад, створення набору Playbooks вимагає певних

витрат часу, але налаштування нових машин стає неймовірно швидким, коли ви маєте достатню кількість пакунків.

3. Pulumi: Pulumi - це інструмент IaC, який відрізняється від інших платформ «Інфраструктура як код» більшою гнучкістю. Він підтримує кілька мов програмування, таких як Python, JavaScript, C#, Go та TypeScript. Забезпечуючи більшу підтримку мовних опцій, Pulumi може відповідати більшій кількості варіантів використання IaC DevOps і охопити більшість розробників. Більша кількість мов також означає, що у вас є більше інструментів та фреймворків для побудови та тестування вашої інфраструктури.

Унікальним аспектом Pulumi як інструменту IaC є те, що він чудово справляється з підтримкою основних концепцій та функцій відомих інструментів, таких як Terraform, пропонуючи при цьому підтримку хмарних гігантів AWS, GCP та Azure Cloud.

4. Vagrant: Розроблений тим же творцем Terraform, HashiCorp, Vagrant пропонує рішення для професіоналів, які використовують невелику кількість віртуальних машин замість великих хмарних інфраструктур. Продукт орієнтований на розробників, які працюють у значно менших масштабах, оскільки він чудово підходить для швидкого створення середовищ розробки.

За допомогою Vagrant ви можете налаштувати віртуальну машину, запустити тести та зберегти всі конфігурації на цій віртуальній машині у Vagrantfile. Ви можете поділитися цим з іншими розробниками, щоб вони могли відтворити ті ж результати і працювати з тим же середовищем розробки.

Vagrant може працювати разом з VirtualBox, AWS та будь-яким іншим хмарним провайдером, який надає рішення для віртуальних машин як частину своїх послуг. Він також може бути інтегрований з іншими інструментами IaC, такими як Chef і Puppet.

5. Chef: Chef - один з найпопулярніших інструментів Інфраструктури як Коду на ринку. Зараз він переживає деякі зміни після того, як його придбала компанія

Progress. Однак, цей неспокійний період та велика кількість звільнень, що послідували за ним, підштовхнули багатьох користувачів до переходу на Ansible.

Цей інструмент IaC використовує «рецепти» та «кулінарні книги», що базуються на доменній мові (DSL) на основі Ruby. Користувач повинен писати код на кожному кроці конфігурації, щоб досягти бажаного стану для додатків, сервісів та утиліт. Chef - це хмарна діагностика, що працює з великими хмарними провайдерами, такими як AWS, GCP та Azure Cloud. Він також підтримує API для надання послуг, що робить його чудовим інструментом Infrastructure as Code, який можна використовувати разом з Terraform.

Його абсолютна гнучкість у поєднанні з вбудованою функцією усунення дрейфу та можливістю конфігурувати політики у вигляді коду робить його масштабованим і придатним для використання в будь-яких існуючих конвеєрах CI/CD. Ці особливості роблять Chef одним з найсильніших претендентів у нашому списку найкращих інструментів для розробки інфраструктури як коду.

6. Puppet: Puppet має багато спільного з Chef у порівнянні з іншими інструментами IaC з нашого списку і є частиною фундаменту багатьох конвеєрів CI/CD, побудованих інженерами DevOps. Він використовує DSL на основі Ruby, де ви можете задекларувати кінцевий стан вашої інфраструктури і те, що ви хочете, щоб вона робила. Puppet заповнює прогалину, знаходячи найкращий спосіб досягти задекларованого раніше стану конфігурації.

Якщо після цього відбувається будь-яке відхилення конфігурації, Puppet відстежує і автоматично виправляє будь-які некоректні зміни. Цей проект з відкритим вихідним кодом наразі підтримує всі відомі хмарні платформи, такі як GCP, Azure Cloud, AWS, що дозволяє автоматизувати роботу з різними провайдерами.

7. Packer: Packer — це інструмент з відкритим вихідним кодом від компанії HashiCorp, призначений для автоматизації створення ідентичних образів віртуальних машин (Machine Images) для різних платформ із єдиної джерельної конфігурації.

На відміну від інструментів керування конфігураціями (як Ansible або Chef), які налаштовують сервер уже після його запуску, Packer працює на етапі попередньої підготовки. Він дозволяє реалізувати підхід «незмінної інфраструктури» (Immutable Infrastructure) шляхом створення так званих «золотих образів» (Golden Images). Ці образи є незмінними артефактами, що вже містять встановлену операційну систему, налаштовані системні параметри та інстальоване програмне забезпечення.

Архітектура Packer базується на трьох основних компонентах:

- Builders (Будівельники): відповідають за створення віртуальних машин у специфічних середовищах (наприклад, VirtualBox, VMware, AWS, Azure, Docker).
- Provisioners (Провіженери): дозволяють використовувати скрипти (Shell, PowerShell) або сторонні інструменти автоматизації (Ansible, Puppet) для налаштування системи всередині образу під час його збірки.
- Post-Processors (Пост-процесори): виконують дії з готовим образом, наприклад, стиснення, експорт у хмару або генерацію Vagrant-боксу.

Використання Packer забезпечує значне прискорення процесу розгортання (Time-to-Live), оскільки сервери запускаються з вже готового та перевіреного образу, а не налаштовуються з нуля. Це також гарантує повну ідентичність середовищ розробки (Development), тестування (Testing) та експлуатації (Production), мінімізуючи ризики помилок конфігурації.

1.4. Постановка завдання дослідження

На основі проведеного аналізу принципів роботи серверів та їх класифікації, операційних систем, що використовуються для серверів, а також сучасних технологій автоматизованого управління, було визначено потребу в єдиних стандартах налаштування серверів. Це дозволить уникнути хаосу в конфігураціях та прискорити розширення інфраструктури. Оскільки кількість серверів постійно зростає, ними потрібно керувати ефективно, щоб бізнес працював без збоїв. Саме

необхідність автоматизувати ці процеси й робить тему актуальною, а також визначає мету мого дослідження.

Мета дослідження: розробити підходи до автоматизації управління серверами, що базуються на операційній системі Linux, та розробка системи автоматизованого розгортання, налаштування та управління корпоративною серверною інфраструктурою з використанням сучасних засобів оркестрації та управління конфігураціями.

Для досягнення мети були визначені такі задачі:

- Провести аналіз принципів роботи серверів, класифікації їх видів, операційних систем, зокрема Linux, та сучасних технологій, методів та засобів автоматизації управління серверами

- Розробити модель автоматизованої системи управління корпоративними серверами на базі Linux.

- Виконати розгортання моделі та тестування роботи розробленої системи, провести аналіз її функціональних можливостей

Досягнення поставленої мети дозволить забезпечити ефективне управління корпоративними серверами шляхом впровадження автоматизованих інструментів, що мінімізують витрати часу на адміністративні задачі та зменшують кількість помилок, пов'язаних із ручними процесами.

Висновок до першого розділу:

У першому розділі проаналізовано сучасні підходи до побудови серверної інфраструктури. Встановлено, що операційні системи сімейства Linux є оптимальним вибором для корпоративного сектору завдяки стабільності та гнучкості. Дослідження парадигми «Інфраструктура як код» (IaC) показало, що автоматизація процесів адміністрування дозволяє мінімізувати помилки людського фактору. За результатами порівняльного аналізу інструментів автоматизації обрано Ansible як основний засіб управління конфігураціями завдяки його без-агентній архітектурі.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ КОРПОРАТИВНОЇ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ

2.1. Архітектура топології та середовище віртуалізації

Розробка системи автоматизованого управління серверами вимагає створення контрольованого, ізольованого середовища, яке дозволяє безпечно моделювати сценарії розгортання інфраструктури без ризику порушення роботи реальних корпоративних мереж. У цьому підрозділі обґрунтовано вибір платформи віртуалізації, описано спроектовану мережеву топологію та визначено технічні специфікації віртуальних вузлів.

2.1.1. Вибір та обґрунтування платформи віртуалізації

Для реалізації практичної частини дослідження було обрано використання гіпервізора другого типу (Hosted Hypervisor). Такий підхід дозволяє розгорнути серверні середовища безпосередньо на робочій станції адміністратора, забезпечуючи при цьому достатній рівень продуктивності для тестових задач.

В якості платформи віртуалізації обрано Oracle VM VirtualBox (або альтернативно KVM/QEMU, залежно від хост-системи), керуючись наступними критеріями:

- Сумісність з інструментами IaC: Наявність офіційних та стабільних провайдерів для Terraform, що дозволяє керувати життєвим циклом віртуальних машин через код.

- Гнучкість мережевих налаштувань: Підтримка складних мережевих конфігурацій, включаючи NAT, мостові з'єднання (Bridged) та ізольовані мережі хоста (Host-only), що є критичним для моделювання корпоративної інфраструктури.

- Економічна ефективність: Відсутність ліцензійних витрат (Open Source), що відповідає вимогам до навчально-дослідних проєктів.

- Функціонал знімків стану (Snapshots): Можливість миттєвого створення контрольних точок відновлення, що є необхідним при відлагодженні сценаріїв

автоматизації Ansible, дозволяючи швидко повернути систему до "чистого" стану у разі помилки конфігурації.

2.1.2. Проєктування топології мережі та схема адресації

Архітектура тестового середовища побудована за принципом мінімально необхідної достатності та складається з ізольованого кластера віртуальних машин. Для забезпечення балансу між безпекою та доступністю до зовнішніх репозиторіїв ПЗ, мережева взаємодія організована через використання двох мережевих інтерфейсів на кожному вузлі.

Схема мережевих інтерфейсів:

- Інтерфейс 1 (NAT): Забезпечує вихід віртуальних машин до мережі Інтернет. Це необхідно для завантаження пакетів оновлень ОС, Docker-образів та бібліотек Terraform/Ansible. Цей інтерфейс ізолює VM від прямих вхідних з'єднань з зовнішньої мережі.

- Інтерфейс 2 (Host-Only Network): Створює приватну віртуальну локальну мережу між хостом (фізичною машиною) та гостьовими системами (VM). Саме через цей інтерфейс здійснюється управління інфраструктурою: SSH-з'єднання від Ansible, передача файлів та внутрішня комунікація між серверами кластера (наприклад, реплікація баз даних або трафік Kubernetes).

Такий підхід дозволяє жорстко прописати IP-адреси у конфігураційних файлах Ansible (inventory), уникаючи залежності від DHCP та забезпечуючи передбачуваність інфраструктури.

2.1.3. Характеристики та ролі віртуальних вузлів

Система складається з трьох логічних одиниць, кожна з яких виконує специфічну роль у процесі автоматизації. В якості операційної системи для вузлів обрано дистрибутиви Ubuntu Server 24.04 LTS, який поєднує стабільність ядра Linux з актуальними версіями пакетів програмного забезпечення, і CentOS Stream 10, який надає ранній доступ до технологій майбутніх версій Red Hat Enterprise Linux, забезпечуючи середовище для розробки та тестування, максимально наближене до промислових корпоративних стандартів.

Детальна специфікація вузлів:

1. Вузол управління (Control Node)

- Роль: Центральна консоль адміністратора. На цьому вузлі не виконуються бізнес-додатки; він слугує виключно для запуску коду інфраструктури.

- Ресурси: 2 vCPU, 2 GB RAM, 30 GB SSD.

- Програмне забезпечення: Packer (для розгортання VM), Ansible (для конфігурації), Git (контроль версій), Python 3 (середовище виконання). Генерування SSH-ключів для безпарольного доступу до керованих вузлів відбувається саме тут, Kubernetes (для контролю усіх worker node).

2. Керований вузол 2 (WebApp/K8s Backend Worker)

- Роль: Використовуватися для розміщення вебдодатка та бази даних

- Ресурси: 4 vCPU, 4 GB RAM, 40 GB SSD

2.2. Обґрунтування та аналіз інструментарію автоматизації

Ефективність побудови сучасної серверної інфраструктури безпосередньо залежить від вибору інструментальних засобів, що реалізують парадигму "Інфраструктура як Код" (IaC). Для виконання завдань дослідження було обрано гібридний підхід, що поєднує Packer для надання ресурсів (Provisioning) та Ansible для управління конфігураціями (Configuration Management). Такий розподіл дозволяє використовувати сильні сторони кожного інструменту, забезпечуючи модульність, гнучкість та ідемпотентність процесів розгортання.

2.2.1. Packer як інструмент оркестрації ресурсів

Packer[14] від компанії HashiCorp було обрано як основний інструмент для автоматизованого створення образів віртуальних машин. На відміну від Terraform, який спеціалізується на керуванні життєвим циклом інфраструктури (створення, зміна, видалення ресурсів), Packer фокусується на створенні ідентичних, попередньо налаштованих артефактів (образів дисків). Використання Packer дозволяє реалізувати підхід Immutable Infrastructure (незмінна інфраструктура), де

оновлення системи відбувається шляхом заміни всього образу, а не внесення змін у працюючу систему.

Ключові аспекти застосування Packer у проєкті:

1. Уніфікація конфігурації (HCL): Як і Terraform, Packer використовує мову HashiCorp Configuration Language (HCL). Це дозволяє описувати процес збірки образу (Build) у вигляді коду, що є зрозумілим для людини та підтримує модульність. Це забезпечує легкий перехід між інструментами екосистеми HashiCorp.

2. Непорушність середовища (Immutability): На відміну від класичного підходу Terraform, де налаштування відбувається *після* запуску ВМ (що займає час і може призвести до помилок під час виконання), Packer створює готовий знімок системи. Усі необхідні залежності, бібліотеки та конфігурації "запікаються" в образ на етапі збірки. Це гарантує, що віртуальна машина, запущена з цього образу в середовищі VirtualBox, поводитиметься ідентично до тестової, усуваючи проблему "it works on my machine".

3. Архітектура Builders та Provisioners: Packer чітко розділяє процес на етапи:

- Builders (Будівельники): відповідають за взаємодію з гіпервізором (у даному випадку VirtualBox/ISO) для створення "сирої" машини.
- Provisioners (Провіженери): дозволяють використовувати скрипти Shell або Ansible playbooks для налаштування системи всередині Packer. Це дозволяє повторно використовувати код Ansible ще на етапі створення образу.

4. Швидкість розгортання: Використання Packer дозволяє значно скоротити час готовності системи (Time-to-Live). Замість того, щоб чекати 10-15 хвилин, поки Terraform та Ansible встановлять програмне забезпечення на "чисту" ОС, ми отримуємо готову до роботи систему за лічені секунди після завантаження образу.

У розробленій системі Packer відповідає за підготовчий етап: автоматичне встановлення ОС (Ubuntu/CentOS), базове налаштування безпеки (SSH, користувачі) та створення оптимізованого .box або .ova файлу, який надалі використовується як шаблон для розгортання.

2.2.2. Ansible як засіб управління конфігураціями

Після того як Packer створює "чисті" віртуальні машини, в роботу вступає Ansible[15] від Red Hat. Його завдання — перетворити базову операційну систему на повноцінний сервер додатків, налаштувавши оточення відповідно до одного з трьох сценаріїв (Bare Metal, Docker, Kubernetes).

Технічні переваги Ansible для даного проєкту:

1. Без-агентна архітектура (Agentless): На відміну від аналогів (Puppet, Chef), Ansible не вимагає встановлення додаткового клієнтського ПЗ на цільові сервери. Взаємодія відбувається через стандартний протокол SSH, що значно спрощує початкове розгортання та зменшує споживання ресурсів на малопотужних віртуальних машинах.

2. Структура Playbooks та Ролей: Логіка автоматизації описується у форматі YAML. Для структурування проєкту використовується концепція Ролей (Roles) — незалежних модулів, кожен з яких відповідає за окремий сервіс (наприклад, роль docker для встановлення рушія контейнеризації, роль nginx для веб-сервера). Це дозволяє повторно використовувати код у різних сценаріях.

3. Ідемпотентність: Ansible гарантує, що багаторазове виконання одного й того ж сценарію (Playbook) не призведе до дублювання дій чи помилок. Якщо пакет вже встановлено або конфігураційний файл має правильний зміст, Ansible не вноситиме змін. Це забезпечує стабільність системи при частих оновленнях.

4. Модульність: Завдяки широкій бібліотеці модулів (apt, service, docker_container, k8s), Ansible дозволяє керувати як системними налаштуваннями Linux, так і об'єктами кластера Kubernetes без необхідності написання складних bash-скриптів.

2.2.3. Модель інтеграції та розмежування зон відповідальності

У проєкті реалізовано чітке розмежування сфер впливу інструментів, що відповідає кращим практикам DevOps. Взаємодія організована послідовно:

1. Етап 1 (Infrastructure Layer): Packer розгортає віртуальні ресурси. Після успішного створення ВМ, Packer автоматично генерує файл `inventory.ini` з актуальними IP-адресами створених машин.

2. Етап 2 (Application Layer): Ansible підключається до нових серверів, проводить базове налаштування безпеки (оновлення ядра, налаштування SSH) і розгортає необхідний стек технологій (на Bare Metal, Docker, Kubernetes).

Така архітектура дозволяє легко змінювати рівень реалізації сервісів (наприклад, перехід від Docker Compose до Kubernetes), не змінюючи код інфраструктури (Packer), або навпаки — масштабувати кількість серверів, не змінюючи логіку налаштування додатків (Ansible).

2.3. Моделювання сценаріїв автоматичного розгортання та конфігурації

Для всебічного дослідження ефективності обраного інструментарію (Packer та Ansible) було розроблено три експериментальні сценарії розгортання. Вони відображають різні рівні абстракції та складності керування інфраструктурою: від налаштування операційної системи на "голому залізі" (Bare Metal) до розгортання мікросервісів у кластері Kubernetes.

2.3.1. Сценарій 1: Автоматизація класичного серверного середовища

Цей сценарій моделює традиційний підхід до адміністрування, де додатки встановлюються безпосередньо в операційну систему сервера. Метою сценарію є створення базової точки відліку (baseline) для оцінки складності конфігурації.

Етап I: Ініціалізація інфраструктури (Packer) На цьому етапі Packer описує створення одного віртуального екземпляра Target Node. Конфігурація включає:

- Виділення ресурсів (CPU/RAM).
- Налаштування мережевого мосту для доступу по SSH.
- Генерацію `inventory`-файлу, що містить IP-адресу новоствореного хоста.

Етап II: Конфігурація та розгортання (Ansible) Для цього сценарію розроблено Ansible Playbook `deploy_bare_metal.yaml`, який виконує послідовне налаштування сервера. Логіка розбита на ролі:

1. Роль `common`: Виконує базові налаштування безпеки (створення користувача з `sudo`-правами, налаштування SSH-ключів, оновлення кешу пакетного менеджера `apt`).

2. Роль `app_deployment`:

- Клонує репозиторій проєкту та встановлює системні залежності (Golang, `build-essential`).
- Розгортає та ініціалізує базу даних MongoDB.
- Налаштовує середовище Node.js через NVM, збирає бекенд та забезпечує автозапуск усіх сервісів за допомогою менеджера процесів PM2.

Результат: Повнофункціональний веб-сервер, налаштований без ручного втручання, готовий до обслуговування трафіку.

2.3.2. Сценарій 2: Контейнеризація додатків засобами Docker

Цей сценарій демонструє сучасний підхід до ізоляції додатків. Замість прямого встановлення залежностей в ОС, додаток розгортається у легкому контейнері, що забезпечує портативність та вирішує проблему конфлікту версій бібліотек ("Dependency Hell").

Етап I: Підготовка хоста (Packer). Аналогічно до першого сценарію, Terraform розгортає віртуальну машин Target Node. Однак вимоги до ОС мінімальні, оскільки основне навантаження буде ізольоване в контейнерах.

Етап II: Налаштування середовища виконання (Ansible Role: `docker_engine`)
Ansible виконує підготовку хост-системи:

- Встановлення необхідних утиліт (`ca-certificates`, `curl`, `gnupg`).
- Додавання офіційного GPG-ключа та репозиторію Docker.
- Інсталяція пакетів `docker-ce`, `docker-ce-cli`, `containerd.io`.
- Забезпечення автозапуску служби Docker (`systemd`).

Етап III: Розгортання контейнера (Ansible Playbook: `deploy_docker.yaml`).
Замість копіювання файлів сайту, Ansible взаємодіє з демоном Docker через Python-модуль `community.docker`. Процес включає:

1. Збірка (Build) або Завантаження (Pull): Отримання актуального образу веб-сервера (наприклад, `nginx:alpine`) або збірка власного образу з локального `Dockerfile`.

2. Запуск контейнера: Створення контейнера з визначеними параметрами:

- `ports`: Прокидання порту 8080 хоста на порт 8080 контейнера.
- `volumes`: Монтування локальної директорії з конфігурацією в контейнер.
- `restart_policy`: Налаштування `unless-stopped` для автоматичного перезапуску при збоях.

Результат: Додаток працює в ізольованому середовищі, що спрощує його оновлення та перенесення на інший сервер.

2.3.3. Сценарій 3: Оркестрація кластера Kubernetes (K8s)

Цей сценарій моделює розгортання легковагового, але повністю сертифікованого CNCF кластера Kubernetes на базі K3s[16] від Rancher. На відміну від важнагових рішень на кшталт `kubeadm`, K3s оптимізований для роботи в середовищах з обмеженими ресурсами (Edge/IoT/Dev), постачається у вигляді єдиного бінарного файлу та суттєво спрощує архітектуру, об'єднуючи компоненти Control Plane в один процес.

Етап I: Підготовка інфраструктури (Packer та Host). Оскільки K3s має значно нижчі системні вимоги, конфігурація обладнання є більш гнучкою. Архітектура кластера є гібридною:

1. K3s Server (Control Plane): Розгортається безпосередньо на хост-машині (Ubuntu). Це дозволяє ефективно використовувати наявні ресурси без оверхеду на віртуалізацію для керуючого шару.

2. K3s Agent (Worker Node): Розгортається на віртуальній машині CentOS, підготовленій за допомогою Packer. Для цього вузла буде виділено 2 vCPU та 4 GB RAM

Етап II: Базове налаштування вузлів (Ansible Role: `k3s_common`). Перед встановленням Ansible виконує підготовку різnorідних середовищ (Ubuntu Host та CentOS VM) до роботи в єдиній мережі:

- Встановлення залежностей: K3s не потребує попереднього встановлення Docker або containerd, оскільки містить оптимізоване середовище виконання (containerd) всередині себе. Ansible встановлює лише базові утиліти (curl, git).

Етап III: Бутстрапінг Control Plane (Ansible Role: k3s_server). На цьому етапі відбувається ініціалізація кластера на хост-машині. Ansible виконує:

1. Інсталяція: Запуск скрипта встановлення (curl -sL https://get.k3s.io | sh -). Скрипт автоматично налаштовує systemd-сервіс.

2. Вбудовані компоненти: На відміну від kubeadm, K3s автоматично розгортає CNI-плагін (Flannel) для мережі подів та Ingress-контролер (Traefik) для маршрутизації вхідного трафіку, що усуває необхідність їх ручного встановлення.

3. Експорт конфігурації: Отримання файлу /etc/rancher/k3s/k3s.yaml та налаштування доступу для поточного користувача хоста, щоб керувати кластером без sudo.

4. Отримання токена: Зчитування унікального токена вузла (node-token), який необхідний для приєднання воркерів.

Етап IV: приєднання робочого вузла (Ansible Role: k3s_agent). Інтеграція віртуальної машини CentOS у кластер. Ansible підключається до VM і виконує команду установки з передачею параметрів оточення:

- K3S_URL: IP-адреса хост-машини (Control Plane).
- K3S_TOKEN: Токен, отриманий на попередньому етапі. Це реєструє CentOS VM як повноцінний робочий вузол, на якому запускається процес k3s-agent.

Етап V: Декларативне розгортання додатків. Після об'єднання хоста і VM у кластер, Ansible демонструє роботу платформи:

- Deployment: Розгортання веб-сервера або мікросервісу. Планувальник K3s автоматично розміщує поди на доступному вузлі (CentOS VM або Host, залежно від налаштувань).

- Ingress: Використання вбудованого Traefik для доступу до додатку через доменне ім'я або IP-адресу хоста, без необхідності налаштування складних LoadBalancer-ів.

Результат: Сценарій демонструє побудову гібридного Kubernetes-кластера з мінімальним споживанням ресурсів, поєднуючи зручність розробки на локальному хості з ізоляцією робочих навантажень у віртуальному середовищі CentOS.

Висновок до другого розділу:

У другому розділі було здійснено комплексне проєктування архітектури системи автоматизованого управління корпоративною інфраструктурою. Для реалізації безпечного та контрольованого середовища моделювання було обрано платформу віртуалізації Oracle VM VirtualBox, яка дозволяє розгорнути ізольовані вузли безпосередньо на робочій станції адміністратора. Ключовим елементом проєктування стала розробка мережевої топології, що базується на використанні двох інтерфейсів: NAT для доступу до зовнішніх репозиторіїв та Host-Only для забезпечення захищеного внутрішнього каналу управління між вузлами.

Особливу увагу в розділі приділено обґрунтуванню вибору інструментальних засобів реалізації підходу «Інфраструктура як код» (IaC). В якості засобу оркестрації ресурсів визначено HashiCorp Packer, який дозволяє реалізувати концепцію незмінної інфраструктури шляхом створення уніфікованих образів віртуальних машин. Для управління конфігураціями обрано Ansible, головними перевагами якого є без-агентна архітектура та здатність забезпечувати ідемпотентність процесів налаштування.

Для практичного дослідження ефективності запропонованих рішень було спроектовано гетерогенне середовище на базі операційних систем Ubuntu Server та CentOS Stream 10. В рамках роботи розроблено три експериментальні моделі розгортання, що покривають різні рівні абстракції: від класичного налаштування серверів (Bare Metal) і контейнеризації додатків засобами Docker до побудови гібридного кластера оркестрації Kubernetes (K3s). Такий підхід забезпечує гнучкість системи та можливість її адаптації під різні виробничі потреби.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ СИСТЕМИ АВТОМАТИЗОВАНОГО УПРАВЛІННЯ

3.1. Реалізація сценарію автоматизації класичного серверного середовища

У першому сценарії дослідження реалізовано підхід "Infrastructure as Code" для розгортання монолітної архітектури веб-додатка безпосередньо в операційну систему. Цей процес розділено на дві фази: підготовка образу віртуальної машини (Provisioning) та конфігурація програмного забезпечення (Configuration Management).

3.1.1. Підготовка образу операційної системи засобами Packer

Для створення уніфікованого образу сервера було використано інструмент HashiCorp Packer із плагіном virtualbox-iso. Конфігурація описана у файлі ubuntu.pkr.hcl(Додаток А). Процес побудови віртуальної машини (VM) включає наступні кроки:

1. Визначення апаратних ресурсів: У блоці source задано параметри VM, які відповідають вимогам для розгортання повного стека додатків (Backend + Frontend + DB):

- Процесор (CPUs): 4 ядра
- Оперативна пам'ять (Memory): 4096 МБ.
- Дисковий простір: 40 ГБ.

2. Налаштування мережевих інтерфейсів: Для забезпечення керування та доступу до Інтернету налаштовано два мережеві адаптери:

- nic1 (NAT): для завантаження пакетів із зовнішніх репозиторіїв.
- nic2 (Host-Only): для підключення з хост-машини та Ansible по внутрішній мережі.

3. Автоматизація інсталяції OS: Використано механізм autoinstall Ubuntu. Команда завантаження (boot_command) передає ядру Linux параметри для отримання конфігурації user-data через вбудований HTTP-сервер Packer (http://192.168.56.1:{{ .HTTPEndPoint }}).

Ключовим елементом налаштування є файл user-data(Додаток В), який виконує "тиху" установку системи (cloud-init). У ньому визначено:

- Мережева адресація: Статична IP-адреса 192.168.56.101/24 на інтерфейсі enp0s8, що дозволяє Ansible мати фіксовану точку входу.
- Ідентифікація та доступ: Створення користувача takarov та додавання публічного SSH-ключа (ssh-ed25519...) у список авторизованих ключів. Це критично важливо для подальшої роботи Ansible без паролів.
- Фіналізація: Після завершення установки система повідомляє Packer про успіх через створення файлу-маркера /target/etc/packer_build_info.

Запуск процесу збірки ініціюється командою: packer build ubuntu.pkr.hcl

```
==> virtualbox-iso.ubuntu (shell-local):
==> virtualbox-iso.ubuntu (shell-local): E:\All_LB\???\packer\ubuntu>
VBoxManage startvm Ubuntu_Server --type gui
==> virtualbox-iso.ubuntu (shell-local): Waiting for VM "Ubuntu_Server" to power
on...
==> virtualbox-iso.ubuntu (shell-local): VM "Ubuntu_Server" has been successfull
y started.
Build 'virtualbox-iso.ubuntu' finished after 30 minutes 24 seconds.

==> Wait completed after 30 minutes 24 seconds

==> Builds finished. The artifacts of successful builds are:
--> virtualbox-iso.ubuntu: VM files in directory: E:\VirtualBox VMs\Ubuntu_Serve
r
--> virtualbox-iso.ubuntu: VM files in directory: E:\VirtualBox VMs\Ubuntu_Serve
r

Alex@DESKTOP-08S00BT MINGW64 /e/All_LB/магістр/Диплом/packer/ubuntu
$ packer build -force .|
```

Рисунок 3.1 – Результат виконання команди packer

Після завершення роботи Packer у директорії виводу створюється готова віртуальна машина, зареєстрована у VirtualBox, яка готова до прийому SSH-з'єднань на адресі 192.168.56.101.

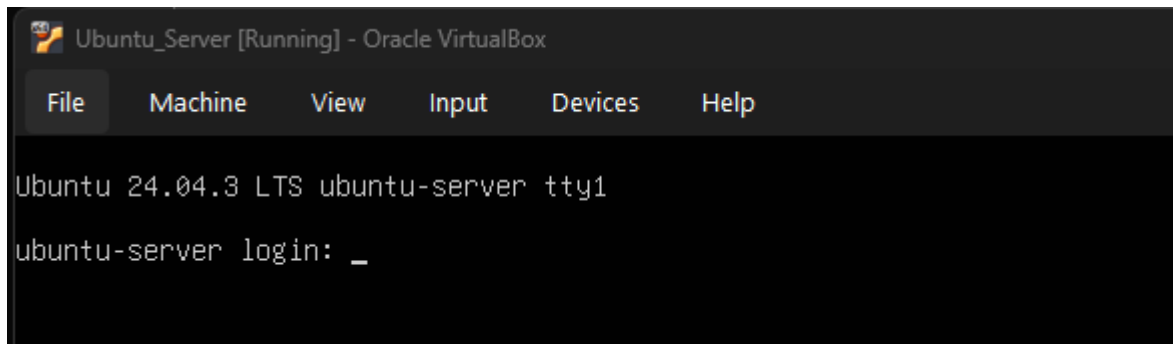


Рисунок 3.2 – Створена віртуальна машина

3.1.2. Налаштування конфігурації сервера веб-додатка за допомогою Ansible

Після підняття віртуальної машини, роль налаштування програмного оточення бере на себе Ansible. Розроблений плейбук `server_ubuntu.yml` виконує декларативний опис цільового стану сервера для розгортання додатку "Company Employees".

Процес налаштування включає такі етапи:

1. Встановлення системних залежностей: Першим завданням (Install basic system packages) встановлюється базовий набір інструментів: `git`, `curl`, компілятор `golang-go` та середовище `nodejs/npm`. Використовується модуль `apt` з оновленням кешу пакетів.

2. Налаштування бази даних MongoDB: Оскільки в офіційних репозиторіях Ubuntu часто містяться застарілі версії, плейбук підключає офіційний репозиторій MongoDB 8.0:

- Імпорт GPG-ключа безпеки (`apt_key`).
- Додавання репозиторію у `sources.list.d`.
- Встановлення пакету `mongodb-org` та запуск сервісу `mongod`.

3. Розгортання коду додатку: Модуль `git` клонує репозиторій проєкту з GitHub у директорію `/home/makarov/company_employees`. Важливо, що ця дія виконується від імені користувача (`become: no`), щоб уникнути проблем із правами доступу до файлів.

4. Ініціалізація даних: За допомогою модуля `shell` виконується скрипт `mongo.init`, який наповнює базу даних початковою структурою (колекція `programmers`).

5. Збірка та запуск Backend (Golang): Серверна частина компілюється командою `go build`. Для керування процесом використовується менеджер процесів `PM2`, який встановлюється глобально через `npm`. Запуск бекенду здійснюється командою: `pm2 start ./app --name backend`.

6. Налаштування Frontend (Node.js): Для забезпечення сумісності версій використовується `NVM` (Node Version Manager). Оскільки `Ansible` працює у неінтерактивному шелі, завдання реалізовано через блок `shell`, який встановлює `NVM`, активує `Node.js` версії 12.14, встановлює залежності (`npm install`) та запускає фронтенд через `PM2`.

7. Забезпечення відмовостійкості: Фінальні кроки генерують та активують `systemd`-скрипт для `PM2` (`pm2 startup`), що гарантує автоматичний перезапуск додатків (`backend` та `frontend`) після перезавантаження сервера.

Запуск конфігурації виконується командою: `ansible-playbook -i /etc/ansible/hosts server_ubuntu.yml -u Makarov`

```
PLAY [Configure Ubuntu Server for Company Employees App] *****

TASK [Gathering Facts] *****
[WARNING]: Host 'server_scene1' is using the discovered Python interpreter at '/usr/bin/python3.12', but future installation of another Python interpreter could cause a different interpreter to be discovered. See https://docs.ansible.com/ansible-core/2.19/reference_appendices/interpreter_discovery.html for more information.
ok: [server_scene1]

TASK [Install basic system packages] *****
changed: [server_scene1]

TASK [Import MongoDB GPG key] *****
changed: [server_scene1]

TASK [Add MongoDB Repository] *****
changed: [server_scene1]

TASK [Install MongoDB] *****
changed: [server_scene1]

TASK [Ensure MongoDB is running and enabled] *****
changed: [server_scene1]

TASK [Clone the repository] *****
changed: [server_scene1]

TASK [Initialize MongoDB Data] *****
changed: [server_scene1]

TASK [Install PM2 globally] *****
changed: [server_scene1]

TASK [Build Go Backend] *****
changed: [server_scene1]

TASK [Start Backend with PM2] *****
changed: [server_scene1]

TASK [Install NVM and Frontend Dependencies] *****
changed: [server_scene1]

TASK [Generate PM2 Startup Script] *****
changed: [server_scene1]

TASK [Save PM2 list] *****
changed: [server_scene1]

PLAY RECAP *****
server_scene1 : ok=14 changed=13 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

ale@DESKTOP-08500BT:~$
```

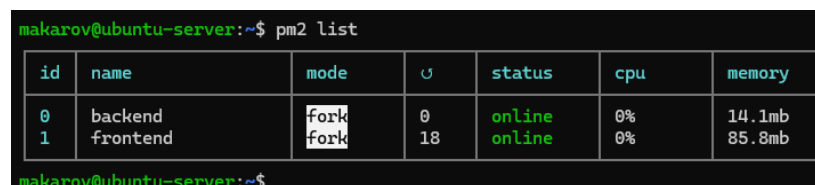
Рисунок 3.3 – Результат роботи ansible

3.1.3. Перевірка працездатності та відмовостійкості розгорнутого рішення

Після завершення роботи Ansible було проведено перевірку функціональності розгорнутої системи:

1. Перевірка статусів сервісів: Підключення до сервера по SSH підтвердило, що менеджер процесів PM2 успішно керує двома мікросервісами:

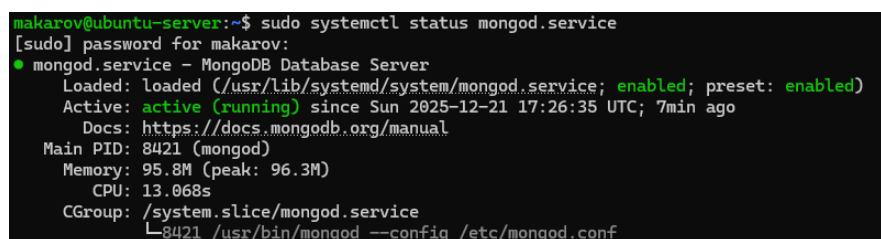
- backend: статус online
- frontend: статус online



id	name	mode	⚙	status	cpu	memory
0	backend	fork	0	online	0%	14.1mb
1	frontend	fork	18	online	0%	85.8mb

Рисунок 3.4 – Перевірка роботи frontend і backend сервісів

2. Перевірка бази даних: Сервіс MongoDB (mongod) активний (Active: active (running)) і слухає порт 27017. Дані успішно імпортовано.



```
makarov@ubuntu-server:~$ sudo systemctl status mongod.service
[sudo] password for makarov:
● mongod.service - MongoDB Database Server
   Loaded: loaded (/usr/lib/systemd/system/mongod.service; enabled; preset: enabled)
   Active: active (running) since Sun 2025-12-21 17:26:35 UTC; 7min ago
     Docs: https://docs.mongodb.org/manual
    Main PID: 8421 (mongod)
      Memory: 95.8M (peak: 96.3M)
         CPU: 13.068s
    CGroup: /system.slice/mongod.service
            └─8421 /usr/bin/mongod --config /etc/mongod.conf
```

Рисунок 3.5 – Перевірка роботи MongoDB



```
makarov@ubuntu-server:~$ mongo
Current Mongosh Log ID: 69483006577089a1c8de665
Connecting to:  mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.10
Using MongoDB:  8.0.17
Using Mongosh:  2.5.10
test> use programmers
switched to db programmers
programmers> show collections
programmers
programmers> db.programmers.find()
[
  {
    _id: ObjectId('69482dda3af5b418dd8de666'),
    name: 'Ann Lee',
    title: 'Full Stack Developer',
    company: 'Google',
    picture: 'https://images.pexels.com/photos/415829/pexels-photo-415829.jpeg?auto=compress&cs=tinysrgb&dpr=1&w=500',
    skills: [
      { name: 'JavaScript', icon: 'js', importance: 5 },
      { name: 'Java', icon: 'java', importance: 0 },
      { name: 'Node.js', icon: 'node-js', importance: 5 },
      { name: 'Express', icon: 'express', importance: 0 },
      { name: 'Angular', icon: 'angular', importance: 1 },
      { name: 'HTML', icon: 'html5', importance: 1 },
      { name: 'CSS', icon: 'css3-alt', importance: 1 }
    ]
  }
]
```

Рисунок 3.6 – Перевірка наявності даних в MongoDB

3. Доступність веб-інтерфейсу: При зверненні через браузер за адресою `http://192.168.56.101:3000` (порт фронтенду) відображається головна сторінка додатку "Company Employees", яка успішно отримує дані з бекенду.

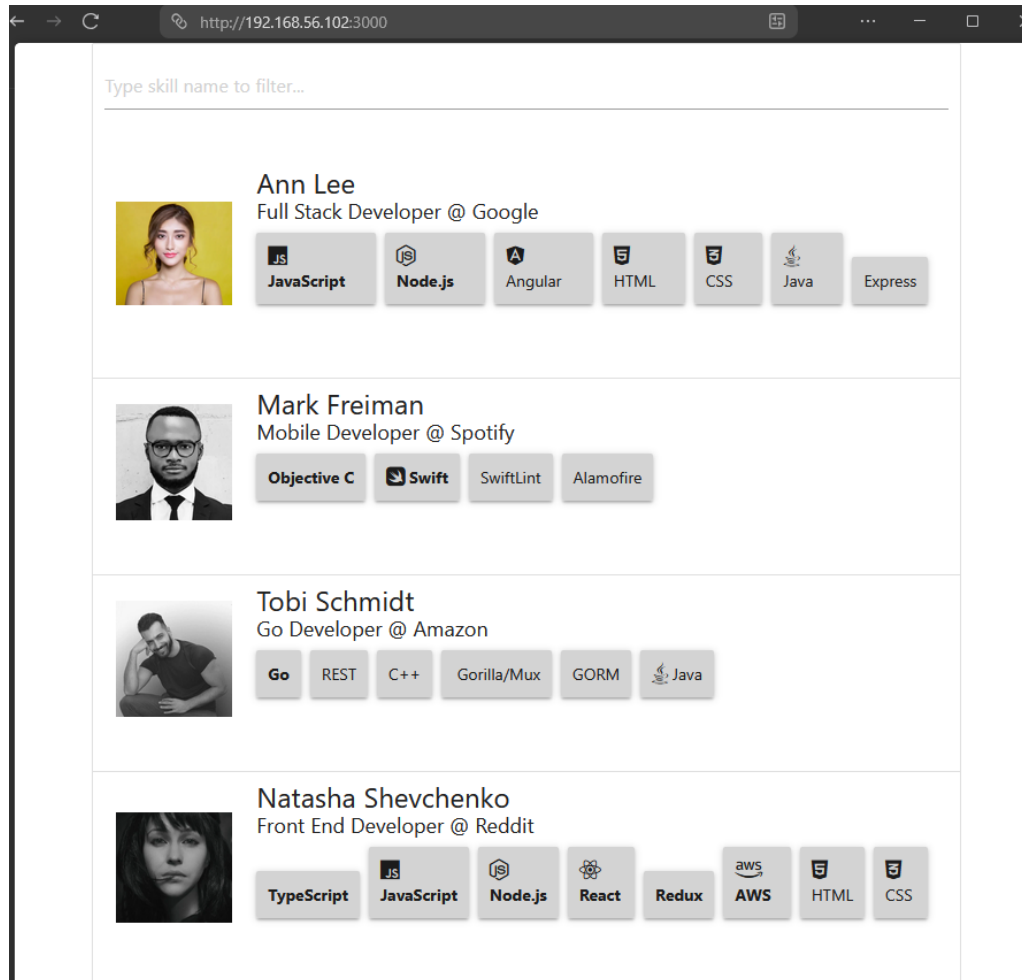


Рисунок 3.7 – Перевірка доступу до веб-сервісу

Таким чином, сценарій Bare Metal продемонстрував повну автоматизацію розгортання: від "чистого заліза" (через Packer) до працюючого бізнес-додатку (через Ansible), усуваючи необхідність ручного налаштування.

3.2. Розгортання контейнеризованої інфраструктури на базі Docker

Другий сценарій дослідження присвячений реалізації концепції контейнеризації. На відміну від попереднього підходу (Bare Metal), де всі залежності встановлювалися безпосередньо в операційну систему, тут основна система залишається "чистою". Вона виконує роль хоста для Docker Engine, а всі

компоненти додатку (Frontend, Backend, Database) розгортаються в ізольованих контейнерах.

3.2.1. Підготовка образу операційної системи засобами Packer

В якості базової операційної системи для цього сценарію було обрано CentOS Stream 10. Процес створення образу автоматизовано за допомогою Packer файлу `centos_docker.pkr.hcl` (Додаток А) у поєднанні з файлом конфігурації автоматичного встановлення операційної системи `ks.cfg` (Додаток В).

Процес підготовки інфраструктури (Packer): Конфігурація Packer використовує будівельник `virtualbox-iso` і передбачає наступні параметри:

- Ресурси: Для стабільної роботи Docker-хоста виділено 4 vCPU та 4096 МБ RAM.
- Завантаження: Використовується прошивка EFI (`firmware = "efi"`), що відповідає сучасним стандартам серверного обладнання.
- Автоматизація інсталяції: Packer запускає вбудований HTTP-сервер для передачі файлу конфігурації `ks.cfg` безпосередньо в інсталятор Anaconda під час завантаження ядра Linux.

Налаштування операційної системи (Kickstart): Файл `ks_kube.cfg` визначає критичні параметри системи ще до першого входу користувача:

- Мережева конфігурація: Налаштовано два інтерфейси. `enp0s3` отримує адресу по DHCP (NAT), а для `enp0s8` жорстко задано статичну IP-адресу 192.168.56.102.
- Безпека та доступ: Створено користувача `makarov` із правами адміністратора (`groups=wheel`). Важливим кроком є ін'єкція публічного SSH-ключа (`ssh-ed25519...`) на етапі інсталяції, що дозволяє Ansible підключатися до сервера одразу після його створення.
- Системні політики: Для спрощення мережевої взаємодії контейнерів на етапі розробки вимкнено `firewall`, але залишено активним SELinux у режимі `enforcing`.

Запуск збірки образу виконується командою: `packer build centos_docker.pkr.hcl`

```
Alex@DESKTOP-08500BT MINGW64 /e/All_LB/магістр/Диплом/packer/centos
$ packer build -force centos_docker.pkr.hcl
virtualbox-iso.centos: output will be in this color.

==> virtualbox-iso.centos: Retrieving Guest additions
==> virtualbox-iso.centos: Trying C:\Program Files\Oracle\VirtualBox\VBBoxGuestAdditions.i
so
==> virtualbox-iso.centos: Trying file://C:/Program%20Files/Oracle/VirtualBox/VBoxGuestAd
ditions.iso
==> virtualbox-iso.centos: file://C:/Program%20Files/Oracle/VirtualBox/VBoxGuestAdditions
.iso => C:/Program Files/Oracle/VirtualBox/VBoxGuestAdditions.iso
==> virtualbox-iso.centos: Retrieving ISO
==> virtualbox-iso.centos: Trying E:\Downloads\CentOS-Stream-10-latest-x86_64-dvd1.iso
==> virtualbox-iso.centos: Trying file://E:/Downloads/CentOS-Stream-10-latest-x86_64-dvd1
.iso?checksum=sha256%3Ada759c3fdec79ede1b9ebf921c72495b688ece424bcc890f5c7ce01b0d6909c5
==> virtualbox-iso.centos: file://E:/Downloads/CentOS-Stream-10-latest-x86_64-dvd1.iso?ch
ecksum=sha256%3Ada759c3fdec79ede1b9ebf921c72495b688ece424bcc890f5c7ce01b0d6909c5 => E:/Do
wnloads/CentOS-Stream-10-latest-x86_64-dvd1.iso
==> virtualbox-iso.centos: Starting HTTP server on port 9624
==> virtualbox-iso.centos: Creating virtual machine...
==> virtualbox-iso.centos: Creating hard drive E:\VirtualBox VMs\CentOS_Server\CentOS_Doc
```

Рисунок 3.8 – Виконання команди packer

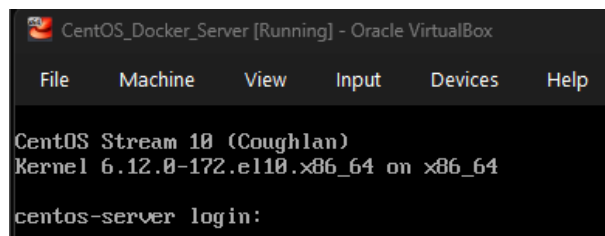


Рисунок 3.9 – Створена віртуальна машина

3.2.2. Побудова та запуск контейнерів додатку через Ansible-плейбуки

Після успішного створення віртуальної машини CentOS_Server, розгортання середовища виконується за допомогою Ansible плейбука `server_centos_docker.yml` (Додаток В). Логіка плейбука суттєво відрізняється від сценарію Bare Metal: замість компіляції коду та налаштування сервісів, Ansible готує систему до запуску оркестратора контейнерів.

Етапи виконання плейбука:

1. Очищення системи: Перший крок гарантує відсутність конфліктів шляхом видалення старих версій Docker (`docker`, `docker-engine`, `docker-client`) за допомогою пакетного менеджера `dnf`.

2. Встановлення Docker Engine: Ansible додає офіційний репозиторій Docker CE (`download.docker.com`) та встановлює сучасний набір інструментів:

- `docker-ce` (середовище виконання);

- containerd.io (управління контейнерами);
- docker-compose-plugin

3. Налаштування Docker Compose: Для забезпечення сумісності скрипт додатково завантажує бінарний файл docker-compose версії v2.38.2 безпосередньо з GitHub у директорію /usr/local/bin. Це дозволяє використовувати як команду docker compose, так і docker-compose.

4. Розгортання проекту: Клонування репозиторію company_employees у домашню директорію користувача, а потім Запуск додатку однією командою: docker compose up -d.

Команда “docker compose up -d” зчитує файл docker-compose.yml (який знаходиться в репозиторії) і автоматично:

- Завантажує необхідні образи (MongoDB, Node.js, Golang).
- Запускає контейнери у фоновому режимі.

Запуск автоматизації: ansible-playbook -i /etc/ansible/hosts server_centos_docker.yml -u makarov

```
alex@DESKTOP-0859081:~$ ansible-playbook -i /etc/ansible/hosts server_centos_docker.yml -u makarov -K
BECOME password:

PLAY [Configure CentOS Server for Docker App] *****

TASK [Gathering Facts] *****
[WARNING]: Host 'server_scene2' is using the discovered Python interpreter at '/usr/bin/python3.12', but future installation of another Python interpreter could cause a different interpreter to be discovered. See https://docs.ansible.com/ansible-core/2.19/reference_appendices/interpreter_discovery.html for more information.
ok: [server_scene2]

TASK [Remove old Docker packages] *****
ok: [server_scene2]

TASK [Install DNF plugins and Git] *****
changed: [server_scene2]

TASK [Add Docker CE Repository] *****
changed: [server_scene2]

TASK [Install Docker Engine and Plugins] *****
changed: [server_scene2]

TASK [Ensure Docker service is running] *****
changed: [server_scene2]

TASK [Install Docker Compose Standalone] *****
changed: [server_scene2]

TASK [Clone the repository] *****
changed: [server_scene2]

TASK [Run Docker Compose Up] *****
changed: [server_scene2]

PLAY RECAP *****
server_scene2      : ok=9    changed=7    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Рисунок 3.10 – Результат роботи ansible

3.2.3. Перевірка роботи контейнерів

Завершальним етапом реалізації другого сценарію стала верифікація працездатності розгорнутої інфраструктури. На відміну від традиційного моніторингу процесів, тестування контейнеризованого середовища вимагає перевірки не лише доступності сервісів, а й коректності їх ізоляції від хост-системи.

Перевірка статусу контейнерів та оркестрації Для контролю успішного запуску мікросервісів буде використано інструментарій CLI Docker.

```
alex@DESKTOP-08S00BT:~$ ssh makarov@192.168.56.102
Last login: Sun Dec 21 20:43:58 2025 from 192.168.56.1
[makarov@centos-server ~]$ sudo docker ps
[sudo] password for makarov:
CONTAINER ID   IMAGE                  COMMAND                  CREATED        STATUS        PORTS                               NAMES
85372fe94da1   company_employees-app  "/bin/sh -c ./server"   7 minutes ago  Up 7 minutes  0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp  app
f5e7d5ce25ad   mongo:4.2.2           "docker-entrypoint.s..." 7 minutes ago  Up 7 minutes  27017/tcp                             db
[makarov@centos-server ~]$
```

Рисунок 3.11 – Перевірка контейнерів

Команда `docker ps` підтвердила, що після виконання Ansible-задачі `Run Docker Compose Up` у системі було ініціалізовано два незалежні контейнери, що відповідають архітектурі додатку:

- `app`: Серверна частина, що обробляє API-запити.
- `db`: Сховище даних.

Підтвердження ізоляції середовища Однією з ключових цілей сценарію було уникнення "забруднення" основної операційної системи залежностями. Для перевірки цього було проведено спробу виклику компонентів додатку безпосередньо з консолі хоста CentOS:

```
[makarov@centos-server ~]$ node -v
-bash: node: command not found
[makarov@centos-server ~]$ go version
-bash: go: command not found
[makarov@centos-server ~]$ mongosh
-bash: mongosh: command not found
```

Рисунок 3.12 – Спроба виконати команди

Це підтверджує, що операційна система залишається "чистою", містячи лише Docker Engine та системні утиліти, визначені у конфігурації Packer. Такий підхід

унеможлиблює конфлікти версій бібліотек, які часто виникають при Bare Metal розгортанні.

Функціональне тестування мережевого доступу Доступ до веб-додатку здійснювався через статичну IP-адресу 192.168.56.103, яка була жорстко задана у файлі конфігурації Kickstart для інтерфейсу enp0s8. Тестування показало, що Docker Проху коректно маршрутизує вхідний трафік:

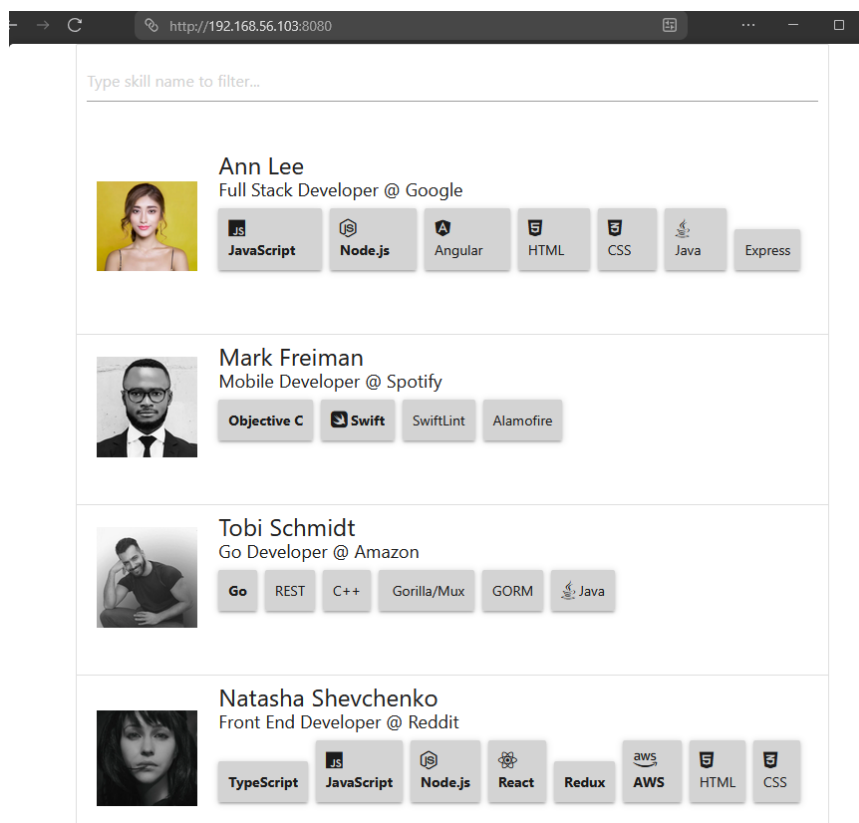


Рисунок 3.13 – Перевірка доступу до контейнера

Таким чином, використання Docker на базі CentOS дозволило досягти повної ізоляції компонентів системи, спростити керування залежностями та забезпечити передбачувану поведінку додатку незалежно від налаштувань хост-системи.

3.3. Організація та оркестрація кластера Kubernetes

Третій, найбільш комплексний сценарій дослідження, передбачає створення кластерної архітектури для оркестрації контейнерів. Для реалізації було обрано дистрибутив K3s — легковагову сертифіковану версію Kubernetes. Архітектура рішення є гібридною: роль керуючого вузла (Control Plane) виконує хост-система

(WSL/Ubuntu), а роль робочого вузла (Worker Node) — віртуальна машина на базі CentOS Stream 10.

Такий підхід дозволяє оптимізувати використання ресурсів, уникаючи витрат на віртуалізацію для компонентів управління.

3.3.1. Підготовка образу операційної системи засобами Packer

Першочерговим завданням є створення віртуальної машини, яка виконуватиме роль робочого вузла (Worker Node). Оскільки Ansible потребує вже працюючого сервера для підключення, цей етап виконується першим за допомогою інструменту Packer.

Конфігурація описана у файлі `centos_kuber.pkr.hcl` (Додаток В). Вона базується на автоматичній інсталяції ОС CentOS Stream 10 через механізм Kickstart.

Виділення апаратних ресурсів: Для забезпечення стабільної роботи контейнерів у віртуальній машині закладено підвищені системні вимоги:

- CPU: 4 віртуальних ядра.
- RAM: 4096 МБ оперативної пам'яті.
- Disk: 40 ГБ дискового простору.

Налаштування операційної системи (Kickstart): Файл відповідей `ks_kube.cfg` (Додаток В) автоматизує налаштування системи ще до її першого запуску. Ключові параметри включають:

- Мережева ідентифікація: Віртуальній машині примусово присвоюється статична IP-адреса 192.168.56.103 на інтерфейсі `enp0s8` (Host-Only мережа). Це критична вимога архітектури: Ansible повинен мати фіксовану, відому заздалегідь адресу для підключення та додавання вузла в кластер.

- Підготовка безпеки: Скрипт автоматично створює користувача `makarov`, додає його до групи `wheel` (`sudo`) та ін'єктує публічний SSH-ключ адміністратора. Це дозволяє уникнути ручного введення паролів при подальшому використанні Ansible.

- Оптимізація для Kubernetes: У конфігурації явно вимкнено системний фаєрвол (`firewall --disable`). Це необхідно для коректної роботи CNI-плагіна

(Flannel), який використовує VXLAN-тунелі для зв'язку між подами. Блокування цього трафіку фаєрволом є найпоширенішою причиною несправності кластера.

Запуск збірки образу виконується командою: `packer build centos_docker.pkr.hcl`

```
Alex@DESKTOP-D8S00BT MINGW64 /e/All_LB/maістр/Диплом/packer/centos
$ packer build -force centos_kuber.pkr.hcl
virtualbox-iso.centos: output will be in this color.

==> virtualbox-iso.centos: Retrieving Guest additions
==> virtualbox-iso.centos: Trying C:\Program Files\Oracle\VirtualBox\VBxGuestAdditions.i
so
==> virtualbox-iso.centos: Trying file://C:/Program%20Files/Oracle/VirtualBox/VBoxGuestAd
ditions.iso
==> virtualbox-iso.centos: file://C:/Program%20Files/Oracle/VirtualBox/VBoxGuestAdditions
.iso => C:/Program Files/Oracle/VirtualBox/VBoxGuestAdditions.iso
==> virtualbox-iso.centos: Retrieving ISO
==> virtualbox-iso.centos: Trying E:\Downloads\CentOS-Stream-10-latest-x86_64-dvd1.iso
==> virtualbox-iso.centos: Trying file://E:/Downloads/CentOS-Stream-10-latest-x86_64-dvd1
.iso?checksum=sha256%3Ada759c3fdec79edeb9ebf921c72495b688ece424bcc890f5c7ce01b0d6909c5
```

Рисунок 3.14 – Виконання команди packer

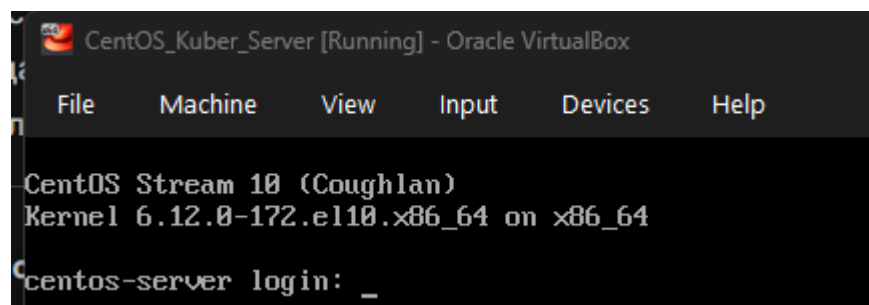


Рисунок 3.15 – Створена віртуальна машина

3.3.2. Ініціалізація та налаштування кластера

Після того, як інфраструктура створена, запускається Ansible-плейбук `server_centos_kuber.yml`, який об'єднує хост-систему та нову віртуальну машину в єдиний кластер. Процес складається з трьох послідовних "п'єс" (Plays).

Перша частина плейбука виконується на хості (hosts: master). Її мета — запустити керуючі компоненти K3s та підготувати дані для підключення воркерів, а саме:

1. Інсталяція: Виконується офіційний скрипт встановлення: `curl -sfL https://get.k3s.io | sh -.`

2. Конфігурація доступу: Файл `/etc/rancher/k3s/k3s.yaml` копіюється у домашню директорію користувача (`~/kube/config`) з правами `0600`, що дозволяє керувати кластером без `sudo`.

3. Експорт секретів: Ansible автоматично зчитує Token кластера (з файлу `/var/lib/rancher/k3s/server/node-token`) та визначає IP-адресу Майстра у внутрішній мережі. Ці змінні зберігаються в пам'яті (`hostvars`) для передачі на наступний крок.

Інтеграція Worker Node Друга частина плейбука підключається до створеної Packer'ом віртуальної машини (`hosts: server_scene3`, IP: `192.168.56.103`), а саме:

1. Перевірка зв'язку: Модуль `wait_for` перевіряє доступність порту `6443` на Майстрі. Це гарантує, що API-сервер готовий приймати запити.

2. Приєднання до кластера: Запускається агент K3s із спеціальними параметрами: `K3S_URL=https://{ master_ip }:6443 K3S_TOKEN={{ token }}` `sh -s - --flannel-iface enp0s8`.

Фінальна частина плейбука налаштовує логіку роботи додатків у кластері, а саме:

1. Ізоляція Майстра (Tainting): Виконується команда `kubectrl taint nodes ... NoSchedule`. Це забороняє планувальнику розміщувати бізнес-додатки на керуючому вузлі, примусово направляючи їх на віртуальну машину CentOS.

2. Запуск додатку: Ansible клонує репозиторій коду, створює конфігурацію бази даних (`ConfigMap`) та застосовує маніфести (`kubectrl apply -f ./webapp.yaml`), розгортаючи мікросервіси `app` та `db`.

Запуск автоматизації: `ansible-playbook -i /etc/ansible/hosts server_centos_docker.yml -u makarov`

```
alex@DESKTOP-08S00BT:~$ ansible-playbook -i /etc/ansible/hosts server_centos_kuber.yml -u makarov -K
BECOME password:

PLAY [Install K3s Master] *****

TASK [Gathering Facts] *****
[WARNING]: Host 'localhost' is using the discovered Python interpreter at '/usr/bin/python3.12', but future installation
of another Python interpreter could cause a different interpreter to be discovered. See https://docs.ansible.com/ansibl
e-core/2.19/reference_appendices/interpreter_discovery.html for more information.
ok: [localhost]

TASK [Install K3s Server (Master)] *****
changed: [localhost]

TASK [Ensure .kube directory exists] *****
ok: [localhost]

TASK [Copy k3s config to project directory] *****
changed: [localhost]

TASK [Wait for Node Token to be generated] *****
ok: [localhost]

TASK [Clone the repository] *****
ok: [localhost]

TASK [Creating config for db] *****
changed: [localhost]

TASK [Applying deployment of webapp] *****
changed: [localhost]

PLAY RECAP *****
localhost      : ok=15   changed=7   unreachable=0   failed=0   skipped=0   rescued=0   ignored=0
server_scene3  : ok=3    changed=1   unreachable=0   failed=0   skipped=1   rescued=0   ignored=0
```

Рисунок 3.16 – Результат роботи ansible

3.3.3. Перевірка роботи кластеру

Після завершення автоматичного розгортання критично важливим етапом є підтвердження того, що кластер функціонує як єдиний організм, а політики розподілу ресурсів виконуються коректно.

Перевірка топології та статусу вузлів Першим кроком за допомогою утиліти kubectl було перевірено стан реєстрації вузлів. Команда kubectl get nodes -o wide підтвердила наявність двох активних учасників кластера:

```
alex@DESKTOP-08S00BT:~$ kubectl get nodes -o wide
NAME             STATUS    ROLES          AGE   VERSION           INTERNAL-IP    EXTERNAL-IP    OS-IMAGE
centos-server    Ready    <none>         6m24s v1.33.6+k3s1     192.168.56.103 <none>         CentOS Stream 10 (Coughlan)
desktop-08s0obt  Ready    control-plane,master 7m33s v1.33.6+k3s1     172.31.24.42  <none>         Ubuntu 24.04.3 LTS
```

Рисунок 3.17 – Вивід інформацію про ноди

- desktop-08s0obt (Control Plane): Хост-система з версією ядра Linux (WSL), яка має статус Ready та роль control-plane,master.
- centos-server (Worker Node): Віртуальна машина CentOS з IP-адресою 192.168.56.103, яка також має статус Ready.

Це свідчить про успішну роботу CNI-плагіна Flannel через налаштований інтерфейс enp0s8.

Виконання команди `kubectl get pods,services -o wide` показало наступний розподіл:

```
alex@DESKTOP-08S00BT:~$ kubectl get pods,services -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED	NODE	READINESS	GATES
pod/app-7549b7c56b-swwff	1/1	Running	0	18m	10.42.1.5	centos-server	<none>		<none>	
pod/db-5459b84b8f-c2ljt	1/1	Running	0	18m	10.42.1.4	centos-server	<none>		<none>	

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
service/app-service	LoadBalancer	10.43.177.55	172.31.24.42,192.168.56.103	8080:30837/TCP	18m	app=app
service/db	ClusterIP	10.43.45.176	<none>	27017/TCP	18m	app=db

Рисунок 3.18 – Вивід інформацію про поди і сервіси

Поди `app` та `db` запущені тільки на вузлі `centos-server` (192.168.56.103) – це значить що задіяні налаштування, яка обмежує створення подів на ноди з роллю `control-plane`, працюють.

Тепер перевіримо доступ до веб-додатку.

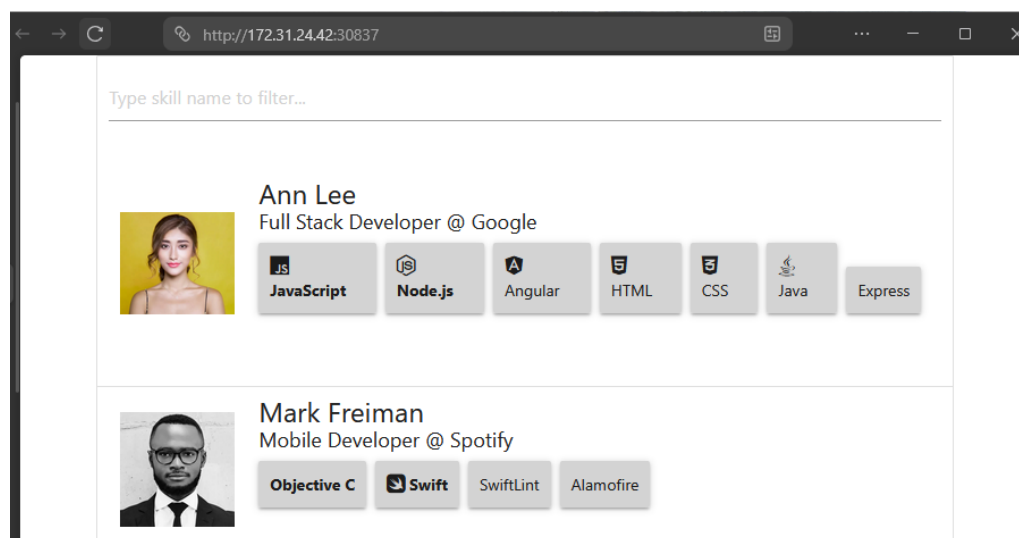


Рисунок 3.19 – Перевірка доступу до пода

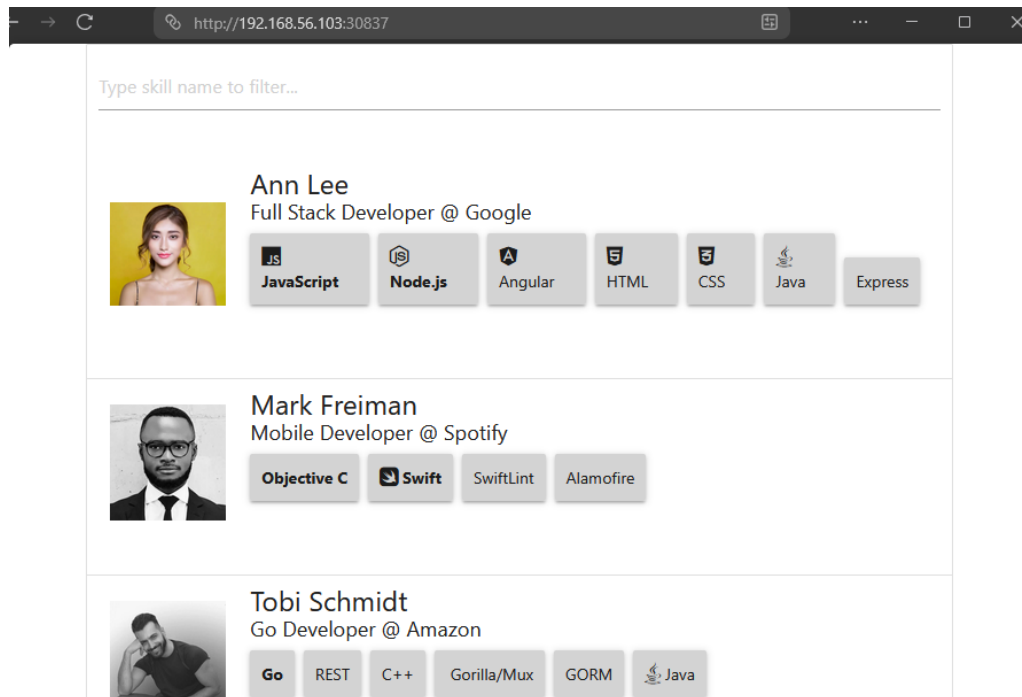


Рисунок 3.20 – Перевірка доступу до пода через іншу адресу

Перевірка доступності веб-додатку здійснювалася через браузер. Завдяки вбудованому в K3s Ingress-контролеру Traefik, трафік автоматично маршрутизується до відповідного сервісу. При зверненні за адресою кластера додаток "Company Employees" успішно завантажується, відображаючи дані з бази MongoDB, що свідчить про коректну роботу внутрішньої мережі сервісів (ClusterIP) та DNS-різолвінгу всередині кластера.

Для демонстрації переваг Kubernetes над звичайним Docker було проведено імітацію збою. Примусове видалення поду бекенду командою `kubectl delete pod`.

```
alex@DESKTOP-08S00BT:~$ kubectl delete pod app-7549b7c56b-swwff
pod "app-7549b7c56b-swwff" deleted from default namespace
alex@DESKTOP-08S00BT:~$ kubectl get pods,services -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP
pod/app-7549b7c56b-fsv5p	1/1	Running	0	4s	10.42.1.6
pod/db-5459b84b8f-c2ljt	1/1	Running	0	25m	10.42.1.4

Рисунок 3.21 – Перевірка доступу до пода через іншу адресу

Це призвело до миттєвої реакції контролера ReplicaSet, який автоматично створив новий екземпляр поду менш ніж за декілька секунди. Це гарантує високу доступність сервісу.

3.4. Порівняльний аналіз реалізованих підходів до автоматизації

У ході виконання третього розділу було практично досліджено три еволюційні стадії керування серверною інфраструктурою: класичне розгортання на "чистому залізі" (Bare Metal), контейнеризацію (Docker) та оркестрацію (Kubernetes/K3s). Для об'єктивної оцінки ефективності кожного підходу проведено порівняння за чотирма ключовими технічними критеріями.

3.4.1. Ефективність використання ресурсів та ізоляція

У першому сценарії всі сервіси (MongoDB, Go Backend, Node.js Frontend) запускалися безпосередньо в просторі користувача операційної системи. Споживання ресурсів є найнижчим, оскільки відсутні проміжні шари абстракції. Процесорний час витрачається виключно на виконання бізнес-логіки додатку. Ізоляція практично відсутня. Всі сервіси ділять спільну файлову систему та бібліотеки. Як показав досвід налаштування NVM, це створює ризик конфліктів версій ("Dependency Hell"), коли оновлення системної бібліотеки може порушити роботу додатку.

У другому сценарії додатки було упаковано в ізольовані контейнери, що запускалися на базі Docker Engine. Споживання ресурсів помірне. Додаткові витрати ресурсів (CPU/RAM) йдуть на роботу демона dockerd та середовища виконання containerd. Однак, завдяки використанню оптимізованих образів (наприклад, Alpine Linux), загальний обсяг дискового простору, що займає додаток, зменшився. Ізоляція, при цьому, висока. Кожен контейнер має власну файлову систему і мережевий стек. Тестування показало, що на хост-системі CentOS відсутні сліди Node.js або Go, що гарантує чистоту ОС.

У третьому сценарії використано кластерну архітектуру. Споживання ресурсів на цей раз найвище. Навіть у стані спокою на вузлах працюють системні поди: coredns (DNS-різолюїнг), metrics-server (моніторинг), traefik (Ingress-контролер) та CNI-плагін flannel. Це створює постійне навантаження на рівні 10-15% CPU та ~500 МБ RAM на кожному вузлі, що є "платою" за можливості автоматизації. Час розгортання: ~5-8 хвилин (залежить від потужності CPU

сервера). При оновлення додатку, потрібно зупиняти сервіс, перекомпілювати та повторного запускати, що призводить до простою (Downtime).

3.4.2. Швидкість розгортання та оновлення

У першому сценарії, процес розгортання виявився найтривалішим. Ansible-плейбук `server_ubuntu.yml` виконував компіляцію коду з вихідних текстів безпосередньо на сервері (`go build, npm install`).

У другому сценарії, завдяки шаруватій структурі образів (Layered Filesystem), Docker завантажує лише ті шари, які змінилися. Час розгортання: ~1-2 хвилини (переважно на завантаження образів). Оновлення відбувається шляхом заміни контейнера. Це швидше за компіляцію, але все ще може спричинити короточасний розрив з'єднань під час перезапуску `docker compose up -d`.

У третьому сценарії, первинна ініціалізація кластера є найдовшою через налаштування віртуальної машини через Packer та приєднання вузлів. Проте оновлення додатку відбувається миттєво для користувача.

3.4.3. Складність конфігурації

Ansible для Bare Metal вимагає написання найбільш складних та об'ємних плейбуків. Адміністратор повинен вручну описувати кожну дрібницю: створення systemd-юнітів для PM2, керування користувачами, правами доступу до папок та змінними оточення. Це збільшує ризик людської помилки.

Ansible для Docker значно спрощує код інфраструктури. Роль Ansible зводиться до встановлення Docker Engine. Вся логіка запуску додатку інкапсульована у файл `docker-compose.yml`, який є стандартом де-факто для розробників. Це чітко розділяє відповідальність: адмін налаштовує сервер, розробник — контейнер.

Ansible для K3s має найвищий поріг входження. Необхідно керувати не лише серверами, а й об'єктами кластера. Плейбук `server_centos_kuber.yml` демонструє складну логіку передачі токенів між хостами, налаштування Taints та роботу з `kubectl`.

3.4.4. Підсумкова порівняльна таблиця

Узагальнені технічні характеристики реалізованих систем наведено у таблиці 3.1.

Таблиця 3.1

Порівняння підходів до автоматизації серверної інфраструктури

Характеристика	Bare Metal (Сценарій 1)	Docker (Сценарій 2)	Kubernetes/K3s (Сценарій 3)
Базова ОС	Ubuntu Server 24.04	CentOS Stream 10	Гібридна (Ubuntu + CentOS)
Інструмент запуску	Systemd + PM2	Docker Compose	K3s (Containerd)
Метод доставки	git clone	Docker Image	Docker Image + YAML маніфести
Масштабування	Вертикальне (додавання CPU/RAM)	Ручне горизонтальне (копії контейнерів)	Автоматичне (ReplicaSet / HPA)
Самовідновлення	Обмежене (PM2 restart)	Середнє (Docker restart policy)	Повне (Rescheduling на інший вузол)
Складність	Середня	Низька	Висока
Мережева модель	Host Networking	Bridge Network	Overla Network (VXLAN)

Висновок до третього розділу:

У третьому розділі кваліфікаційної роботи було успішно розроблено та імплементовано комплексну систему автоматизованого управління серверами. Використання зв'язки Packer + Ansible дозволило реалізувати парадигму Immutable Infrastructure (у сценаріях 2 та 3), де базові образи віртуальних машин створюються автоматично і не змінюються в процесі роботи, що гарантує передбачуваність системи.

Результати досліджень дозволяють сформулювати рекомендації щодо застосування кожного з підходів:

1. Bare Metal доцільно використовувати для легасі-додатків або систем з екстремальними вимогами до продуктивності, де оверхед віртуалізації є неприпустимим.

2. Docker є ідеальним вибором для середовищ розробки (Dev/Test) та малих проєктів завдяки простоті конфігурації та швидкості розгортання.

3. Kubernetes (K3s) є безальтернативним рішенням для критично важливих корпоративних систем (Production), оскільки лише він забезпечує автоматичне відновлення після збоїв, нульовий час простою при оновленнях та гнучке масштабування, що було підтверджено під час тестування відмовостійкості кластера.

ВИСНОВКИ

У магістерській роботі вирішено актуальне науково-практичне завдання, пов'язане з автоматизацією процесів розгортання, налаштування та управління корпоративною серверною інфраструктурою. Основною метою дослідження було створення ефективної, масштабованої та надійної системи, що мінімізує ручну працю системних адміністраторів та знижує ризик людських помилок при обслуговуванні ІТ-систем підприємства. У процесі виконання роботи було проаналізовано сучасні підходи до управління конфігураціями, практично реалізовано концепцію «Інфраструктура як код» (IaC) та побудовано кластерне рішення для оркестрації контейнеризованих додатків.

На етапі теоретичного дослідження було детально розглянуто архітектурні особливості операційних систем сімейства Linux, зокрема дистрибутивів CentOS та Ubuntu, та обґрунтовано їх вибір як базової платформи для корпоративних серверів. Проведено порівняльний аналіз сучасних інструментів автоматизації та віртуалізації, за результатами якого було сформовано оптимальний технологічний стек. Для автоматизованого створення образів віртуальних машин було обрано інструмент Packer у поєднанні з технологією Kickstart, що дозволило стандартизувати процес підготовки серверів. В якості системи управління конфігураціями використано Ansible завдяки його безагентній архітектурі та гнучкості написання сценаріїв. Для оркестрації контейнерів перевагу було надано легкому дистрибутиву Kubernetes — K3s, який забезпечує високу продуктивність при помірному споживанні ресурсів.

У ході практичної реалізації проєкту було розроблено та впроваджено комплексний конвеєр автоматизації. Створено сценарії Packer, які дозволяють автоматично генерувати готові до роботи образи операційних систем із попередньо встановленим програмним забезпеченням та необхідними налаштуваннями безпеки. Розроблено структуру Ansible-плейбуків, що забезпечують ідемпотентне налаштування серверів, розгортання кластера K3s із розділенням ролей на Master та Worker-ноди, а також конфігурацію мережевої взаємодії між ними. Було

реалізовано механізм передачі секретних даних та змінних між етапами розгортання, що дозволило створити повністю автономний процес інсталяції без необхідності ручного втручання.

Фінальним етапом роботи стала перевірка працездатності створеної інфраструктури шляхом розгортання тестового веб-додатку. Результати тестування підтвердили, що запропоноване рішення дозволяє суттєво скоротити час на введення нових серверів в експлуатацію, забезпечує уніфікацію налаштувань у всьому серверному парку та гарантує швидке відновлення сервісів у разі збоїв. Розроблена система автоматизації є гнучкою та може бути легко адаптована для потреб будь-якого підприємства, що прагне модернізувати свої ІТ-процеси та впровадити практики DevOps.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Server Operating System: Server OS Types & How to Choose [Електронний ресурс] – Режим доступу до ресурсу: <https://phoenixnap.com/kb/server-operating-system> (дата звернення: 10.01.2025).
2. 15 different types of servers in computing. Zenarmor - Agile Service Edge Security. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.zenarmor.com/docs/network-basics/types-of-servers> (дата звернення: 10.01.2025).
3. Best Linux server distro [Електронний ресурс] – Режим доступу до ресурсу: <https://www.techradar.com/best/best-linux-server-distro> (дата звернення: 10.01.2025).
4. Comparison of Linux Distributions [Електронний ресурс] – Режим доступу до ресурсу: https://eylenburg.github.io/linux_comparison.htm (дата звернення: 10.01.2025).
5. CentOS vs CoreOS | What are the differences? [Електронний ресурс] – Режим доступу до ресурсу: <https://stackshare.io/stackups/centos-vs-coreos> (дата звернення: 10.01.2025).
6. Debian Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.debian.org/doc/> (дата звернення: 15.01.2025).
7. Ubuntu Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://ubuntu.com/server/docs> (дата звернення: 15.01.2025).
8. CentOS Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.centos.org/en-US/docs/> (дата звернення: 15.01.2025).
9. Fedora Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.fedoraproject.org/en-US/fedora-server/> (дата звернення: 15.01.2025).
10. CoreOS Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.redhat.com/en/documentation/openshift_container_platform/4.1/html/architecture/architecture-rhcos#rhcos-about_architecture-rhcos (дата звернення: 17.01.2025).

11. The most popular linux distributions statistics [Электронный ресурс] – Режим доступа до ресурсу: <https://jobera.com/linux-distributions-statistics/> (дата звернення: 10.01.2025).
12. What is Infrastructure as Code (IaC)? [Электронный ресурс] – Режим доступа до ресурсу: <https://www.geeksforgeeks.org/what-is-infrastructure-as-code-iac/> (дата звернення: 10.01.2025).
13. 16 most useful infrastructure as code (iac) tools [Электронный ресурс] – Режим доступа до ресурсу: <https://spacelift.io/blog/infrastructure-as-code-tools> (дата звернення: 10.01.2025).
14. HashiCorp Packer Documentation. Automate Image Builds. [Электронный ресурс] – Режим доступа до ресурсу: <https://developer.hashicorp.com/packer/docs> (дата звернення: 12.11.2025).
15. Ansible Documentation [Электронный ресурс]. – Режим доступа: <https://docs.ansible.com/> (дата звернення: 19.11.2025).
16. K3s Documentation. Lightweight Kubernetes. [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.k3s.io/> (дата звернення: 10.11.2025).

ДОДАТКИ

Конфігураційні packer файли

ubuntu.pkr.hcl:

```

packer {
  required_plugins {
    virtualbox = {
      version = "~> 1.0.0"
      source  = "github.com/hashicorp/virtualbox"
    }
  }
}

source "virtualbox-iso" "ubuntu" {
  vm_name = "Ubuntu_Server"
  guest_os_type = "Ubuntu_64"
  iso_url   = "E:\\Downloads\\ubuntu-24.04.3-live-server-amd64.iso"
  disk_size = 40000
  memory    = 4096
  cpus      = 4
  output_directory = "E:\\VirtualBox VMs\\Ubuntu_Server"
  http_directory = "http"
  ssh_username = "makarov"
  ssh_password = "makarov"
  ssh_timeout = "30m"
  iso_interface = "sata"
  hard_drive_interface = "sata"
  boot_wait = "5s"
  skip_export = true
  keep_registered = true
  iso_checksum = "c3514bf0056180d09376462a7a1b4f213c1d6e8ea67fae5c25099c6fd3d8274b"
  shutdown_command = "echo 'makarov' | sudo -S shutdown -P now"
  http_bind_address = "192.168.56.1"
  vboxmanage = [
    ["modifyvm", "{{.Name}}", "--nic1", "nat"],
    ["modifyvm", "{{.Name}}", "--nic2", "hostonly"],
    ["modifyvm", "{{.Name}}", "--hostonlyadapter2", "VirtualBox Host-Only Ethernet Adapter"]
  ]
  boot_command = [
    "c", "<wait>",
    "linux /casper/vmlinuz <wait>",
    "--- ",
    "autoinstall <wait>",
    "ds=nocloud-net\\;<wait>s=http://192.168.56.1:{{ .HTTPEndPoint }}/ ",
    "<enter><wait>",
    "initrd <wait>/casper/initrd<enter><wait>",
    "boot<enter>"
  ]
}

```

```

build {
  sources = ["source.virtualbox-iso.ubuntu"]
  post-processor "shell-local" {
    inline = [
      "VBoxManage startvm Ubuntu_Server --type gui"
    ]
  }
}

```

centos_docker.pkr.hcl:

```

packer {
  required_plugins {
    virtualbox = {
      version = "~> 1.0.0"
      source  = "github.com/hashicorp/virtualbox"
    }
  }
}

source "virtualbox-iso" "centos" {
  vm_name      = "CentOS_Server"
  guest_os_type = "RedHat_64"
  iso_url      = "E:\\Downloads\\CentOS-Stream-10-latest-x86_64-dvd1.iso"
  iso_checksum =
"sha256:DA759C3FDEC79EDE1B9EBF921C72495B688ECE424BCC890F5C7CE01B0D6909C5"
  http_port_min = 9100
  http_port_max = 10000
  disk_size     = 40000
  memory        = 4096
  cpus          = 4

  output_directory = "E:\\VirtualBox VMs\\CentOS_Server"

  http_directory = "http"
  http_bind_address = "192.168.56.1"

  ssh_username = "makarov"
  ssh_password = "makarov"
  ssh_timeout  = "30m"
  firmware    = "efi"
  iso_interface = "ide"
  hard_drive_interface = "ide"

  boot_wait = "5s"
  skip_export = true
  keep_registered = true
  shutdown_command = "echo 'makarov' | sudo -S /sbin/shutdown -hP now"

  vboxmanage = [
    ["modifyvm", "{{.Name}}", "--nic1", "nat"],

```

```

["modifyvm", "{{.Name}}", "--nic2", "hostonly"],
["modifyvm", "{{.Name}}", "--hostonlyadapter2", "VirtualBox Host-Only Ethernet Adapter"]
]

boot_command = [
    "c<wait>",
    "linux /images/pxeboot/vmlinuz<wait> inst.text inst.ks=http://192.168.56.1:{{ .HTTPPort }}/ks.cfg<enter><wait>",
    "initrdefi /images/pxeboot/initrd.img<enter><wait>",
    "boot<enter>",
]
}

build {
    sources = ["source.virtualbox-iso.centos"]

    post-processor "shell-local" {
        inline = [
            "VBoxManage startvm CentOS_Server --type gui"
        ]
    }
}

```

centos_kube.pkr.hcl:

```

packer {
    required_plugins {
        virtualbox = {
            version = "~> 1.0.0"
            source  = "github.com/hashicorp/virtualbox"
        }
    }
}

source "virtualbox-iso" "centos" {
    vm_name      = "CentOS_Server"
    guest_os_type = "RedHat_64"
    iso_url      = "E:\\Downloads\\CentOS-Stream-10-latest-x86_64-dvd1.iso"
    iso_checksum =
"sha256:DA759C3FDEC79EDE1B9EBF921C72495B688ECE424BCC890F5C7CE01B0D6909C5"
    http_port_min = 9100
    http_port_max = 10000
    disk_size     = 40000
    memory        = 4096
    cpus          = 4

    output_directory = "E:\\VirtualBox VMs\\CentOS_Server"

    http_directory = "http"
    http_bind_address = "192.168.56.1"

    ssh_username = "makarov"

```

```

ssh_password = "makarov"
ssh_timeout = "30m"
firmware = "efi"
iso_interface = "ide"
hard_drive_interface = "ide"

boot_wait = "5s"
skip_export = true
keep_registered = true
shutdown_command = "echo 'makarov' | sudo -S /sbin/shutdown -hP now"

vboxmanage = [
  ["modifyvm", "${Name}", "--nic1", "nat"],
  ["modifyvm", "${Name}", "--nic2", "hostonly"],
  ["modifyvm", "${Name}", "--hostonlyadapter2", "VirtualBox Host-Only Ethernet Adapter"]
]

boot_command = [
  "c<wait>",
  "linux /images/pxeboot/vmlinuz<wait> inst.text inst.ks=http://192.168.56.1:${HTTPPort}
}/ks_kube.cfg<enter><wait>",
  "initrdefi /images/pxeboot/initrd.img<enter><wait>",
  "boot<enter>",
]
}

build {
  sources = ["source.virtualbox-iso.centos"]

  post-processor "shell-local" {
    inline = [
      "VBoxManage startvm CentOS_Server --type gui"
    ]
  }
}

```

Kickstart конфігурації

user-data:

```
#cloud-config
autoinstall:
  version: 1
  locale: en_US.UTF-8
  keyboard:
    layout: us
  refresh-installer:
    update: no
  network:
    network:
      version: 2
      ethernets:
        enp0s3:
          dhcp4: true
        enp0s8:
          dhcp4: false
      addresses: [192.168.56.101/24]
  identity:
    hostname: ubuntu-server
    username: makarov
    password:
"$6$LI9XUSGS6GlmT9uY$KflYlqwq4WNKv/IWy6KUDej4wZKrdkG.l/0Mc.n2KminDwrya2mKH
bnckvhaOwQXn8p1axHBorlfnTtruf0Wf/"
  ssh:
    install-server: yes
    allow-pw: yes
    authorized-keys:
      - "ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIHq+Cch8TPA6Fy8cGWHPFP1QpymQhZs5hah10/t+eziR
alex@DESKTOP-O8S0OBT"
  storage:
    layout:
      name: direct
  user-data:
    disable_root: false
  late-commands:
    - "echo 'Packer build complete' > /target/etc/packer_build_info"
```

ks.cfg:

```
lang en_US.UTF-8
keyboard us
timezone UTC
network --device=enp0s3 --bootproto=dhcp --onboot=yes --activate
network --device=enp0s8 --bootproto=static --ip=192.168.56.102 --netmask=255.255.255.0 --
onboot=yes --noipv6 --activate
network --hostname=centos-server
```

```

rootpw --plaintext makarov
user --name=makarov --password=makarov --plaintext --groups=wheel
firewall --enabled --ssh
selinux --enforcing
sshkey --username=makarov "ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIHq+Cch8TPA6Fy8cGWHPFP1QpymQhZs5hah10/t+eziR
alex@DESKTOP-O8S0OBT"
cdrom
text
zerombr
clearpart --all --initlabel
autopart
bootloader --location=mbr --append="rhgb quiet crashkernel=auto"
reboot
%packages
@^minimal-environment
kexec-tools
%end
%post
echo 'Packer build complete' > /etc/packer_build_info
%end

```

ks_kube.cfg:

```

lang en_US.UTF-8
keyboard us
timezone UTC
network --device=enp0s3 --bootproto=dhcp --onboot=yes --activate
network --device=enp0s8 --bootproto=static --onboot=yes --ip=192.168.56.103 --
netmask=255.255.255.0 --onboot=yes --noipv6 --activate
network --hostname=centos-server
rootpw --plaintext makarov
user --name=makarov --password=makarov --plaintext --groups=wheel
firewall --disabled
selinux --enforcing
sshkey --username=makarov "ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIHq+Cch8TPA6Fy8cGWHPFP1QpymQhZs5hah10/t+eziR
alex@DESKTOP-O8S0OBT"
cdrom
text
zerombr
clearpart --all --initlabel
autopart
bootloader --location=mbr --append="rhgb quiet crashkernel=auto"
reboot
%packages
@^minimal-environment
kexec-tools
%end
%post
echo 'Packer build complete' > /etc/packer_build_info
%end

```

Ansible playbook конфігурації

server_ubuntu_bare.yml

- name: Configure Ubuntu Server for Company Employees App

hosts: server_scene1

become: yes

vars:

app_user: "makarov"

project_root: "/home/{{ app_user }}/company_employees"

mongodb_gpg_key: "/usr/share/keyrings/mongodb-server-8.0.gpg"

tasks:

- name: Install basic system packages

apt:

name:

- git

- curl

- build-essential

- golang-go

- nodejs

- npm

- gnupg

state: present

update_cache: yes

- name: Import MongoDB GPG key

apt_key:

url: <https://www.mongodb.org/static/pgp/server-8.0.asc>

state: present

- name: Add MongoDB Repository

apt_repository:

repo: "deb [arch=amd64,arm64] <https://repo.mongodb.org/apt/ubuntu/jammy/mongodb-org/8.0>

multiverse"

filename: mongodb-org-8.0

state: present

- name: Install MongoDB

apt:

name: mongodb-org

state: present

update_cache: yes

- name: Ensure MongoDB is running and enabled

service:

name: mongod

state: started

enabled: yes

```

- name: Clone the repository
  git:
    repo: 'https://github.com/AlexKuplev/company_employees.git'
    dest: "{{ project_root }}"
    version: HEAD
  become: no

- name: Initialize MongoDB Data
  shell: |
    mongosh "mongodb://localhost:27017/programmers" --file {{ project_root
  }}/server/db/mongo.init
  ignore_errors: yes

- name: Install PM2 globally
  npm:
    name: pm2
    global: yes
    state: present

- name: Build Go Backend
  shell: |
    go build -o app server.go
  args:
    chdir: "{{ project_root }}/server"
  become: no

- name: Start Backend with PM2
  command: pm2 start ./app --name backend
  args:
    chdir: "{{ project_root }}/server"
  become: no
  ignore_errors: yes

- name: Install NVM and Frontend Dependencies
  become: no
  shell: |
    # Download and install NVM
    curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash

    # Source NVM immediately for this session
    source ~/.bashrc

    # Install Node version and deps
    nvm install 12.14
    nvm alias default 12.14
    nvm use 12.14

    # Install frontend packages
    npm install

```

```

# Start frontend with PM2
pm2 start npm --name frontend -- start
args:
chdir: "{{ project_root }}/webapp"
executable: /bin/bash

- name: Generate PM2 Startup Script
  command: "pm2 startup systemd -u {{ app_user }} --hp /home/{{ app_user }}"
  environment:
    PATH: "/usr/local/bin:{{ ansible_env.PATH }}"
  ignore_errors: yes

- name: Save PM2 list
  command: pm2 save
  become: no

```

server_centos_docker.yml

```

---
- name: Configure CentOS Server for Docker App
  hosts: server_scene2
  become: yes
  vars:
    # Change 'makarov' to your actual user if different
    app_user: "makarov"
    project_root: "/home/{{ app_user }}/company_employees"
    docker_compose_version: "v2.38.2"

  tasks:
    - name: Remove old Docker packages
      dnf:
        name:
          - docker
          - docker-client
          - docker-client-latest
          - docker-common
          - docker-latest
          - docker-latest-logrotate
          - docker-logrotate
          - docker-engine
        state: absent

    - name: Install DNF plugins and Git
      dnf:
        name:
          - dnf-plugins-core
          - git
        state: present

    - name: Add Docker CE Repository
      command: dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-
ce.repo
      args:

```

creates: /etc/yum.repos.d/docker-ce.repo

- name: Install Docker Engine and Plugins

dnf:

name:

- docker-ce
- docker-ce-cli
- containerd.io
- docker-buildx-plugin
- docker-compose-plugin

state: present

- name: Ensure Docker service is running

service:

name: docker

state: started

enabled: yes

- name: Install Docker Compose Standalone

get_url:

url: "https://github.com/docker/compose/releases/download/{{ docker_compose_version }}/docker-compose-linux-x86_64"

dest: /usr/local/bin/docker-compose

mode: '0755'

- name: Clone the repository

git:

repo: 'https://github.com/AlexKuplev/company_employees.git'

dest: "{{ project_root }}"

version: HEAD

Important: Run as the user so they own the files

become: yes

become_user: "{{ app_user }}"

- name: Run Docker Compose Up

shell: docker compose up -d

args:

chdir: "{{ project_root }}"

server_centos_kuber.yml

- name: Install K3s Master

hosts: master

become: yes

vars:

app_user: "alex"

home_dir: "/home/{{ app_user }}"

docker_compose_version: "v2.38.2"

tasks:

- name: Install K3s Server (Master)

shell: curl -sfL https://get.k3s.io | sh -

args:

```

    creates: /usr/local/bin/k3s

- name: Ensure .kube directory exists
  file:
    path: "{{ home_dir }}/.kube"
    state: directory
    owner: "{{ app_user }}"
    group: "{{ app_user }}"
    mode: '0755'
  become: yes

- name: Copy k3s config to project directory
  copy:
    src: /etc/rancher/k3s/k3s.yaml
    dest: "{{ home_dir }}/.kube/config"
    remote_src: yes
    owner: "{{ app_user }}"
    group: "{{ app_user }}"
    mode: '0600'
  become: yes

- name: Wait for Node Token to be generated
  wait_for:
    path: /var/lib/rancher/k3s/server/node-token

- name: Read the Node Token
  slurp:
    src: /var/lib/rancher/k3s/server/node-token
  register: node_token

- name: Save the Token
  set_fact:
    k3s_token: "{{ node_token['content'] | b64decode | trim }}"

- name: Get Master IP (WSL Interface)
  shell: "ip -4 -br addr show eth0 | awk '{print $3}' | cut -d/' -f1"
  register: master_ip_output

- name: Save Master IP as a variable
  set_fact:
    k3s_master_ip: "{{ master_ip_output.stdout }}"

- name: Expose variables to other plays
  add_host:
    name: "K3S_VARIABLES"
    token: "{{ k3s_token }}"
    master_ip: "{{ k3s_master_ip }}"

- name: Install K3s Worker
  hosts: server_scene2
  become: yes

```

```

vars:
  token: "{{ hostvars['K3S_VARIABLES']['token'] }}"
  master_ip: "{{ hostvars['K3S_VARIABLES']['master_ip'] }}"

tasks:
  - name: Check connectivity to Master API
    wait_for:
      host: "{{ master_ip }}"
      port: 6443
      timeout: 10
    ignore_errors: yes
    register: connection_check

  - name: Fail if Master is unreachable
    fail:
      msg: "Cannot reach Master at {{ master_ip }}:6443. Check firewalls/routing!"
    when: connection_check.failed

  - name: Install K3s Agent and Join Cluster
    shell: "curl -sL https://get.k3s.io | K3S_URL=https://{{ master_ip }}:6443 K3S_TOKEN={{
token }} sh -s - --flannel-iface enp0s8"
    args:
      creates: /usr/local/bin/k3s-agent

  - name: Clone and apply manifest
    hosts: master
    vars:
      app_user: "alex"
      project_root: "/home/{{ app_user }}/company_employees"
      docker_compose_version: "v2.38.2"
    tasks:
      - name: Taint Master Node (NoSchedule)
        shell: "kubectl taint nodes --selector='node-role.kubernetes.io/master' node-
role.kubernetes.io/master=true:NoSchedule --overwrite"

      - name: Clone the repository
        git:
          repo: 'https://github.com/AlexKuplev/company_employees.git'
          dest: "{{ project_root }}"
          version: HEAD
        become: yes
        become_user: "{{ app_user }}"

      - name: Creating config for db
        shell: "kubectl create configmap mongo-init --from-file ./server/db/mongo.init"
        args:
          chdir: "{{ project_root }}"

      - name: Applying deployment of webapp
        shell: "kubectl apply -f ./webapp.yaml"
        args:
          chdir: "{{ project_root }}"

```