МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Центр заочної, дистанційної та вечірньої форм навчання

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

22 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 – Комп'ютерних наук,

освітньо-професійної програми «Інформаційно-комунікаційні технології»

на тему: «Інформаційна технологія налаштування, розгортання та контролю віртуальних машин за допомогою Vagrant»

здобувача групи ІК.мз-11с Власенка Олександра Володимировича

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

_____ Олександр ВЛАСЕНКО

(підпис)

Керівник,

доцент кафедри комп'ютерних наук,

к.ф.-м.н., доцент

Сергій ШАПОВАЛОВ _____

(підпис)

Суми – 2023

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
зі спеціальності 122 – Комп'ютерних наук, освітньо-професійної програми
«Інформаційно-комунікаційні технології»
здобувача групи ІК.мз-11с Власенка Олександра Володимировича

1. Тема роботи: «Інформаційна технологія налаштування, розгортання та контролю віртуальних машин за допомогою Vagrant»
затверджую наказом по СумДУ від «01» грудня 2023 р. № 1389-VI
2. Термін здачі здобувачем кваліфікаційної роботи до 22 грудня 2023 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми, постановка й формування завдань для роботи за обраною тематикою. 2) Огляд технологій, що використовуються для розробки налаштування, розгортання та контролю віртуальних машин. 3) Практична реалізація. 4) Аналіз виконаної роботи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» листопада 2023 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми, постановка й формування завдань для роботи за обраною тематикою*		
2	*Огляд технологій, що використовуються для розробки налаштування, розгортання та контролю віртуальних машин*		
3	*Практична реалізація*		
4	*Аналіз виконаної роботи*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 43 стор., 16 рис., 1 додаток, 20 джерел.

Обґрунтування актуальності теми роботи – актуальність обраної тематики для дослідження обумовлена необхідністю автоматизації процесів адміністрування віртуальних машин системними адміністраторами ОС Linux.

Об'єкт дослідження – аналіз технічних рішень проєктування інформаційних технологій за допомогою технології Vagrant.

Мета роботи – провести аналітичний огляд за обраною тематикою дослідження, проаналізувати та обґрунтувати вибір технологій для практичної реалізації.

Методи дослідження – системний аналіз, компонентне проєктування, Linux-адміністрування.

Результати – на основі літературного огляду обрано оптимальні інструменти для розробки, створено графічний інтерфейс для розгортання кластеру з використанням Vagrant.

JAVA FX, VAGRANT, ГРАФІЧНИЙ ІНТЕРФЕЙС,
ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, ОС LINUX,
СИСТЕМНЕ АДМІНІСТРУВАННЯ

ЗМІСТ

ВСТУП

Актуальність. Наразі існує велика кількість систем та пристроїв, що потребують адміністрування для їх належного функціонування. Проте часто виникають проблеми через розрізненість інструментів та методів управління, особливо при масштабуванні інфраструктури. Поширення хмарних технологій та віртуалізації призвело до зростання кількості систем, що потребують управління. Це зумовлює актуальність створення спеціалізованих інструментів адміністрування з єдиною точкою доступу.

Світовий ринок операційних систем є досить стабільним і давно не зазнавав радикальних змін. Основні операційні системи, що використовуються, вже давно відомі користувачам та зарекомендували себе у певних сферах діяльності. Операційна система сімейства Linux давно набула популярності у сферах IT, інженерії та навіть у якості домашніх систем. Тому дослідження є актуальним.

Об’єкт дослідження: процеси адміністрування комп’ютерних систем.

Предмет дослідження: програмні рішення та методи для поліпшення процесів адміністрування.

Гіпотеза. Відсутність єдиної інтегрованої платформи управління ускладнює адміністрування розподілених систем та призводить до зниження ефективності. Розробка такої платформи дозволить спростити процеси адміністрування та підвищити продуктивність.

Структура. Кваліфікаційна робота складається з вступу, аналітичного огляду, вибору методів рішення, програмної реалізацій, висновку, списку використаних джерел та додатку.

1. ІНФОРМАЦІНИЙ ОГЛЯД СИСТЕМНОГО АДМІНІСТРУВАННЯ

1.1 Основні відомості про системне адміністрування

Системне адміністрування – це широкий комплекс заходів, спрямованих на підтримку безперебійної та стабільної роботи інформаційних систем організації. Системний адміністратор – це спеціаліст, який виконує різноманітні можливі налаштування у багатьох сферах [1].

Основні завдання системного адміністратора включають налаштування та обслуговування мережевої інфраструктури, серверного обладнання, робочих станцій, а також системного та прикладного програмного забезпечення.

Адміністратор повинен забезпечити надійний захист мережі від можливих загроз, а також її працездатність і продуктивність. Він відповідає за встановлення, моніторинг, налаштування та оновлення офісних додатків, серверних операційних систем (ОС) тощо. Як показує досвід, ще одним важливим обов’язком є своєчасне резервне копіювання даних і відновлення у разі їх пошкодження чи втрати. Для цього системний адміністратор може застосовувати різноманітне програмне забезпечення та методи [2-3].

Одним з важливих напрямків роботи системного адміністратора є віртуалізація – створення та обслуговування віртуальних машин. Це дозволяє розгортати на одному фізичному сервері декілька операційних систем та/або середовищ. Завдання адміністратора у такому випадку – налаштувати правильний розподіл обчислювальних ресурсів (процесор, пам’ять, дисковий простір) між віртуальними ОС, оптимізувати їх роботу, а також усувати можливі конфлікти та несправності.

Ще один напрямок – адміністрування баз даних, які використовуються для зберігання великих обсягів різноманітної інформації – від персональних даних до внутрішніх даних систем компанії. Адміністратор БД відповідає за встановлення та налаштування систем управління базами (Oracle, MS SQL

Server тощо), подальший моніторинг, оптимізацію роботи, вирішення проблем, своєчасне резервне копіювання даних та їх відновлення у разі втрати.

Управління системами уніфікованих комунікацій, що об'єднують різні канали зв'язку в компанії, адміністратор таких рішень налаштовує та інтегрує окремі компоненти – електронну пошту, IP-телефонію та відеозв'язок, служби миттєвих повідомлень тощо. Це дозволяє співробітникам легко взаємодіяти між собою, проводити аудіо- чи відеоконференції [4].

Окрема категорія адміністраторів займається технічним обслуговуванням обладнання – комп'ютерної техніки, серверів, мережевих пристроїв. Вони відповідають за монтаж, налаштування, моніторинг, діагностику та ремонт апаратних компонентів IT-інфраструктури.

Таким чином, можна зробити висновок, що системне адміністрування – це комплексна послуга з підтримки IT-інфраструктури та інформаційних систем бізнесу. Вона включає кілька ключових аспектів, кожен з яких вимагає глибокої спеціалізації та уваги до деталей. Спільна робота команди адміністраторів дає змогу підтримувати високу продуктивність та надійність всіх технологічних систем компанії [3-4].

1.2 Системне адміністрування комп'ютерних мереж і серверів

Комп'ютерна мережа – це поєднання пристроїв та програмного забезпечення для спільного доступу користувачів до різноманітних ресурсів: технічних, інформаційних, програмних тощо [5].

Часті збої в роботі мережі вказують на глибинні проблеми, які потребують негайного вирішення. Причинами можуть бути застаріле обладнання, недоліки в системному адмініструванні, недостатня кваліфікація персоналу.

Нестабільна мережа не лише утруднює роботу співробітників, а й заважає бізнес-процесам. Тому не можна ігнорувати цю проблему. Саме тому

системне адміністрування та інженерія є вкрай важливими складовими IT-інфраструктури будь-якої компанії.

В обов’язки адміністраторів входить увесь комплекс заходів: від встановлення обладнання до моніторингу, оптимізації та профілактики. Це не просто техпідтримка, а ціла стратегія підтримки працездатності інформаційних систем на високому рівні.

Головною перевагою професійного системного адміністрування є чітко спланований підхід до виконання поточних завдань та подальшого розвитку IT-інфраструктури. Адміністратор повинен ретельно аналізувати потреби бізнесу, враховуючи специфіку діяльності компанії, особливості взаємодії підрозділів. Мережеве середовище має будуватися з огляду на всі ці фактори. Усі складові системи, незалежно від конфігурації, повинні бути надійними технічно та зручними для користувачів. Для визначення стратегії IT-розвитку проводиться комплексний аудит [5-6].

У повсякденній роботі адміністратори займаються налаштуванням обладнання, моніторингом мережі, профілактикою та усуненням проблем. Важливий аспект – забезпечення кібербезпеки шляхом запобігання зовнішнім загрозам, налаштування захисного ПЗ.

1.3 Основна інформація про сервери

Сервер – це пристрій, що надає великій кількості користувачів доступ до файлів, додатків, принтерів та інших мережевих ресурсів. Від його роботи залежить продуктивність усіх підключених клієнтських комп’ютерів. На відміну від звичайних робочих станцій, сервер обслуговує багато користувачів одночасно, тому пріоритетами є надійність, відмовостійкість і безперервна робота.

Адміністратор повинен вживати заходів для мінімізації простоїв і прискорення відновлення після збоїв, налаштовувати оптимальне середовище,

реалізовувати резервне копіювання та інші методи підвищення доступності. Проте можливості оптимізації обмежені величезною кількістю користувачів.

Фізично сервери відрізняються від звичайних ПК потужнішим обладнанням та іншою економічною моделлю. Вони встановлюються у спеціальних серверних приміщеннях із контрольованим доступом та мікрокліматом.

Сервер можна розглядати з точки зору апаратного та програмного забезпечення, кожен з яких потребує уваги адміністратора. З апаратної точки зору, сервер – це комп'ютер, виділений з групи ПК чи робочих станцій для виконання сервісних завдань з мінімальною участю людини або автономно. За конфігурацією сервер зазвичай схожий на потужну робочу станцію. Основна відмінність – ступінь залученості користувача. Деякі сервіси можуть виконуватись на звичайному ПК поряд з основними завданнями. Фізична консоль потрібна серверам переважно під час налаштування та обслуговування. Управління відбувається віддалено.

З програмної точки зору сервер – це компонент системи, що надає певний сервіс чи доступ до ресурсу за запитом клієнта. Реалізує концепцію «клієнт-сервер». Формат взаємодії визначається протоколом – зазвичай стандартизованим (RFC) для сумісності різних додатків. Сервер поєднує спеціалізоване обладнання та ПЗ для ефективного виконання сервісних завдань.

1.3.1 Апаратна реалізація сервера

Серверні системи суттєво відрізняються від настільних ПК за своїми характеристиками та можливостями. Іноді намагаються заощадити, використовуючи звичайне апаратне забезпечення з серверним ПЗ. Але для масштабних чи довгострокових проектів, де потрібна висока надійність, такий підхід неприйнятний. Професійне серверне обладнання має переваги, які виправдовують вищу ціну:

– можливості розширення – більше слотів для плат розширення, процесорів, накопичувачів, а також спеціалізовані високошвидкісні роз’єми для підключення периферії;

– потужні багатоядерні процесори з додатковими технологіями, зокрема перевіркою, кешуванням, динамічним розподілом ресурсів;

– високопродуктивний ввід-вивід, що важливо для обробки запитів великої кількості користувачів – швидкі накопичувачі, мережеві інтерфейси;

– додаткове серверне ПЗ забезпечує кластеризацію, резервування, автоматичне перемикання на бекап при збоях тощо.

Сервери часто модернізують, а не замінюють повністю, оскільки вони розраховані на зростаюче навантаження. Можлива поступова заміна процесорів на більш потужні без суттєвих змін в апаратурі – оскільки вони встановлені в окремих слотах чи знімних сокетах.

Також сервери обов’язково повинні передбачати можливість монтажу в спеціальні стійки, оптимізовані за габаритами та системою охолодження. Це вимагає використання прямокутних корпусів, зручного доступу до компонентів без виймання зі стійки і наскрізної вентиляції для відведення тепла. Замість придбання власних фізичних серверів, можна скористатися хмарними сервісами провідних постачальників – обчислювальні потужності, сховища даних та бази даних надаються на вимогу як послуга.

1.3.2 Хмарні обчислення для реалізації сервера

Хмарні технології (cloud computing) – це модель надання різноманітних IT-сервісів через мережу Інтернет: серверні потужності, мережі, моніторинг, сховища даних, програмне забезпечення, бази даних тощо (рис.1.1).

Такий підхід дозволяє швидко впроваджувати інновації, масштабувати ресурси за потребою, а також економити кошти. Оплачуються лише фактично використані сервіси, що знижує витрати і дає можливість гнучко розширюватися. Переваги хмарних технологій:

– економічна ефективність – не потрібно інвестувати в офлайн інфраструктуру наперед, а лише оплачувати фактичне використання.

– масштабованість – можливість швидкого нарощування або зменшення ресурсів по потребі завдяки гнучким інструментам моніторингу.

– доступність – можливість отримати доступ до потрібних IT ресурсів (сервери, сховища, мережі тощо) протягом кількох хвилин, що підвищує оперативність бізнесу.

– швидкість розгортання – просте і швидке виведення сервісу на ринок нового регіону завдяки розгортанню хмарної інфраструктури за лічені хвилини. це забезпечує кращий досвід клієнтів з мінімальними витратами (рис.1.2).

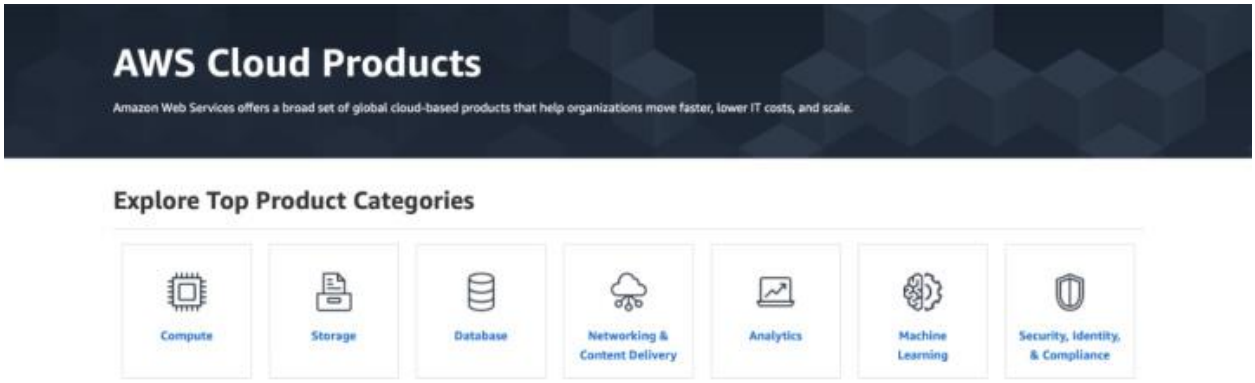

Рисунок 1.1 – Продукти які доступні для роботи в хмарі AWS

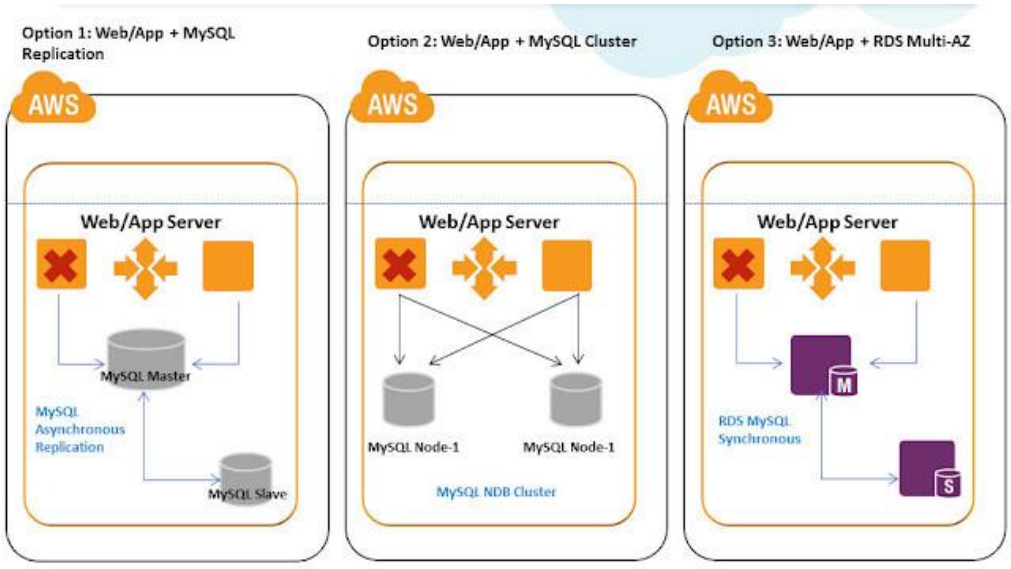

Рисунок 1.2 – Приклад технологій розгортання веб-застосунку

Можна зробити висновок, що хмарні технології дають свободу у швидкому масштабуванні і отриманні потрібних ресурсів та сервісів від провайдера на вимогу за хвилини.

1.4 Серверні операційні системи

Вибір серверної операційної системи завжди є предметом дискусій в IT-індустрії. З одного боку, вона споживає ресурси сервера, які можна було б використати на інші цілі. З іншого боку, ОС є своєрідним диригентом, що перетворює комп'ютер на багатозадачну платформу і полегшує взаємодію різних додатків з апаратурою. Щоб обрати оптимальну ОС, потрібно розуміти особливості популярних на даний момент систем. Основними серверними ОС є – Windows Server та різні дистрибутиви Linux. Кожна має свої переваги, недоліки та ніші для специфічного застосування. Розглянемо найпоширеніші з них детальніше.

1.4.1 Windows Server

Windows Server – популярна корпоративна операційна система від Microsoft, незважаючи на те, що більшість користувачів асоціюють цей бренд виключно з настільною Windows. Залежно від завдань компанії, можна обрати Windows Server 2003 для сумісності із застарілим ПЗ, або новіші версії – Windows Server 2019 чи Windows Server 2022.

Переваги Windows Server:

– простота адміністрування та велика кількість навчальних матеріалів;
– сумісність із програмами, що використовують бібліотеки Microsoft;
– технологія віддаленого доступу RDP;
– універсальність і наявність полегшеної версії Server Core.

Недоліки:

– велике споживання апаратних ресурсів (1+ ядро CPU, 3+ Гб ОЗУ);
– висока вартість ліцензій;

– певні проблеми безпеки через RDP та політики користувачів.

З наведеного можна зробити висновок, що Windows Server підходить для середовищ на базі рішень Microsoft, але вимагає потужного обладнання.

1.4.2 Ubuntu

Ubuntu на сьогодні є одним з найпоширеніших і найстабільніших дистрибутивів операційної системи Linux. Завдяки величезній спільноті користувачів та регулярним оновленням, він став найбільш використовуваною серверною ОС. На відміну від Windows Server, який частіше застосовується для підтримки спеціалізованого програмного забезпечення, Ubuntu як дистрибутив Linux є кращим вибором для веб-розробки та роботи з відкритим кодом. Саме Linux-сервери широко використовуються для розгортання веб-серверів Apache або Nginx, СУБД PostgreSQL і MySQL, а також популярних скриптових мов програмування (рис.1.3). Ubuntu Server добре підходить для встановлення всіх необхідних сервісів маршрутизації та керування трафіком.

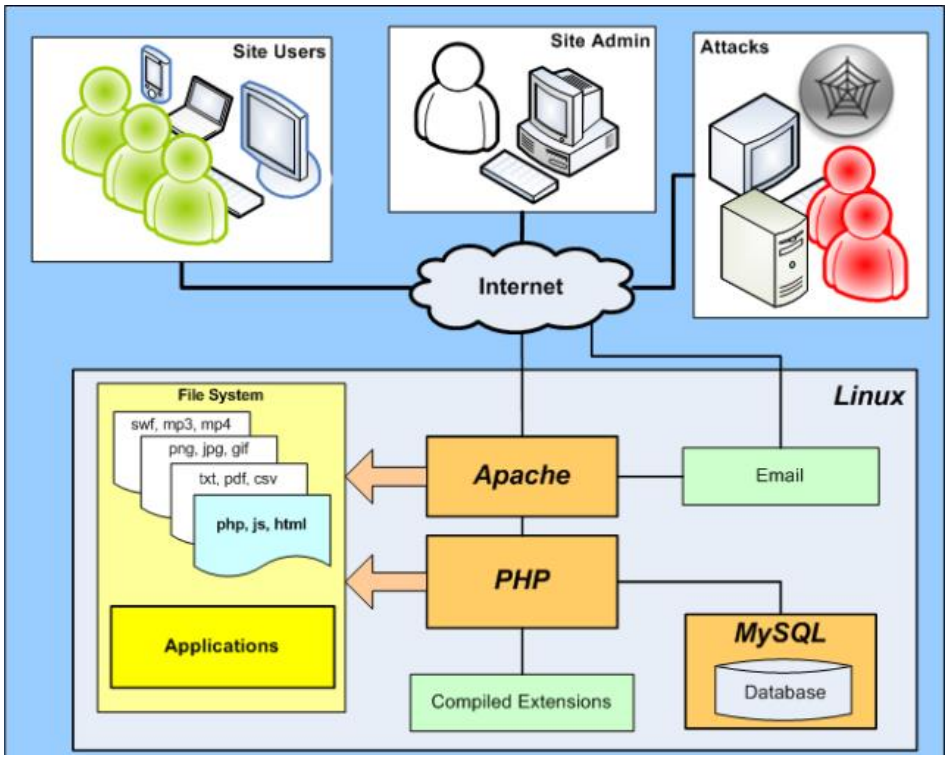

Рисунок 1.3 – Приклад розгортання застосунку на Linux

Серед переваг Ubuntu варто відзначити значно менше споживання ресурсів порівняно з Windows Server, а також можливість роботи в консолі та з пакетними менеджерами, характерну для Linux-систем. Крім того, простота та зручність Ubuntu як домашньої ОС полегшує її адміністрування навіть для не дуже досвідчених користувачів.

Головним недоліком цієї ОС є потреба у певних професійних навичках, особливо для роботи виключно через термінал, що необхідно для серверних конфігурацій. Ще одна особливість полягає в тому, що Ubuntu більше підходить для персонального користування і не завжди може задовольнити корпоративні потреби.

1.4.3 Debian

Debian, як один із найстаріших дистрибутивів операційної системи Linux, є основою для розробки і більш популярного дистрибутиву Ubuntu. Він відзначається вищим рівнем безпеки та стабільності порівняно з Ubuntu та Windows, і, таким чином, виявляється більш привабливим для використання у серверних застосуваннях. Однією з ключових переваг Debian є ефективне використання ресурсів, особливо при роботі через командний рядок, властивість, яка є характерною для більшості Linux-систем.

Додатковою перевагою Debian є активна спільнота, яка відзначається відданістю і ефективним взаємодією з вільним та відкритим програмним забезпеченням, що є основою операційної системи. Проте, гнучкість Debian може також виявити свої обмеження. Розробка Debian підтримується спільнотою без чітко визначеного ядра розробників, що призводить до наявності трьох основних гілок розробки: стабільної (stable), тестової (testing) та нестабільної (unstable).

Ця проблема виражається в тому, що оновлення в стабільній гілці значно відстають за функціональністю від тестової гілки, що може вимагати власноручного компілювання ядра або переходу до нестабільної гілки.

Навпаки, Ubuntu регулярно випускає стабільні версії з підтримкою довгострокового зберігання (LTS) раз в два роки.

Отже, вибір операційної системи повинен відповідати конкретним вимогам і завданням, і враховувати рівень стабільності та актуальності, необхідний для вашого проекту.

1.5 Постановка задачі

Завдання для виконання кваліфікаційної роботи можна розділити на кілька послідовних етапів:

1. Літературний огляд та аналіз: провести літературний огляд з питань віртуалізації та інструмента Vagrant. Аналізувати переваги та обмеження використання цього інструмента в розробці програмного забезпечення.

2. Практична реалізація засобу налаштування Vagrant: розробити конфігураційний інструмент, який дозволить налаштовувати та розгортати віртуальні машини за допомогою Vagrant. Включити можливості налаштування операційних систем, ресурсів та програмного забезпечення.

3. Автоматизація налаштування віртуальних машин: розробити систему автоматизації для налаштування та встановлення віртуальних машин, використовуючи Vagrant та інші інструменти (наприклад, shell-скрипти, Ansible).

4. Тестування та експерименти: організація тестових сценаріїв та проведення експериментів для оцінки продуктивності та ефективності використання Vagrant у розробці програмного забезпечення.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Обґрунтування вибору ОС Linux

Серверне застосування є однією з ключових сфер для операційної системи Linux. Ця ОС добре масштабується, адаптується до різноманітних апаратних та програмних платформ, оптимізована для багатокористувацької роботи. Слід відзначити підвищену надійність, стабільність роботи, високий рівень захисту та гнучкі можливості налаштування, що є критично важливим для серверних систем.

У галузі інформаційних технологій широко поширене використання Linux як десктопної операційної системи. Завдяки своїм технічним перевагам вона підходить для розробки програмного забезпечення, налаштування комп’ютерних мереж, розгортання серверів, моніторингу систем тощо. Серверна версія Linux без графічного інтерфейсу дозволяє ефективно працювати через термінальні текстові команди, що особливо зручно при віддаленій адміністрації через протокол SSH.

Сервери на базі дистрибутива Ubuntu широко застосовуються для веб-сайтів, хостингу програмного забезпечення та інших завдань. Як тестова платформа тут розглядається Ubuntu Server. Саме цей дистрибутив найчастіше обирають для серверних рішень, згідно зі статистикою. На хмарних серверах Amazon EC2 частка Ubuntu переважає з 300 тисячами екземплярів, що втричі більше за Debian. Найпопулярнішими версіями є Ubuntu Server LTS з тривалим терміном підтримки.

2.2 Вибір автоматизованого підходу до розгортання серверів

Опис концепцій віртуалізації та Vagrant

Віртуалізація є технологією, яка дозволяє створювати віртуальні екземпляри обчислювальних ресурсів (віртуальні машини) на базі фізичного

апаратного забезпечення. Це дозволяє одному фізичному серверу виконувати декілька ізольованих екземплярів операційних систем, що працюють незалежно один від одного.

Основні переваги віртуалізації включають:

– ефективне використання ресурсів: віртуалізація дозволяє оптимізувати використання апаратних ресурсів, таких як процесор, пам’ять, диск і мережа. ви можете запускати багато віртуальних машин на одному фізичному сервері;

– ізоляція та безпека: кожна віртуальна машина ізольована від інших, що дозволяє забезпечити безпеку та надійність додатків. якщо одна віртуальна машина виявить проблеми, це не вплине на роботу інших;

– швидкість розгортання: віртуальні машини можна створювати та розгортати швидко, що дозволяє швидко реагувати на зміни в інфраструктурі;

– підтримка тестування і розробки: віртуалізація сприяє розгортанню віртуальних середовищ для тестування та розробки, що полегшує роботу розробників та тестувальників.

Vagrant – це інструмент для управління віртуальними машинами та розгортання розробкових середовищ (рис.2.1). Основні концепції та архітектурні складові Vagrant включають:

– Vagrantfile: це текстовий файл, що містить опис конфігурації віртуальної машини. У ньому визначаються параметри, такі як образ операційної системи, ресурси (процесор, пам’ять, диск), мережеві налаштування та провіжіонінг (скрипти для автоматичної настройки віртуальної машини).

– Провіжіонінг: це процес автоматичної настройки та установки ПЗ на віртуальній машині. Ви можете використовувати різні інструменти для провіжіонінгу, такі як shell-скрипти, Ansible, Puppet або Chef.

– Постачальники (Providers): Vagrant підтримує різні постачальники віртуальних машин, такі як VirtualBox, VMware, Hyper-V тощо. Вибір

постачальника дозволяє вам використовувати різні віртуальні оточення в залежності від вашої потреби.

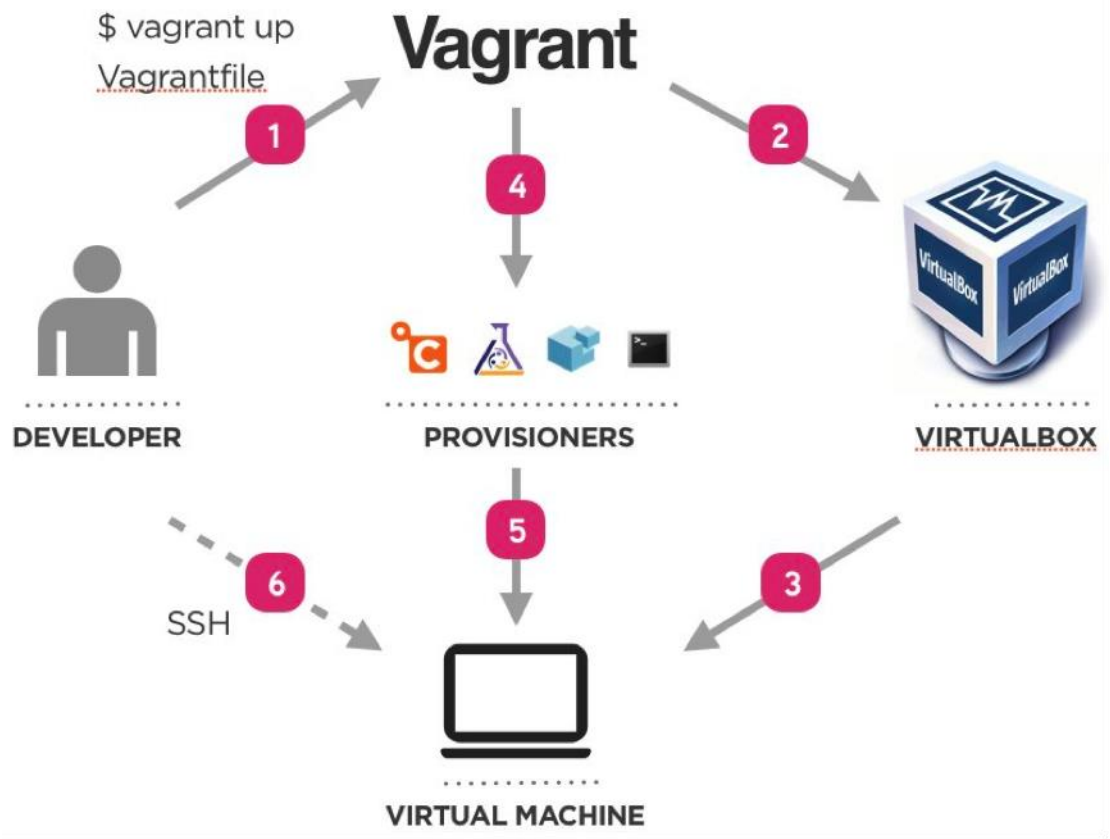

Рисунок 2.1 – Схематичне зображення компонентів інструменту Vagrant

– Vagrant Cloud: Це хмарний сервіс, який дозволяє зберігати та обмінюватися Vagrant-конфігураціями та образами. Ви можете використовувати Vagrant Cloud для зберігання та спільного використання своїх конфігурацій.

– Команди Vagrant: Vagrant надає командний інтерфейс для управління віртуальними машинами. Системний адміністратор або ж звичайний користувач може використовувати команди для запуску, зупинки, перезавантаження та інших операцій з віртуальними машинами.

Розширюваність Vagrant полягає в тому, що доступна можливість розширювати його можливості за допомогою плагінів. Плагіни дозволяють додавати нові функції та інтеграцію з іншими інструментами. Користувач може встановлювати плагіни за допомогою vagrant plugin install та

використовувати їх для автоматизації різних завдань, розширюючи функціональність Vagrant за власними потребами.

Узагальнюючи, Vagrant спрощує роботу з віртуальними машинами, надаючи конфігурацію та автоматизовану настройку через Vagrantfile і провіжіонінг. Він також дозволяє розширювати свої можливості за допомогою плагінів та інтегрувати з різними віртуальними платформами за допомогою постачальників.

2.3. Розробка графічного інтерфейсу

Графічний інтерфейс користувача має особливе значення при розробці програмного забезпечення для системних адміністраторів. Зручний GUI дозволяє спростити виконання складних задач адміністрування та підвищити продуктивність роботи адміністраторів. Завдяки інтуїтивно зрозумілому дизайну адміністратор може швидко знаходити потрібні інструменти та функції, не витрачаючи час на пошук у командному рядку. Рисунок 2.2. показує можливі дії, які має мати змогу виконувати системний адміністратор за допомогою графічного інтерфейсу.

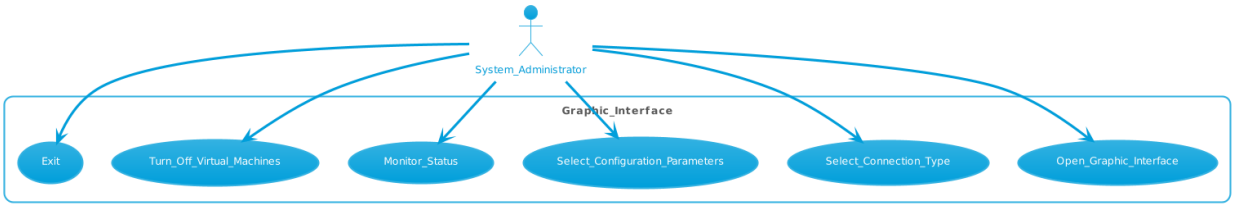

Рисунок 2.2 – Демонстрація дій за допомогою графічного інтерфейсу

Візуальне подання статистики, графіків, схем значно полегшує аналіз та вирішення проблем. Можливість гнучкої настройки інтерфейсу дозволяє адаптувати програму як для новачків, так і досвідчених фахівців. Загалом, зручний GUI робить роботу адміністратора ефективнішою, підвищує задоволеність від роботи з програмою та може зменшити кількість помилок,

пов’язаних зі складністю використання. Тому розробка гарного графічного інтерфейсу є дуже важливою при створенні систем адміністрування.

JavaFX – це сучасний інструментарій для створення графічних інтерфейсів користувача на мові програмування Java. Вибір JavaFX для розробки GUI має низку переваг:

– кросплатформенність. Завдяки віртуальній JVM-машині додатки JavaFX працюють на різних операційних системах - Windows, Linux, macOS без додаткових зусиль;

– потужні засоби візуалізації. JavaFX має широкий набір компонентів і можливостей для створення сучасних анімованих інтерфейсів з використанням CSS і FXML;

– інтеграція з Java. JavaFX повністю інтегрований в екосистему Java, дозволяє використовувати всі переваги цієї мови і код Java безпосередньо в GUI;

– зручний Scene Builder. Цей візуальний редактор дозволяє швидко проектувати інтерфейс за допомогою драг-енд-дроп елементів;

– відмінна продуктивність. JavaFX оптимізований для створення швидких і надійних додатків зі складним інтерфейсом.

Виділимо основні ключові моменти які стосуються графічного інтерфейсу:

1. Обробка подій: реалізовано базову функціональність кнопок для розгортання та налаштування віртуальних машин. Кнопки виконують команди терміналу для автоматизації рутинних завдань. Це спрощує взаємодію користувача.

2. Інтерфейс: використовується мінімалістичний дизайн без зайвих елементів. Це полегшує сприйняття і зосередження на основних функціях. Відображаються лише найнеобхідніші елементи управління, кольорова гамма обмежена.

3. Пакування та розповсюдження: програма та необхідні файли розпаковані в одному виконуваному архіві. Це спрощує розгортання та використання. Архів легко розповсюджувати серед кінцевих користувачів.

Загалом, дизайн та функціонал зведені до мінімуму, аби зосередитися на вирішенні основних завдань.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СЕРВЕРА НА БАЗІ ОС UBUNTU

Для ефективної роботи та повноцінного використання необхідних інструментів було вирішено розгорнути віртуальну машину на основі операційної системи Ubuntu. Для створення віртуальної машини було вирішено скористатися програмними засобами VirtualBox та Vagrant.

VirtualBox надає платформу для створення віртуальної машини з використанням ресурсів фізичного комп'ютера. Цей софт є добре відомим більшості системних інженерів і часто застосовується завдяки широким функціональним можливостям. Віртуальна машина створює власне ізольоване середовище на реальній машині, що складається з віртуальних компонентів: процесора, жорсткого диску, оперативної пам'яті тощо.

Процес встановлення VirtualBox є досить простим і зрозумілим. На відміну від нього, налаштування Vagrant вимагає певних теоретичних знань користувача та навичок роботи з консоллю. Vagrant дозволяє створювати та конфігурувати легке, переносне і повторюване середовище для розробки. Він забезпечує: ізольованість, уникаючи конфліктів з іншими програмами; повторюваність, даючи змогу швидко відтворити робоче середовище; а також переносність, дозволяючи розгортати потрібне оточення універсальним способом на будь-якій системі. Саме це є ключовим критерієм вибору даного інструменту.

3.1 Опис конфігурації віртуальних машин у Vagrantfile

Конфігурація віртуальних машин у файлі Vagrantfile показана на рисунку 3.1 є важливою частиною практичної реалізації використання інструменту Vagrant. Код конфігурації написано мовою Ruby при збереженні не обов'язково зазначати формат розширення, оскільки інформація з нього зчитується автоматично при його визначенні. У цьому файлі адміністратор визначає всі необхідні параметри та налаштування для віртуальної машини.

```
Vagrant.configure("2") do |config|
  N = 4
  (1..N).each do |i|
    config.vm.define "server#{i}" do |server|
      server.vm.box = "ubuntu/bionic64"
      server.vm.network "private_network", type: "dhcp"

      server.vm.provider "virtualbox" do |vb|
        vb.memory = "512"
        vb.cpus = 1
      end

      if i == 1
        server.vm.provision "shell", path: "scripts/setup_load_balancer.sh"
      elsif i <= 3

        server.vm.provision "shell", path: "scripts/setup_web_server.sh"
      else

        server.vm.provision "shell", path: "scripts/setup_database_server.sh"
      end
    end
  end
end
```

Рисунок 3.1 – Файл конфігурації Vagrantfile

Розглянемо приклад Vagrantfile та пояснення ключових параметрів:

– config.vm.define – цей параметр задає базовий образ операційної системи, який буде використано для віртуальної машини. У даному випадку це Ubuntu, проте за потреби можна обрати готовий образ з репозиторію Vagrant Cloud чи іншого джерела;

– server.vm.provider – у цьому блоці вказуються параметри постачальника віртуальної інфраструктури, наприклад VirtualBox. Тут задають необхідну кількість пам'яті, процесорних ядер тощо. Виходячи з обмежених ресурсів, в прикладі вказані мінімальні параметри. Проте для хмарних середовищ чи фізичних серверів ці параметри можуть бути значно вищими.;

– server.vm.network – дозволяє налаштувати мережеві параметри віртуальної машини. У прикладі використовується private_network для отримання IP-адреси через DHCP;

– server.vm.provision – це параметр, який вказує на те як буде проведено налаштування і установка програмного забезпечення на віртуальні машини. Це поле вказує, які команди чи сценарії виконуватимуться на віртуальній машині після її створення або під час процесу провізіонування (у контексті віртуалізації та управління інфраструктурою означає процес налаштування, наступного розгортання та встановлення необхідного програмного забезпечення на віртуальних машинах або фізичних серверах. Це дозволяє автоматизувати процес створення та налаштування середовища для додатків або системи, щоб забезпечити їхню працездатність і готовність до роботи). У нашому випадку процес відбувається за допомогою shell-скрипту setup.sh.

Це базовий приклад конфігурації, до якого за необхідності можна додавати додаткові параметри та налаштування відповідно до потреб. Після створення файлу Vagrantfile, для роботи (запуску) необхідно використовувати команди

– vagrant up − використовується для створення та запуску віртуальних машин на основі конфігурацій, визначених у Vagrantfile;

– vagrant halt − використовується для зупинки або вимкнення віртуальних машин. Усі процеси та ресурси, пов’язані з віртуальними машинами, будуть призупинені;

– vagrant provision − використовується для виконання скриптів або процедур на вже запущених віртуальних машинах, можна використовувати цю команду для оновлення конфігурації вже запущених машин.

3.2 Опис компонентів проєкту

З метою спрощення або певною мірою пришвидшення процесу розгортання віртуальних машин для адміністраторів, нами було розроблено графічний інтерфейс користувача (рис. 3.2) на основі бібліотеки Java FX. Такий підхід дозволяє спростити адміністрування та моніторинг процесу створення віртуальних машин за рахунок інтуїтивно зрозумілого графічного

інтерфейсу та вікна терміналу. Це дозволяє виконувати налаштування віртуальних машин поетапно та послідовно. Одночасно з графічним інтерфейсом відкривається вікно терміналу, що надає можливість спостерігати за перебігом процесу ініціалізації в режимі реального часу.

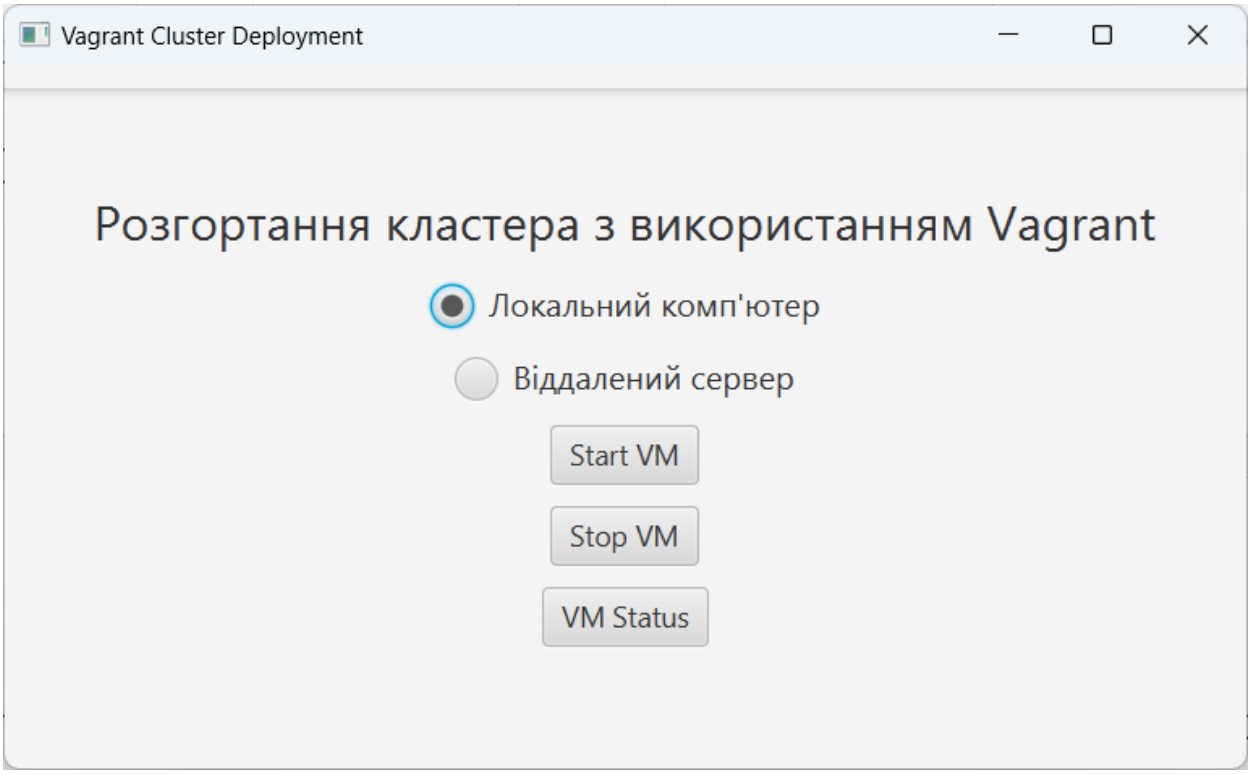

Рисунок 3.2 – Початок процесу розгортання кластера

На першому етапі користувач повинен обрати місце розгортання віртуальних машин, оскільки підходи відрізняються для локальної та віддаленої ініціалізації. У разі вибору локального комп’ютера через команду «cmd» відкривається командний рядок, де безпосередньо викликається інструмент Vagrant. Для віддаленого підключення до серверу спочатку здійснюється з’єднання за допомогою протоколу SSH з використанням IP-адреси. Приклад реалізації методу executeRemoteCommand() для такого підключення має наступний вигляд:

```
private void executeRemoteCommand(String command) {
    Properties properties =
loadConfigFromFile(CONFIG_FILE_PATH);
```

```
    if (properties == null) {
        System.err.println("Error loading configuration from
file.");
        return;
    }

    String user = properties.getProperty("ssh.user");
    String password =
properties.getProperty("ssh.password");
    String host = properties.getProperty("ssh.host");
    int port =
Integer.parseInt(properties.getProperty("ssh.port", "22"));

    try {
        JSch jsch = new JSch();
        Session session = jsch.getSession(user, host, port);
        session.setPassword(password);
        session.setConfig("StrictHostKeyChecking", "no");
        session.connect();

        ChannelExec channel = (ChannelExec)
session.openChannel("exec");
        channel.setCommand(command);
        channel.connect();

        try (BufferedReader reader = new BufferedReader(
                new
InputStreamReader(channel.getInputStream()))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        }

        int exitStatus = channel.getExitStatus();
        channel.disconnect();
        session.disconnect();

        if (exitStatus < 0) {
            System.out.println("Done, but exit status not
set!");
        } else if (exitStatus > 0) {
            System.out.println("Done, but with error!");
        } else {
            System.out.println("Done!");
        }

    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
        e.printStackTrace();
    }
}
```

Після вибору користувачем способу підключення (локального чи віддаленого) відбувається перехід до наступного етапу ініціалізації віртуальних машин шляхом натискання кнопки «Start VM». Це запускає виконання конфігураційного файла Vagrantfile та розпочинає процес створення віртуальних машин. Оскільки це ресурсоємний процес, він потребує певного часу для завершення. На рисунку 3.3 представлено приклад візуалізації перебігу ініціалізації віртуальних машин у режимі реального часу через графічний інтерфейс програми.

Такий підхід дозволяє користувачу контролювати процес створення віртуальних машин та отримувати візуальне підтвердження успішності виконання кожного етапу ініціалізації. Зворотний зв'язок у режимі реального часу підвищує зручність використання програми для адміністраторів при розгортанні віртуальної інфраструктури.

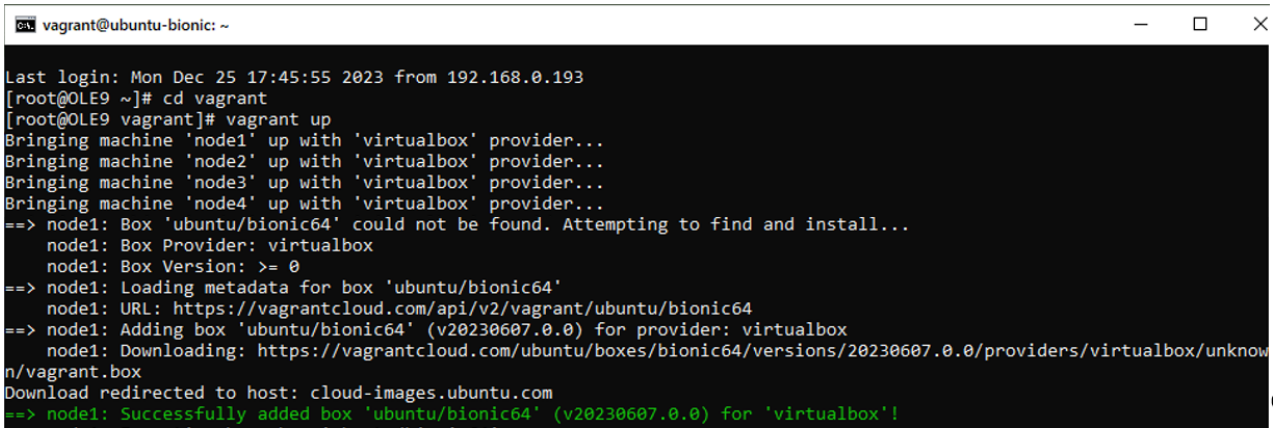

Рисунок 3.3 – Процес створення 4-х віртуальних машин

Після завершення етапу ініціалізації віртуальних машин користувач має можливість перевірити їх поточний статус, натиснувши кнопку «VM Status» в інтерфейсі програми. Це відкриває вікно зі списком усіх створених віртуальних машин та їхнім поточним станом (рис. 3.4). Таким чином користувач може пересвідчитися, що процес ініціалізації пройшов успішно і всі віртуальні машини запущені та готові до подальшої роботи.

Наступним кроком є підтвердження успішності створення віртуальних машин. Для цього відкривається додаткове діалогове вікно (рис. 3.5), де користувач повинен вказати, чи були віртуальні машини успішно проініціалізовані відповідно до заданих параметрів. Це дозволяє додати ще один рівень перевірки коректності проходження усього процесу.

Описаний функціонал дає можливість користувачу контролювати і верифікувати успішність створення та запуску віртуальних машин за допомогою інтуїтивного графічного інтерфейсу програми.

```
[root@OLE9 ~]# vagrant global-status
id       name   provider   state   directory
-----------------------------------------------------------------
6beece7  node1  virtualbox running /root/vagrant
6e96248  node2  virtualbox running /root/vagrant
477135a  node3  virtualbox running /root/vagrant
dd41c52  node4  virtualbox running /root/vagrant
```

Рисунок 3.4 – Перевірка статусу створених віртуальних машин

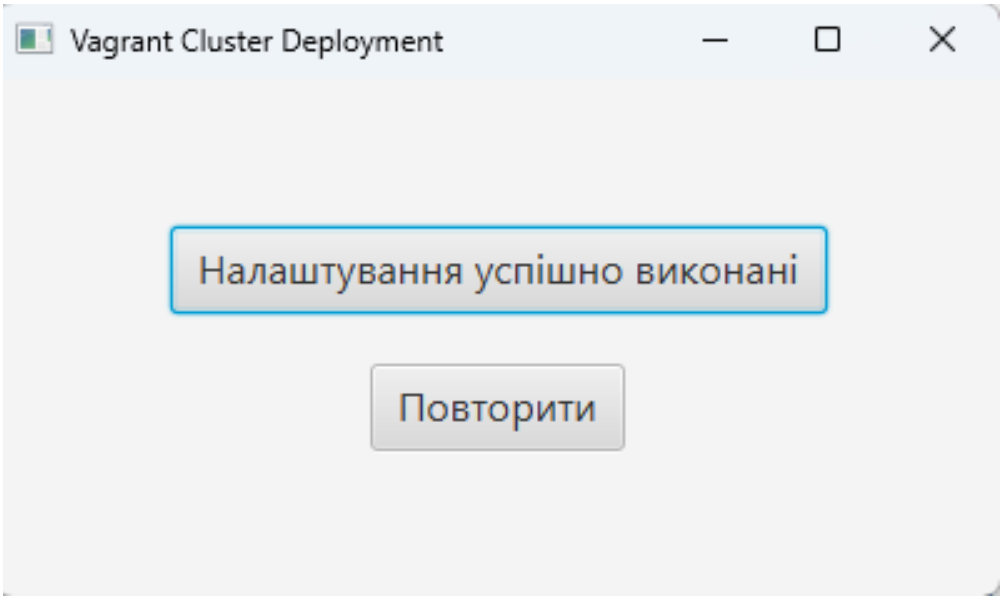

Рисунок 3.5 – Перевірка успішного сценарію створення віртуальних машин

У разі успішного завершення попередніх етапів налаштування та ініціалізації, відкривається наступне вікно (рис. 3.6), що надає можливість

подальшої конфігурації віртуальних машин. Зокрема, користувач може виконати такі дії:

– додати необхідну кількість користувачів на певну віртуальну машину, вибравши її номер з розкривного списку;

– вказати перелік програмного забезпечення для встановлення та обрати цільову віртуальну машину для його деплойменту;

– запустити процес установки ПЗ та конфігурації користувачів за допомогою кнопки «Прийняти».

– відмовитися від змін за допомогою кнопки «Відхилити».

– перейти до головного вікна програми за допомогою відповідної кнопки вгорі, щоб мати можливість зупинки або перевірки поточного статусу віртуальних машин.

Даний функціонал модального вікна дозволяє гнучко керувати процесом налаштування та конфігурації віртуального середовища після початкового етапу створення віртуальних машин. Інтерфейс користувача оптимізовано для зручності виконання типових задач адміністрування.

Після завершення процесу налаштування та конфігурації віртуального середовища, адміністратор має можливість самостійно перевірити коректність виконаних дій. Для цього можна скористатися наступними варіантами:

– виконати перевірку поточних налаштувань за допомогою необхідних команд терміналу. Це дозволяє безпосередньо проаналізувати стан віртуальних машин та їх конфігурації (рис.3.7, рис.3.8);

– дочекатися завершення усього процесу налаштування та скористатися кнопкою «VM Status» в інтерфейсі програми. Це надасть можливість переглянути підсумковий статус віртуальних машин та їх параметри (рис.3.9);

– поєднати обидва підходи, спочатку перевіривши конфігурацію через термінал, а потім остаточно переконатися у коректності за допомогою графічного інтерфейсу.

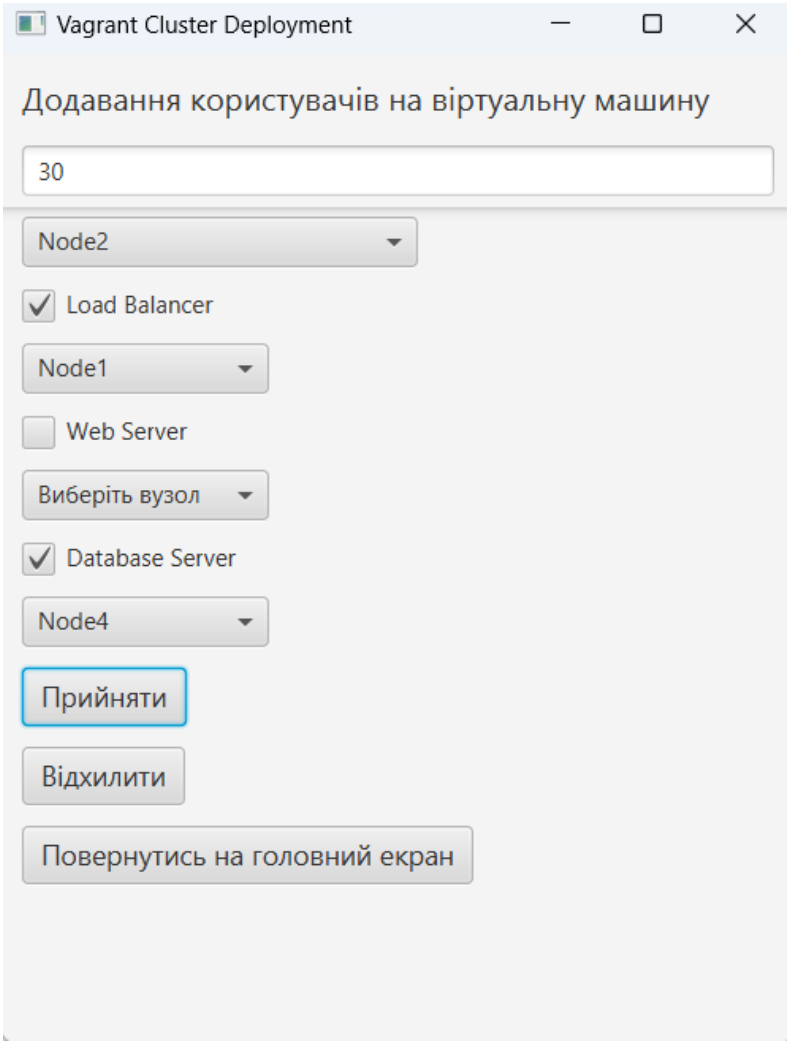

Рисунок 3.6 – Подальша конфігурація віртуальних машин

```
vagrant@ubuntu-bionic:~$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2023-12-25 22:25:30 UTC; 1min 10s ago
     Docs: man:nginx(8)
 Main PID: 2924 (nginx)
    Tasks: 2 (limit: 546)
   CGroup: /system.slice/nginx.service
           ├─2924 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
           └─2925 nginx: worker process
```

Рисунок 3.7 – Процес ручної перевірки стану nginx

```
vagrant@ubuntu-bionic:~$ systemctl status mysql
● mysql.service - MySQL Community Server
   Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2023-12-25 22:31:30 UTC; 1min 27s ago
 Main PID: 3091 (mysqld)
    Tasks: 27 (limit: 546)
   CGroup: /system.slice/mysql.service
           └─3091 /usr/sbin/mysqld --daemonize --pid-file=/run/mysqld/mysqld.pid
```

Рисунок 3.8 – Процес ручної перевірки стану mysql-server

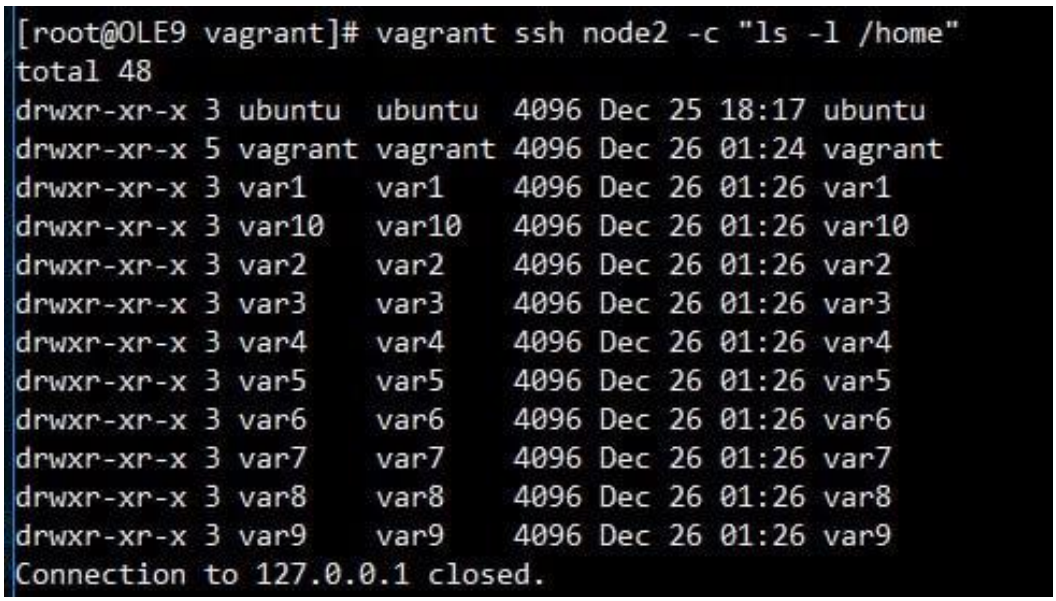

Рисунок 3.9 – Процес перевірки створення користувачів на обраній віртуальній машині

У Додатку А наведено частковий код, який відповідає за візуальну частину створення графічного інтерфейсу, тобто вікна, у які необхідно вносити значення.

Далі необхідно продумати про балансування навантаження. При використанні Vagrant для створення кластеру віртуальних машин балансування навантаження за допомогою Nginx може знадобитися для ситуації, якщо одна з віртуальних машин виходить з ладу або перезавантажується, у такому випадку Nginx автоматично спрямовує трафік на решту серверів завдяки балансуванню. Таким чином ми отримуємо масштабований та відмовостійкий кластер віртуальних машин для веб-сервісу.

З точки зору налаштувань необхідно виконати наступні кроки:

1. створити кілька однакових за налаштуваннями віртуальних машин за допомогою Vagrant (у конфігураційному файлі зазначається вся необхідна інформація) з встановленим веб-сервером Nginx.

2. встановити окрему віртуальну машину з Nginx як балансувальник навантаження. Дану віртуальну машину необхідно конфігурувати за допомогою upstream на адреси веб-серверів ВМ з кроку 1.

Кластер можна легко масштабувати, додаючи нові ВМ через Vagrant і дописуючи їх в upstream конфігурацію Nginx.

Автоматичне балансування навантаження за допомогою інструментів, таких як Nginx, має на меті покращення продуктивності та доступності сервісу, шляхом розподілу запитів між декількома серверами, дозволяючи обробляти більше запитів за один і той же проміжок часу. Крім того, можна проводити технічні роботи на окремих серверах без зупинки сервісу, оскільки балансувальник перенаправляє трафік на працюючі сервери. Використання менш потужних, але більш доступних серверів, об’єднаних у кластер, дозволяє оптимізувати використання ресурсів та витрат.

Резервне копіювання даних є вкрай важливим елементом у забезпеченні безперебійної роботи інфраструктури компанії або просто кластеру. Регулярне створення резервних копій дозволяє уникнути втрати цінних даних внаслідок збоїв програмного чи апаратного забезпечення, кібератак або помилок персоналу.

Для автоматизації процесу резервного копіювання було налаштовано щоденне копіювання логів, файлів та баз даних наших серверів на зовнішній NAS, а також щотижневе копіювання найбільш критичних даних на хмарне сховище S3 для більш тривалого зберігання. Така комплексна політика резервного копіювання суттєво підвищує захищеність даних.

Щоб створити скрипт на основі rsync для регулярного резервного копіювання даних було створено файл backup.sh з наступним змістом:

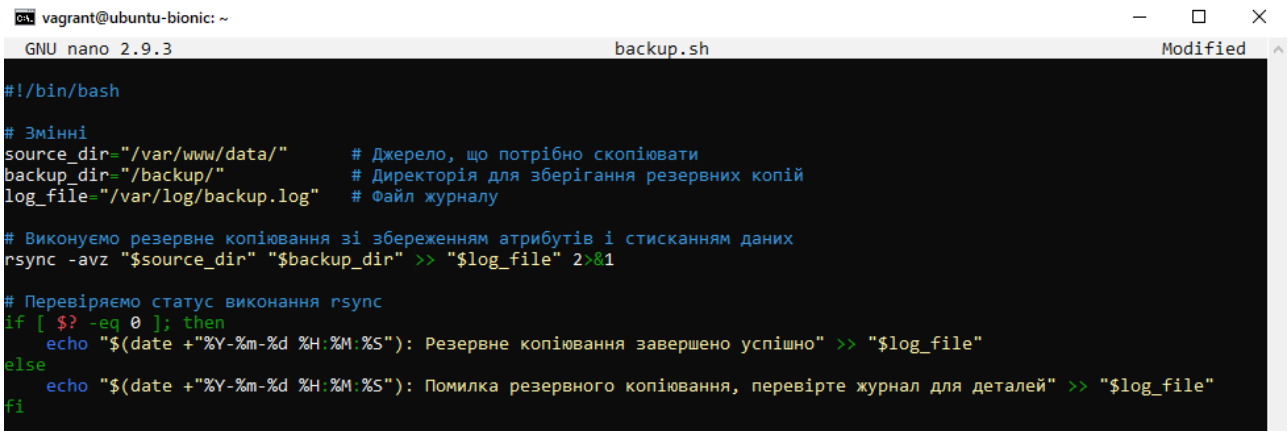

Рисунок 3.10 – Налаштування резервного копіювання

Даний скрипт можна додати до розкладу регулярного запуску, що цілком є логічним налаштуванням. Після оптимізації даної конфігурації налаштування буде додано моніторинг і логування процесів розгорнутого кластера. Ці процеси допомагають забезпечити стабільну та надійну роботу системи, а також вчасно виявляти та вирішувати проблеми. Правильно налаштована система моніторингу і логування допомагає операторам та адміністраторам ефективно управляти інфраструктурою та вчасно реагувати на будь-які аномалії чи події.

3.3 Тестування працездатності

Для аналізу ефективності налаштувань і надійності тестового кластера розглянемо наступні тест-кейси:

Тест-кейс 1: Зупинка одного сервера;

Тестуємо, як кластер реагує на відключення одного сервера.

Тест-кейс 2: Зупинка кількох серверів (для цього кроку було створено 10 серверів розгорнувши ті ж налаштування що і на початку):

```
[root@OLE9 ~]# cd vagrant
[root@OLE9 vagrant]# vagrant up
Bringing machine 'node1' up with 'virtualbox' provider...
Bringing machine 'node2' up with 'virtualbox' provider...
Bringing machine 'node3' up with 'virtualbox' provider...
Bringing machine 'node4' up with 'virtualbox' provider...
Bringing machine 'node5' up with 'virtualbox' provider...
Bringing machine 'node6' up with 'virtualbox' provider...
Bringing machine 'node7' up with 'virtualbox' provider...
Bringing machine 'node8' up with 'virtualbox' provider...
Bringing machine 'node9' up with 'virtualbox' provider...
Bringing machine 'node10' up with 'virtualbox' provider...
```

Рисунок 3.11 – Створення тестових 10 серверів

Тестуємо, як кластер реагує на відключення декількох серверів.

Тест-кейс 3: Відновлення відключених серверів.

Тестуємо, як кластер реагує на відновлення раніше відключених серверів.

Тест-кейс 1 кроки до створення:

1. запускаємо кластер з трьома віртуальними машинами за допомогою Vagrant;

2. відкриємо термінал і переходимо в папку з проєктом.

3. у терміналі введимо команду vagrant halt node2 для зупинки сервера;

4. спостерігаємо за реакцією кластера та балансуванням навантаження.

Результат позитивний: кластер продовжує працювати коректно після зупинки сервера №2. Балансування навантаження робочого навантаження відбувається без втрати функціональності.

Тест-кейс 2 кроки до створення (оскільки тут змінюються тільки кроки впливу на сервери перші два кроки і останній не дублюємо): у терміналі вводимо команду vagrant halt node2 і vagrant halt node3 для зупинки серверів.

Результат позитивний: кластер продовжує працювати коректно після зупинки серверів. Балансування навантаження робочого навантаження відбувається без втрати функціональності.

Тест-кейс 3 кроки до створення:

1. у терміналі вводимо команду vagrant halt node2 для зупинки сервера;

2. після зупинки, вводимо команду vagrant up server2 для відновлення.

Результат позитивний: другий сервер успішно піднімається. Кластер автоматично включає його до балансування навантаження.

Таким чином розробку інформаційної технології налаштування, розгортання та контролю віртуальних машин за допомогою Vagrant можна вважати успішною. У подальшому система буде допрацьована для практичного застосування.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було зроблено наступне:

1. Проведено детальний аналіз про системне адміністрування і концепції віртуалізації та інструменту Vagrant. Визначено переваги та недоліки використання Vagrant для розгортання віртуальних середовищ розробки.

2. Розроблено конфігураційний механізм на основі Vagrant, який дозволяє гнучко налаштовувати і розгортати віртуальні машини з різними операційними системами та програмним забезпеченням.

3. Створено графічний інтерфейс для системного адміністратора для встановлення та налаштування віртуальних машин на базі Vagrant та bash-скриптів. Це дозволяє прискорити і спростити розгортання віртуальних середовищ.

4. Проведено тестування розробленого та підтверджено ефективність використання Vagrant для прискорення налаштування середовищ розробки.

Результати роботи можуть бути використані для подальших досліджень віртуалізації та розгортання хмарних середовищ на основі Vagrant.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Онлайн курс \"Системне адміністрування і послуги IT-інфраструктури\" [Електронний ресурс] – Режим доступу до ресурсу: https://www.coursera.org/learn/system-administration-it-infrastructure-services-ua.

2. Системний адміністратор: обов’язки, ролі, плюси та мінуси професії [Електронний ресурс] – Режим доступу до ресурсу: https://blog.iteducenter.ua/sysadministration/system_administrator/.

3. Calvert S. Fedora Linux Servers with Systemd / Sheldon Calvert., 2017.

4. Що таке Linux і для чого він потрібен? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.linuxadictos.com/uk/que-es-linux.html.

5. В чому переваги Linux та як почати працювати з цією ОС. Поради початківцям [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://dou.ua/forums/topic/41333/.

6. LINUX — ПЕРЕВАГИ ТА НЕДОЛІКИ ОПЕРАЦІЙНОЇ СИСТЕМИ [Електронний ресурс] – Режим доступу до ресурсу: https://nspace.ua/info/linux-perevagi-ta-nedoliki-operatsijnoyi-sistemi/.

7. System administrator (sysadmin) [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.techtarget.com/searchnetworking/definition/system-administrator.

8. System Administrator Roles and Responsibilities | Skills [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.simplilearn.com/systems-administrator-article.

9. Vagrant [Електронний ресурс] – Режим доступу до ресурсу: https://developer.hashicorp.com/vagrant/tutorials.

10. A Guide to Vagrant [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/ops/vagrant-guide.

11. How to Create and Manage Virtual Machines with the Vagrant Command Line Tool [Електронний ресурс]. – 2023. – Режим доступу до

ресурсу: https://www.freecodecamp.org/news/create-and-manage-virtual-machines-with-vagrant/.

12. Хостинг Україна [Електронний ресурс] – Режим доступу до ресурсу: https://www.ukraine.com.ua/uk/wiki/account/.

13. Debian - Універсальна операційна система [Електронний ресурс] – Режим доступу до ресурсу: https://www.debian.org/index.uk.html.

14. JavaFX [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm.

15. Create a new JavaFX projec [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/help/idea/javafx.html.

16. virtualbox.org [Електронний ресурс] – Режим доступу до ресурсу: https://www.virtualbox.org/manual/ch01.html.

17. How to Use VirtualBox: Quick Overview [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.nakivo.com/blog/use-virtualbox-quick-overview/.

18. Virtual Box Tutorial For Beginners: Step By Step Guides [Електронний ресурс] // https://devopscube.com/. – 2022. – Режим доступу до ресурсу: https://devopscube.com/virtual-box-tutorial/.

19. Oracle VM VirtualBox. Programming Guide and Reference [Електронний ресурс] – Режим доступу до ресурсу: https://download.virtualbox.org/virtualbox/SDKRef.pdf.

20. Step-by-Step Guide to Installing Oracle VirtualBox: A Beginner's Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@tony.aloysius.77/step-by-step-guide-to-installing-oracle-virtualbox-a-beginners-tutorial-eba18cfe6d2d.

ДОДАТОК А

Лістинг коду класу приєднання до конфігураційного файлу:

```
package com.example.demo1.test;
import com.jcraft.jsch.ChannelExec;
import com.jcraft.jsch.JSch;
import com.jcraft.jsch.Session;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.concurrent.Executors;
import java.util.Properties;

public class VagrantManager {
    private static final String LOCAL_VAGRANT_PATH =
"C:\\Users\\vlasenko\\IdeaProjects\\vagrant";

    public void executeStart(boolean isLocal) {
        if (isLocal) {
            executeLocalCommand("vagrant up");
        } else {
            executeRemoteCommand("vagrant up");
        }
    }

    public void executeStop(boolean isLocal) {
        if (isLocal) {
            executeLocalCommand("vagrant halt");
        } else {
            executeRemoteCommand("vagrant halt");
        }
    }

    public void executeStatus(boolean isLocal) {
        if (isLocal) {
            executeLocalCommand("vagrant status");
        } else {
            executeRemoteCommand("vagrant status");
        }
    }

    private void executeLocalCommand(String command) {
        try {
            ProcessBuilder builder = new
ProcessBuilder("cmd.exe", "/c", command);
            builder.directory(new
java.io.File(LOCAL_VAGRANT_PATH));
            Process process = builder.start();

            Executors.newSingleThreadExecutor().submit(() -> {
                try (BufferedReader reader = new BufferedReader(
```

```
                        new
InputStreamReader(process.getInputStream()))) {
                    reader.lines().forEach(System.out::println);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            });

            int exitCode = process.waitFor();
            if (exitCode != 0) {
                System.out.println("Error executing local
Vagrant command");
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    private static final String CONFIG_FILE_PATH =
"config.properties";
    public Properties loadConfigFromFile(String filePath) {
        Properties properties = new Properties();
        try (FileInputStream fileInputStream = new
FileInputStream(filePath)) {
            properties.load(fileInputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return properties;
    }
    private void executeRemoteCommand(String command) {
        Properties properties =
loadConfigFromFile(CONFIG_FILE_PATH);
        if (properties == null) {
            System.err.println("Error loading configuration from
file.");
            return;
        }

        String user = properties.getProperty("ssh.user");
        String password =
properties.getProperty("ssh.password");
        String host = properties.getProperty("ssh.host");
        int port =
Integer.parseInt(properties.getProperty("ssh.port", "22"));

        try {
            JSch jsch = new JSch();
            Session session = jsch.getSession(user, host, port);
            session.setPassword(password);
            session.setConfig("StrictHostKeyChecking", "no");
            session.connect();

            ChannelExec channel = (ChannelExec)
```

```
session.openChannel("exec");
            channel.setCommand(command);
            channel.connect();

            try (BufferedReader reader = new BufferedReader(
                    new
InputStreamReader(channel.getInputStream()))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    System.out.println(line);
                }
            }

            int exitStatus = channel.getExitStatus();
            channel.disconnect();
            session.disconnect();

            if (exitStatus < 0) {
                System.out.println("Done, but exit status not
set!");
            } else if (exitStatus > 0) {
                System.out.println("Done, but with error!");
            } else {
                System.out.println("Done!");
            }

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
package com.example.demo1.test;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class VagrantManagerUI extends Application {

    @Override
    public void start(Stage primaryStage) {
        showWelcomeWindow(primaryStage);
    }

    public void showWelcomeWindow(Stage stage) {
        stage.setTitle("Vagrant Cluster Deployment - Welcome");
```

```
        Label welcomeText = new Label("Розгортання кластера з
використанням Vagrant");
        welcomeText.setFont(new Font(24));

        Button successButton = new Button("Налаштування успішно
виконані");
        Button retryButton = new Button("Повторити");

        successButton.setOnAction(e -> {
            System.out.println("Налаштування успішно виконані");
            showConfigurationWindow(stage);
        });

        retryButton.setOnAction(e -> {
            System.out.println("Повторення налаштувань...");
        });

        VBox layout = new VBox(20);
        layout.setAlignment(Pos.CENTER);
        layout.getChildren().addAll(welcomeText, successButton,
retryButton);

        successButton.setFont(new Font(16));
        retryButton.setFont(new Font(16));

        Scene scene = new Scene(layout, 400, 200);
        setNewScene(stage, scene);
    }

    public void showConfigurationWindow(Stage stage) {
        stage.setTitle("Vagrant Cluster Deployment -
Configuration");

        VBox vbox = new VBox(10);
        vbox.setPadding(new Insets(10));


        Label label = new Label("Додавання користувачів на
віртуальну машину");
        label.setFont(new Font(16));


        TextField usersCountField = new TextField();
        usersCountField.setPromptText("Кількість користувачів");
        usersCountField.setPrefWidth(200);


        ComboBox<String> vmComboBox = new ComboBox<>();
        vmComboBox.getItems().addAll("Node1", "Node2", "Node3",
"Node4"); //
        vmComboBox.setPromptText("Виберіть VM");
        vmComboBox.setPrefWidth(200);
```

```
        // Створення CheckBox-ів для ПЗ
        CheckBox loadBalancerCheckBox = new CheckBox("Load
Balancer");
        CheckBox webServerCheckBox = new CheckBox("Web Server");
        CheckBox databaseServerCheckBox = new CheckBox("Database
Server");

        // Створення ComboBox для вибору номеру вузла для ПЗ
        ComboBox<String> loadBalancerNodeComboBox = new
ComboBox<>();
        loadBalancerNodeComboBox.getItems().addAll("Node1",
"Node2", "Node3", "Node4");
        loadBalancerNodeComboBox.setPromptText("Виберіть
вузол");
        ComboBox<String> webServerNodeComboBox = new
ComboBox<>();
        webServerNodeComboBox.getItems().addAll("Node1",
"Node2", "Node3", "Node4");
        webServerNodeComboBox.setPromptText("Виберіть вузол");
        ComboBox<String> databaseServerNodeComboBox = new
ComboBox<>();
        databaseServerNodeComboBox.getItems().addAll("Node1",
"Node2", "Node3", "Node4");
        databaseServerNodeComboBox.setPromptText("Виберіть
вузол");


        Button acceptButton = new Button("Прийняти");
        Button rejectButton = new Button("Відхилити");
        Button backButton = new Button("Повернутись на головний
екран");
        // Збільшення розміру шрифту для кнопок
        acceptButton.setFont(new Font(14));
        rejectButton.setFont(new Font(14));
        backButton.setFont(new Font(14));

        // Додавання обробників подій для кнопок
        acceptButton.setOnAction(e -> {
            try {
                int usersCount =
Integer.parseInt(usersCountField.getText());...
            } catch (NumberFormatException ex) {
                System.out.println("Будь ласка, введіть валідне
число у поле кількості користувачів.");
            }
        });

        rejectButton.setOnAction(e -> {
            // Відхиліть установку ПЗ
        });

        backButton.setOnAction(e -> {
            showWelcomeWindow(stage); // Повернутись до
```

```
вітального вікна
        });

        // Додавання елементів до VBox
        vbox.getChildren().addAll(label, usersCountField,
vmComboBox,
                loadBalancerCheckBox, loadBalancerNodeComboBox,
                webServerCheckBox, webServerNodeComboBox,
                databaseServerCheckBox,
databaseServerNodeComboBox,
                acceptButton, rejectButton, backButton);

        // Створення Scene та встановлення її в primaryStage
        Scene scene = new Scene(vbox, 400, 500); // Збільшення
висоти вікна
        setNewScene(stage, scene);
    }

    public void showDeploymentWindow(Stage stage) {
        stage.setTitle("Vagrant Cluster Deployment -
Deployment");

        Label deploymentText = new Label("Розгортання кластера з
використанням Vagrant");
        deploymentText.setFont(new Font(24));

        Button finishButton = new Button("Завершити
розгортання");
        finishButton.setFont(new Font(16));
        finishButton.setOnAction(e -> stage.close());

        VBox layout = new VBox(20);
        layout.setAlignment(Pos.CENTER);
        layout.getChildren().addAll(deploymentText,
finishButton);

        Scene scene = new Scene(layout, 400, 200);
        setNewScene(stage, scene);
    }

    private void setNewScene(Stage stage, Scene newScene) {
        stage.setScene(newScene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```