Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий

інститут електроенергетики
(інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра

студента Залізняка Максима Григоровича
(ПІБ)
академічної групи 123м-22-1
(шифр)
спеціальності «Комп'ютерна інженерія»
(код і назва спеціальності)

за освітньо-професійною програмою «Комп'ютерна інженерія»
(офіційна назва)

на тему «Обґрунтування структури комп'ютерної системи ігрової студії «Ubisoft Ukraine (Kyiv)» із використанням технологій контейнеризації»
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Цвіркун Л.І.			
розділів:				
теоретична розділ	проф. Цвіркун Л.І.			
синтез системи	Доц. Бешта. Д. О.			
розроблення програмного забезпечення	Ас. Панферова Я. В.			
Рецензент				
Нормоконтролер	Доц. Шедловська Я. І.			

Дніпро
2023

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)
_____ Гнатушенко В.В.
(підпис) (прізвище, ініціали)

«____» _____ 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня магістра

студента Залізняк М. Г. **академічної групи** 123м-22-1
(прізвище та ініціали) (шифр)

спеціальності 123 «Комп'ютерна інженерія»

за освітньо-професійною програмою 123 «Комп'ютерна інженерія»
(офіційна назва)

на тему «Обґрунтування структури комп'ютерної системи ігрової студії «Ubisoft з Ukraine (Kyiv)» із використанням технологій контейнеризації»
затверджену наказом ректора НТУ «Дніпровська політехніка» від 09.10.2023р. № 1227

Розділ	Зміст	Термін виконання
Стан питання та постановка завдання	На основі матеріалів виробничих практик, інших науково технічних джерел сформулювати наукове завдання, конкретизувати предмет та мету досліджень	10.10.2023
Теоретичний	Обґрунтувати теоретичну базу розв'язання наукового завдання, якому присвячено роботу	20.10.2023
Синтез системи	Розробка системи контейнеризації с елементами часткової автоматизації розгортання	25.11.2023
Розробка програмного забезпечення	Розробка методу реалізації контейнерних технологій з впровадженням автоматизації	26.11.2023
Експериментальний розділ	Проведення і обробка результатів експериментів	29.11.2023

Завдання видано _____ проф. Цвіркун Л.І.
(підпис керівника) (прізвище, ініціали)

Дата видачі 06 вересня 2023 р.

Дата подання до екзаменаційної комісії 10.12.2023 р.

Прийнято до виконання _____ Залізняк М. Г.
(підпис керівника) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 88 с., 28 рис., 4 табл., 1 дод., 13 джерел.

КОНТЕЙНЕРИЗАЦІЯ, КОМПЮ'ТЕРНА СИСТЕМА, РОЗГОРТАННЯ ВІРТУАЛЬНИХ СЕРЕДОВИЩ, АВТОМАТИЗАЦІЯ, ВІРТУАЛІЗАЦІЯ

Об'єкт дослідження: методи впровадження та використання технологій контейнеризації в комп'ютерній системі ігрової студії "Ubisoft Ukraine (Kyiv)".

Мета роботи: виявити ефективність використання методів та технологій контейнеризації та виділити найоптимальнішу технологію за її застосування у ігрових студіях з можливістю автоматичного розширення методом використанням автоматичних інструкцій.

Методи дослідження та реалізації – дослідження було виконано методом збору та порівняння фактичних даних одержаних в ході дослідження та висунутого припущення яке було підтверджено. Реалізація була виконано на ОС Ubuntu Linux яка повторює структуру сервера та було реалізовано за допомогою Docker.

У першому розділі роботи було проведено постановку та формулювання задачі для подальшого дослідження та обрання критерій та методів які повинні бути реалізовані у досліджені та реалізації даної роботи.

У другому розділі було проведено огляд наявних програмних продуктів та технологій, розглянуті методи їх роботи та впровадження для подальшого дослідження та як головний продукт дослідження і впровадження Docker.

У третьому розділі було проведено розробку та реалізацію даної системи розгортання контейнерних середовищ Docker, було реалізовано часткову автоматизацію.

У четвертому розділі було проведено розробку та опис програмних інструкцій с частковою автоматизацією процесу розгортки середовища.

У п'ятому розділі було проведено зібрання та дослідження даних та в подальшому підтвердження гіпотези у ефективності роботи головного продукту Docker та виявлення його ефективності над конкурентом.

ЗМІСТ

Перелік умовних позначень, символів, скорочень та термінів 6
Вступ 7
1 Стан питання та постановка завдань дослідження 9
1.1 Постановка завдання 9
1.2 Критичний аналіз і класифікація напрямків досліджень у сфері автоматизації комп’ютерної мережі 11
1.3 Ідея щодо доцільного впровадження автоматизації в ігровій студії «ubisoft ukraine (kyiv)» 13
1.4 Розповсюдженість впровадження та потреби автоматизації в комп’ютерні мережі 14
2 Теоретичний розділ 16
2.1 Загальна характеристика об'єкта дослідження 16
2.2 Розробка моделей процесів ігрової судії «ubisoft ukraine (kyiv)» 18
2.3 Загальна характеристика технології впровадження 21
2.4 Обґрунтування і вибір методів дослідження відповідно до поставленої мети 23
2.5 Аналіз та порівняння обраних технологій 25
2.6 Функціональна класифікація автоматизації комп'ютерної мережі 27
2.7 Контейнеризація як частина нових технологій розгортання додатків у віртуальних середовищах 29
3 Синтез системи контролю контейнерних середовищ 31
3.1 Вибір і обґрунтування принципів побудови функціональної схеми проектованого об’єкта 31
3.2 Обґрунтування прийнятих способів проектування і дослідження 33
3.3 Проектування системи розгортання при використані технологій контейнеризації та автоматизації 38
3.3.1 Розробка функціональних схем вузлів проектованої системи 52
3.3.2 Вибір апаратних засобів і елементної бази 57
3.3.3 Розроблення автоматизованих інструкцій для забезпечення контейнеризації 59
3.4 Висновки по розділу синтезу системи 62
4 Розроблення програмного забезпечення для розгортання контейнерного середовищ 63
4.1 Призначення й сфера застосування програми 63
4.2 Обґрунтування технічних характеристик 63
4.2.1 Постановка завдання на розробку програми 63
4.2.2 Опис алгоритму і функціонування програми 63

4.2.3 Опис і обґрунтування вибору методу організації вхідних і вихідних даних 64
4.3 Опис розробленої програми 65
4.3.1 Загальні відомості 65
4.3.2 Функціональне призначення 65
4.3.3 Опис логічної структури 65
4.3.4 Використовувані технічні засоби 67
4.4 Загальні відомості 67
5 Експериментальний розділ 69
5.1 Сутність експерименту (мета, умови) 69
5.2 Результат експерименту 70
5.3 Аналіз відповідності теоретичних та експериментальних досліджень 75
5.4 Характеристика новизни результатів 76
Висновки 78
Перелік посилань 80
Додаток А. Програмне забезпечення налаштування системи розгортання контейнерних середовищ 82

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

Ав – Автоматизація

Ка – Код автоматизації

КПР – Керований продукт розгортання

API – Application programming interface

CI/CD – Continuous Integration/Continuous Delivery

DevNet – Developer Network

DevOps – Developer Operations

LXC – Linux Containers

ВСТУП

У сучасних мережевих технологій досить значущу роль починають займати та використовуватись впровадження контейнерних середовищ з частковим впровадженням автоматизації. Так як у великих компаніях переведення у стан непрацездатності коштує великих грошей які недоотримає компанія. Тож впроваджування контейнерних автоматизованих систем які допоможуть зберігати критичні інтелектуальні даті та системи які допоможуть швидко розгортати нові контейнери с застосунками або тестові комплексні системи повинні позитивним чином вплинути на якість та ефективність роботи працівників сектору технічної підтримки, і розробників так і кінцевих користувачів.

Мета і завдання дослідження. *Метою роботи* є обґрунтування структури комп'ютерної системи ігрової студії "Ubisoft Ukraine (Kyiv)" із використанням технологій контейнеризації.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– дослідити які технології використовуються зважаючи на об'єм потреб контейнеризації відповідно до розгорнутих продуктів;

– розглянути методи контейнеризації та технології автоматизації у сумісному використані та виділити найбільш адаптивну та гнучку технологію;

– розробити та розгорнути метод контейнеризації с впровадженням автоматичного коду розгортання кінцевого тестового застосунку;

– провести аналіз с подібними стилетами та на основі проведеного дослідження виділити недоліки та переваги впроваджуваного рішення;

Об'єкт дослідження – процес розробки та розгортання і розміщення контейнерних технологій у поєднані с автоматизованими інструкціями динамічної розгортки кінцевих продуктів с доступом віддалених користувачів.

Предмет дослідження – моделі та принципи контейнерних технологій та методи їх роботи с використанням автоматизації.

Методи дослідження. Для виконання висунутих вимог дослідження було проведено практичне моделювання с одержанням даних для подальшого обрахунку формулами ефективності роботи віртуальних та контейнерних середовищ.

Наукові положення:

1. Встановлено, що при використані контейнера Docker та зібраних матеріалів і математично виведених результатів, що система контейнерного розгортання на 15 – 17% ефективніше використовує ресурси та має менш затримки на виконання у поєднані с автоматизованими інструкціями у випадку створення дублікатів контейнерів ніж у конкурента Vagrant.

Наукові результати:

1. Обґрунтування використання та впровадження системи на основі контейнерних технологій таких як Docker для забезпечення швидкого розгортання на одному фізичному сервері багатьох контейнерів с подальшим встановленням та налаштуванням і оновленням розгорнутих контейнерних додатків с можливістю динамічного масштабування.

2. Запропоновано метод контейнерного розгортання с впровадженням автоматизації для зменшення впливу людини на кінцевий контейнерний застосунок та гнучкість подальшого оновлення та модифікації даного контейнера під подальших потреб розробників.

Обґрунтованість і достовірність наукових положень, висновків і рекомендацій підтверджуються за допомогою використання таких методів у даній роботі як: метод дослідження системи масового обслуговування, також використано для проведення формування теорії метод очікування ймовірностей, для дослідження динаміки роботи було використано метод часових рядів.

Практичне значення отриманих результатів полягає в даній реалізації контейнерних середовищ у поєднані с автоматизацією було виявлено які навантаження та здатність бути гнучким та модульним у використані у подібних проектах та вимоги до обладнання.

1 СТАН ПИТАННЯ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Постановка завдання

Підвищення продуктивності та гнучкості розробки та доставки програм для кінцевих користувачів, автоматизація допомагає збільшити швидкість та гнучкість процесів розробки та впровадження програмного забезпечення, спрощуючи і прискорюючи їх. Видалення вузьких місць та зменшення ручної праці, автоматизація дозволяє позбавитися від обмежень та зменшити обсяг ручних операцій, що сприяє зменшенню помилок та підвищенню ефективності. Стандартизація процесів разом з автоматизацією проходить стандартизацію процесів, що додатково знижує ймовірність помилок або недоліків, характерних для ручних підходів до вирішення завдань.

Однак незважаючи плюси які внесе впровадження автоматизації все одно пов’язана с кореляцією зростання віддалено працюючих людей та витрати впровадження мережевої інфраструктури та програмно апаратного комплексу спрямованого на забезпечення діяльності співробітників на віддаленій чи на очній формі праці. Одна як свідчить статистика у 2022 році світові витрати на корпоративне програмне забезпечення склали близько 783 мільярди доларів США, що відзначається зростанням на 7,1% порівняно з попереднім роком. Схоже на тенденцію до високих темпів росту, яка спостерігається у майже всіх підсегментах індустрії IT-послуг. Варто зазначити, що доходи ринку корпоративного програмного забезпечення зросли більш ніж удвічі протягом останнього десятиліття, з 2010 по 2020 рік. З щорічним зростанням, яке часто перевищує 10%, ринок корпоративного програмного забезпечення стає найшвидше зростаючим сегментом в галузі IT. Корпоративне програмне забезпечення спрямоване на задоволення потреб організацій, часто спрямовуючи свої зусилля на підвищення ефективності основних бізнес-процесів. Багато підсегментів корпоративного програмного забезпечення, такі як програмне забезпечення для управління бізнес-процесами (BPM), програмне забезпечення для планування ресурсів

підприємства (ERP) і програмне забезпечення для управління взаємовідносинами з клієнтами (CRM), стали значними ринками в самому собі. Програмне забезпечення CRM спрямоване на аналіз та поліпшення взаємодії з клієнтами, як з поточними, так і з потенційними клієнтами, і очікується, що воно принесе більше ніж 47 мільярдів доларів обсягу продажів у 2022 році. Програмне забезпечення ERP більше спрямоване на збір і інтерпретацію корпоративних даних і, за прогнозами, забезпечить ще 95 мільярдів доларів загального доходу відповідно відображено на рисунку 1.1 [8].

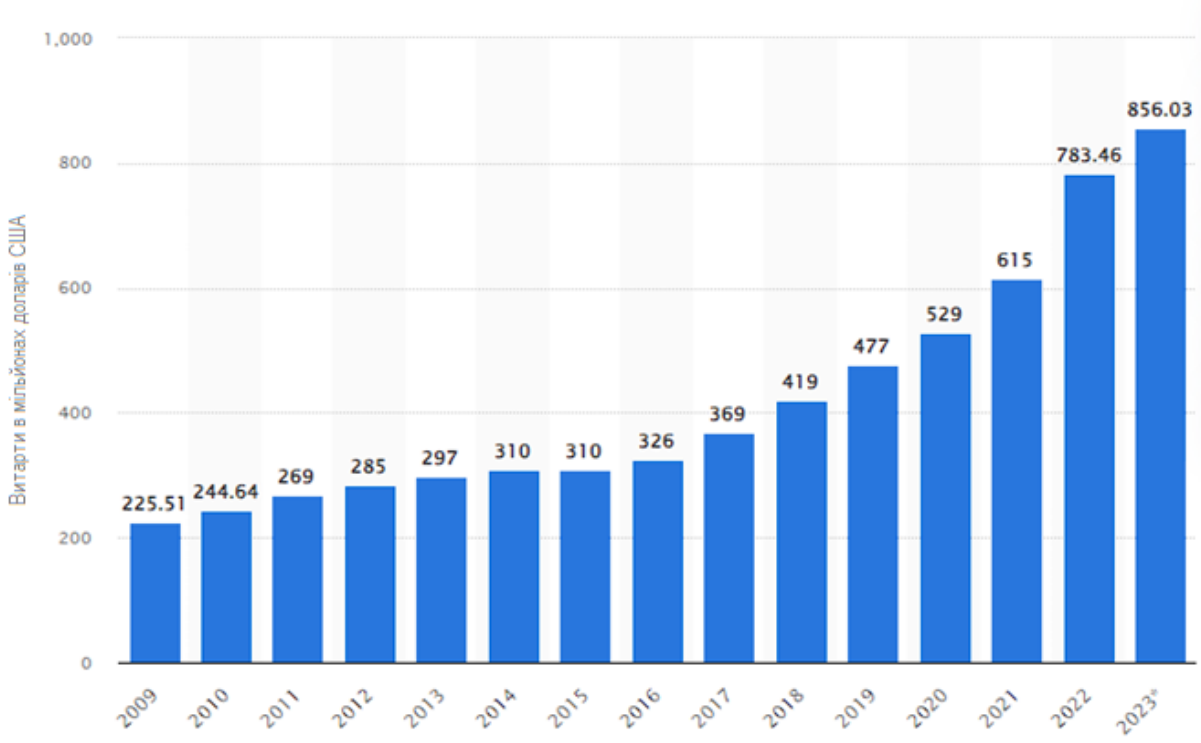


Рисунок 1.1 – Графік корекція витрат компаній на забезпечення апаратно програмного комплексу мереж по роках

Тож згідно статистики так як впровадження таких методика та технологій які пов’язані с автоматизацією мереж та мережевих технологій позитивним чином впливають на всі аспекти впровадження починаючи зі зменшенням виникнення помилок у разі розгортання додаткових частин мережі чи додаткових ресурсів закінчуючи покращенням систем реагування

на загрози яка реагує згідно її налаштованих протоколів, звісно витрати на такі технології дуже великі тому компанії в малих та середніх розмірів скептично, а то і негативно відносяться до таких технологій вважаючи, що вони непотрібні чи не будуть в повній мірі використовуватись у їх підприємствах, однак якщо підходити з більш детальною проробкою вимог до кінцевої мережі і точково обирати критичні частини в яких таке впровадження буде і доцільне і не потребуватиме багато вкладень, що в подальшому буде виражено у меншому часі непрацездатності мережі її захищеності та цілісності інтелектуальної власності компанії та клієнтів [9].

1.2 Критичний аналіз і класифікація напрямків досліджень у сфері автоматизації комп'ютерної мережі

Так як це нова сфера діяльності яка функціонує вже значну кількість часу але поширення та популярність впровадження автоматизації комп'ютерних систем набула у значній мірі в великих компаніях таких як Amazon, Google, Apple.

В цій роботі було проведено аналіз та оцінка впливу та кінцевої доцільності впровадження автоматизації на менш малій компанії на прикладі ігрової студії «Ubisoft Ukraine (Kyiv)». Якщо оцінювати дане підприємство у порівняні с раніше перерахованими компаніями то вона здається середньою за кількість користувачів які використовують мережеві ресурси в даній організації, але опираючись на довідкові дані кількості користувачів за останні декілька років то це може значити що доцільність автоматизації комп'ютерної мережі було виправданою та відобразиться на деяких рівнях адміністрування, тестування, забезпечення безпеки та розгортання додаткових ресурсів, систем та контейнерів у разі потреби при підвищені попиту на нові продукти серед кінцевих користувачів [3].

Тобто для вирішення питання впровадження автоматизації буде потрібно провести комплексний підхід, так як кінцеве рішення буде впроваджуватись на таких рівнях використання мережі:

– адміністрування мережі та вирішення буденних задач налаштувань та виправлень;

– безпекові провадження за допомогою систем автоматичного моніторингу та швидкого реагування на непередбачувані вторгнення;

– використання методів автоматизованого тестування на створюваних продуктах для більш якісного виявлення несправностей та багів;

– впровадження методів контейнеризації та автоматичного виділення ресурсів під випуск нових продуктів та застосунків до яких будуть мати кінцеві користувачі з зовні.

Виходячи с виявлених критичних рівнів які на цьому етапі роботи були класифіковані, такі критичні точки мережі при яких мережа буде втрачати свою повну або часткову функціональну спроможність та доступність співробітників та кінцевих користувачів [5].

Потрібно провести та виділити найбільш доцільні технології відповідно даного критерію функціональної оцінки для впроваджувальної мережі а саме:

– мова конфігурації;

– агент або безагентний підхід;

– спрямованість;

– спільнота та підтримка;

– особливості.

Так як така класифікація порівняння враховує вже наявні апаратно – технічну базу на основі якої може буде виконано впровадження при характерній доцільності аналізованих технологій. Також може буде враховано потреба в до навчанні або принаймні додаткових спеціалістів супроводу та обслуговування кінцевого рішення для даної організації.

1.3 Ідея щодо доцільного впровадження автоматизації в ігровій студії «Ubisoft Ukraine (Kyiv)»

Зважаючи на те, що у попередньому пункті цієї роботи було описано рівні на яких потребує розгляду методів та технологій які окремо використовуються на вище вказаних рівнях мережі.

Тож зважаючи на поставлені задачі щодо автоматизації які потребує підприємство ігрова студія «Ubisoft Ukraine (Kyiv)», а саме такі вимоги як:

– автоматизація, що буде використовуватись для збереження та резервування інтелектуальних продуктів та особливих критичних даних якими оперує організація;

– впровадження контейнеризації у поєднані з автоматизацією для підвищення інтеграції нових продуктів та розгортання і налаштування їх у реальних умовах для використання користувачами.

Для більш доцільного використання ресурсів та досвіду сектору технічного обслуговування у розділі який присвячено теоретичному огляду було розглянуто технологія та обрано найбільш доцільні з них у порівняні з їх конкурентами виходячи з висунутих критеріїв до автоматизованого середовища.

Було виділено деякі технології як Docker – це платформа для контейнеризації додатків. Вона дозволяє ізолювати додатки та їхні залежності для швидкого та надійного розгортання. Kubernetes – це оркестратор контейнерів, який дозволяє автоматизувати управління контейнерами, масштабування та розподілення додатків. Jenkins – це інструмент для автоматизації процесів Continuous Integration і Continuous Delivery. Він дозволяє створювати конвеєри для автоматичної збірки та тестування коду. Так як в цій роботі буде проведено розробку та впровадження технологій контейнеризації то без застосування коду автоматизації для покрашення роботи моделей та демонстрації того що в робочому процесі ці технології виконуються паралельно с десятками, а то із сотнями продуктів одночасно в залежності до їх поставлених задач. І кінцеву реалізацію було виконано у

вигляді демонстрації створення та розгортання тестового додатка який було реалізовано у вигляді веб-ресурсу, який буде нести функції відображення повідомлення про наявність доступу без специфіки реальної роботи, як кінцевий продукт. Для виконання поставленої вимоги реалізації цих технологій буде проведено деякі пункти а саме:

– проведення аналізу доцільності технології;

– проведення створення коду автоматизації для паралельного розгортання декількох додатків;

– виконання розгортання методів контейнеризації на демонстраційному веб-застосунку;

– дослідження впливу доцільності технологій тестування у поєднані з технологіями контейнеризації.

Для більш доцільного використання ресурсів та дослідження методів та технічних аспектів реалізації їх у стані підготовки та запуску у розділі який присвячено теоретичному огляду та синтезу системи було розглянуто технології та обрано найбільш доцільні з них у порівняні з їх конкурентами. Виходячи з висунутих критеріїв до автоматизованої середовища розгортання контейнерних технологій, а також у одержані підтвердження у вигляді таблиць та розрахунків котрі підкріпляють висунуте припущення, щодо вибору продукту розгортання контейнерів.

1.4 Розповсюдженість впровадження та потреби автоматизації в комп’ютерні мережі

Потрібно провести дослідження та виявити найбільш доцільні технології та вживані технології для впровадження у комп’ютерну мережу ігрової студії «Ubisoft Ukraine (Kyiv)». Виходячи зі специфіки роботи підприємства у штатному складі яке має 600 чоловік робочого персоналу та задач розробки, випуску та при потребі розгортання додаткових апаратно–програмних систем, для розширення інфраструктури, для задоволення потреб кінцевих користувачів і створення системи багато

користувацького доступу. Також дуже важливо, щоб обраний комплекс впровадження контейнерного середовища виконував такі критичні функції як:

– розгортання та збірку ізольованих контейнерів;

– можливість мобільного або часткового автоматичного розгортання клонів контейнерів;

– забезпечення легкого корегування та оновлень у контентах;

– забезпечення доступу до керування з внутрішньої консолі окремим контейнером.

Виходячи з того, що компанія має 3,768 мільярдів в рік і дохідність росте, та щорічний приріст клієнтів який складає приблизно від 3 до 4,5 мільйонів клієнтів, то для їх потреб впровадження технологій які використовують в DevNet є дорогою необхідністю, але в іншому випадку потрібно чітко проаналізувати та обрати для кожного критичного рівня свої технології, які будуть найбільш функціональними та гнучкими. Технології контейнеризації є дуже вживаними у компаніях із таким робочим процесом та створюваними продуктами, які потребують впровадження віртуалізації та автоматизації при використанні розгортання контейнерів, додатків при фінальних випробуваннях та релізах розробок. Тож для такої системи впровадження потрібно провести аналіз різниці локальної реалізації та при використані хмарових технологій.

2 ТЕОРЕТИЧНИЙ РОЗДІЛ

2.1 Загальна характеристика об'єкта дослідження

Ігрова студія студія «Ubisoft Ukraine (Kyiv)» має у своєму керуванні два офіси перший котрий розташовується у Києві та другий у Одесі. Загалом мережа підтримує працездатність 683 працівників, та декілька загальних ресурсів на серверному обладнані. Також дана компанія займається своїм інтернет магазином та виготовленням і публікуванням ігрових продуктів під своєю франшизою.

Графічне зображення організаційної структури ігрової студії "Ubisoft Ukraine (Kyiv)" можна описати так що ця студія використовує лінійно-функціональний тип організаційної структури. Це означає, що вона має ієрархічну організацію з чітким поділом зон відповідальності і єдиним керівником що відображено на рисунку 2.1.

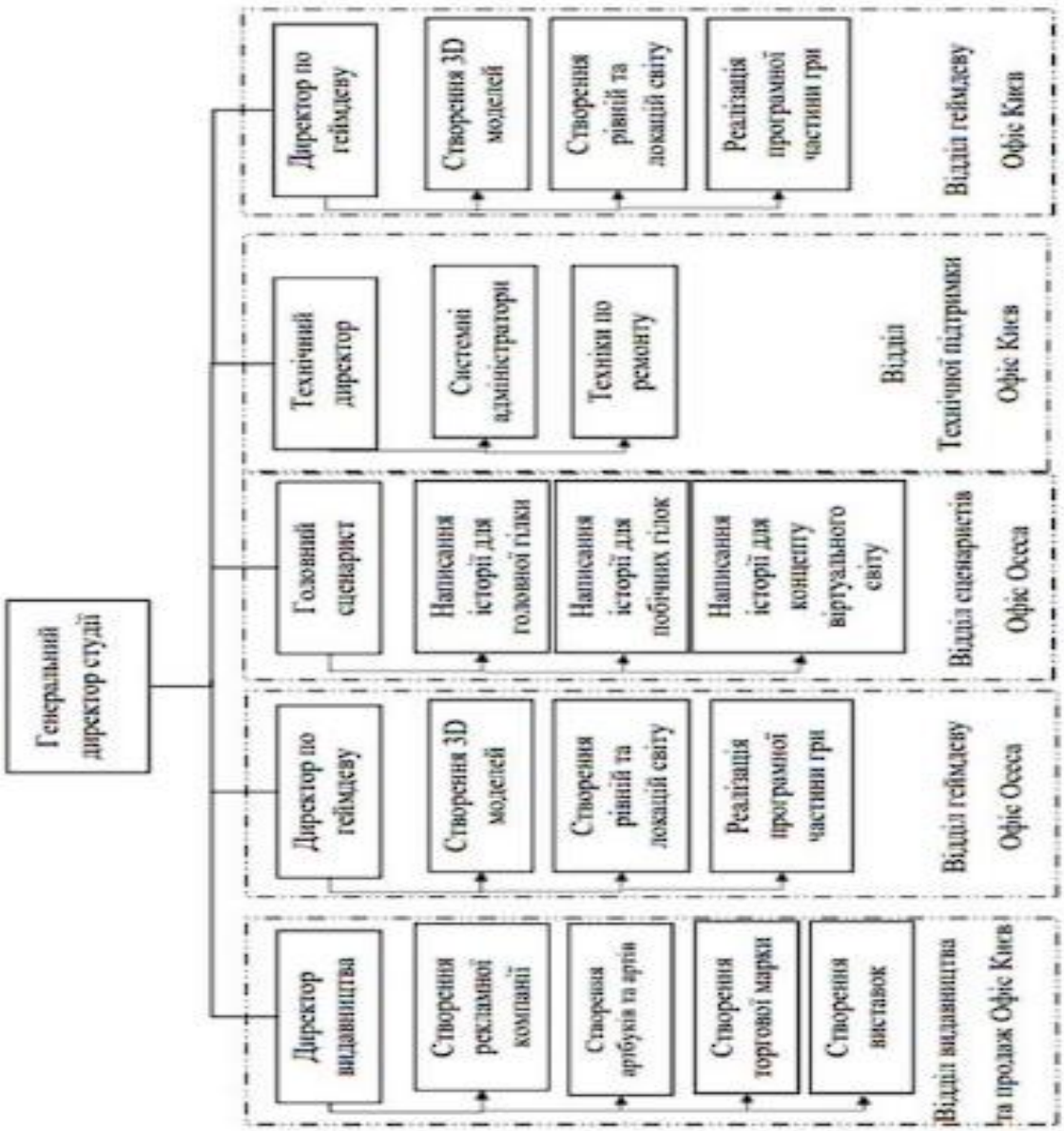


Рисунок 2.1 – Організаційна структура ігрової студії «Ubisoft Ukraine (Kyiv)»

Дана розглядувана мережа ігрової студії має загалом п’ять підмереж. Зв’язок між ними виконується за використанням пограничного маршрутизатора та мережі інтернет. Відповідно мережа ігрової студії представлена у вигляді загальної архітектури. Спираючись на неї було обрано сервер впровадження контейнеризації у найоптимальнішій частині мережі ігрової студії, так як таке впровадження є найбільш ефективнішим для зовнішнього користувача та затримки доступу, що відображена на рисунку 2.2.

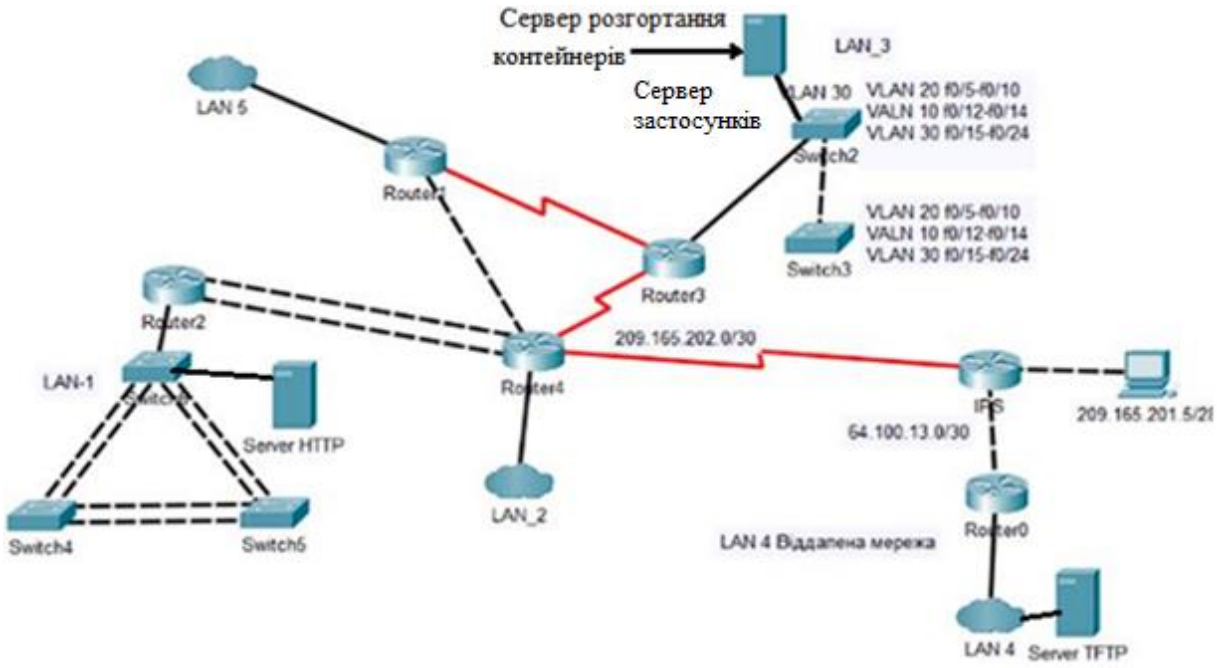


Рисунок 2.2 – Загальна архітектура мережі ігрової студії «Ubisoft Ukraine (Kyiv)»

Відповідно до завдання, спираючись на архітектуру мережі ігрової студії у якій є сервер розгортання контейнерів і потреб впровадження часткової автоматизації по розгортанню контейнерного середовища і застосунків розгортання у автономному режимі роботи, для підвищення ефективності роботи застосунків. Додатково розглянуто можливість вирішення питання автоматизованого налаштування заздалегідь створеними програмами з налаштуваннями контейнерного середовища і динамічного розширення для заощадження часу простою у разі виникнення непередбачуваних програмних чи апаратних поломок.

2.2 Розробка моделей процесів ігрової судії «Ubisoft Ukraine (Kyiv)»

Розробка моделей процесів, які є характерними для об'єкта ігрової студії при впровадженні автоматизації в комп'ютерну мережу, є важливим завданням для забезпечення ефективності та надійності інфраструктури студії. Ось деякі кроки, які буде оброблено для розробки даної моделі.

Визначення процесів ігрової студії «Ubisoft Ukraine (Kyiv)». Основними критичними процесами які виконуються в студії являються тестування продукції, розповсюдження і розгортання готових ігор, обробка та зберігання інтелектуальної та користувацької інформації.

Відповідно до процесів проведено їх огляд функціонування:

– тестування кінцевого виробу потрібен для виявлення недоліків та несправності котрі були невиявлені в розроблювальній оточені, та для більшої оптимізації у розповсюджених системах користувачів;

– розповсюдження і розгортання готових продуктів являє собою важливим кроком котрий пов’язаний с тестуванням та залежить від якісної відладки та виявлення несправностей і усунення їх. Мета даного процесу це розгортання обумовлених ресурсів, а саме за допомогою методів контейнеризації створення та розгортання створених продуктів для доступу до нього зовнішнім користувачам, а саме онлайн проектам з багато користувацьким прогресом. Цей процес потребує автоматизацію для більш швидкого розгортання додаткових ресурсів у разі потреби для збільшення комфортності кінцевого користувача;

– обробка та зберігання інтелектуальної та користувацької інформації так як є багато користувачів у різних проєктах то вони генерують багато прогресу для кожного облікового запису, та додатково ще є внутрішні розробки котрі потрібно резервувати, так своєчасно зберігати тож такі дані, котрі коштують багато грошей потребує впровадження механізмів автоматизованого збереження та резервування.

Визначення точок автоматизації та аналіз поточних процесів включає такі [6]:

– тестування кінцевого виробу, а саме оптимізувати тестування за допомогою Python Automated Test System (pyATS) таке рішення для тестування та перевірки мережних пристроїв на основі Python, спочатку розроблене Cisco для внутрішнього використання, а потім доступне для громадськості та частково з відкритим кодом. Так цій системі покладається

виявлення впроваджених змін перш ніж запустити їх у виробництво, а також продовжити перевірку та моніторинг у виробництві, щоб забезпечити безперебійну роботу;

– розповсюдження і розгортання готових продуктів. Цей процес має декілька корків це виділення продукту який потребує розгортання які тип сервісу буде обрано для нього та саме процеси пов’язані с контейнеризацією можна впровадити автоматизацію для зменшення часу розгортання та усунення помилок при розгортанні та при потреби динамічне додавання потужності. Враховуючи потреби потрібно розглянути технології такі як Docker;

– обробка та зберігання інтелектуальної та користувацької інформації можливо оптимізувати такі кроки як автоматичне керування версіями у поєднані із технологіям Git розгорнутому на локальному чи при потребі на віддаленому сервері для виконання резервуванням критичної інформації та розробок та додатково впроваджувати оновлень для подальших розробок продуктів що покращить інтеграцію та цілісність кінцевих розробок.

Розроблення моделі процесу автоматизації для виділених пунктів:

– тестування кінцевого виробу потрібно провести аналіз функціональності програми згідно розробки та розібрати її на послідовні блоки тестування та виходячи з блоків провести використання скриптів тестування з урахуванням різних частин таких як функціональний тест, тест безпеки, тест завантаження. Також згідно вимог буде використовуватись pyATS;

– розповсюдження і розгортання готових продуктів виділення та аналіз потрібних ресурсів котрі були оговорені та налаштовані на кінцевому продукті, так як це критично важлива частина для автоматизованого розгортання, після узгодження потреб продукту створити відповідно віртуальну середу розгортання та виконати налаштування головного фалу Dockerfile, так як від його заданих налаштувань буде відштовхуватись автоматизоване розгортання для пов’язаних продуктів які будуть в комплексі

з ним. Після зіставлення коду автоматизації проводиться процес розгортання контейнера докер та самого застосунку, потім проводиться тестування розгорнутого продукту на доступність аналогічне тестування як при тестуванні кінцевого виробу, щоб запевнитись у повному функціонуванні у даному середовищі загального користування;

– обробка та зберігання інтелектуальної та користувацької інформації для забезпечення цього процесу потрібно виконати декілька умов по перше усі критичні розробки та вся прогресія користувачів зберігаються на сервері на якому розгорнуто технології Git та додатково на деяких робочих станціях відділів, котрі оперують критичними даними. То для забезпечення автоматизованого копіювання створено код за допомогою Ansible налаштовується код автоматизованого копіювання даних до локального та віддаленого від потреб відділу чи користувачів копіювання даних та створення гілок контролю версій у випадку резервування розробок для подальших модифікацій.

2.3 Загальна характеристика технології впровадження

Розробка програмного забезпечення для мереж (DevNet) є новим та важливим напрямком в області мережевих технологій. DevNet об'єднує розробників програмного забезпечення та мережевих інженерів для створення програмних рішень для автоматизації, управління та оптимізації мереж. Розглядається концепція DevNet, ключові технології та перспективи її розвитку. DevNet сутність та основні концепції. DevNet – це парадигма, яка базується на ідеї розробки програмного забезпечення для мереж з використанням програмного коду, а не традиційних фізичних налаштувань. Основні концепції DevNet включають такі складові як описано нижче [7].

Автоматизація мережі DevNet дозволяє створювати код та програми для автоматизації процесів налаштування та управління мережею.

Управління програмним шляхом, вміст управління мережею стає програмним та здатним до керування за допомогою коду.

Селф–сервіс і мережева оркестрація DevNet розвиває ідею автоматизації через інтерфейси самообслуговування та мережеву оркестрацію.

Технології DevNet використовує різноманітні технології та інструменти для розробки програмного забезпечення для мереж. Серед ключових технологій можна виділити API–інтерфейси використання API для звернення до мережевих пристроїв та платформ дозволяє створювати програми для автоматизації.

Контейнеризація та оркестрація використання Docker та Kubernetes дозволяє створювати та масштабувати мережеві додатки.

Інфраструктура, як код (IaC) використання коду для опису інфраструктури дозволяє створювати мережеві конфігурації програмним способом.

Аналітика та моніторинг важливим елементом DevNet є здатність збирати та аналізувати дані з мережі для оптимізації її роботи.

Перспективи розвитку DevNet . Майбутнє DevNet обіцяє багато цікавих можливостей та викликів. Деякі з перспектив розвитку включають:

– розширення мультиклод-мереж, розробники будуть зосереджуватися на створенні додатків, які працюють в різних хмарних середовищах;

– інтеграція штучного інтелекту (ai) та машинного навчання (ml) використання ai і ml для автоматизації та аналізу мережі;

– безпека DevNet розробники будуть надавати більше уваги заходам безпеки мереж та додатків;

– розвиток спільноти DevNet зростання інтересу до DevNet призведе до розвитку активної спільноти розробників та інженерів.

DevNet представляє собою важливу еволюцію в галузі мережевих технологій. Цей підхід дозволяє розробникам створювати програмне забезпечення для мереж, що є більш гнучким, ефективним і піддається автоматизації. За перспективами розвитку DevNet можна очікувати подальшого зростання його популярності та впровадження в різних галузях [7].

2.4 Обґрунтування і вибір методів дослідження відповідно до поставленої мети

Так як насамперед це дослідження потребує літературний аналіз для того, щоб обрати та порівняти найбільш доцільні технології та методи впровадження автоматизації для більш ефективного та доцільного фінального результату. Тож нижче викладено коротка характеристика та призначення роботи розглядуваних методів для впровадження у кінцеву мережу ігрової студії.

Спочатку було розглянуто та обрані найбільш доцільні інструменти згідно потреб компанії [6].

Ansible – це інструмент автоматизації конфігурації та управління системами. Він дозволяє налаштовувати та керувати мережевими пристроями з використанням декларативних конфігураційних файлів.

Chef – інструмент для конфігурації та управління інфраструктурою. Він дозволяє автоматизувати процеси розгортання і управління мережевими ресурсами.

Puppet – це інструмент для автоматизації конфігурації та управління мережевими пристроями і серверами. Він використовує мову конфігурації Puppet DSL.

Виходячи з того, що компанія створює та розміщує свої продукти в своєму онлайн магазині тож компанія потребує використання технологій для забезпечення швидкого розгортання нових продуктів та розширення ресурсної бази у разі масштабованості проектів та продуктів. Тож нижче коротко описані вже відібрані методи та технології які використовують спеціалісти DevOps для вирішення поставлених вимог.

CI/CD (Continuous Integration/Continuous Delivery) – це практика автоматичного об'єднання коду (Continuous Integration) та автоматичної поставки програмного забезпечення в продакшн (Continuous Delivery). Вона допомагає прискорити розробку і забезпечити стабільне впровадження змін.

Docker – це платформа для контейнеризації додатків. Вона дозволяє ізолювати додатки та їхні залежності для швидкого та надійного розгортання.

Kubernetes – це оркестратор контейнерів, який дозволяє автоматизувати управління контейнерами, масштабування та розподілення додатків.

Jenkins – це інструмент для автоматизації процесів Continuous Integration і Continuous Delivery. Він дозволяє створювати конвеєри для автоматичної збірки та тестування коду.

Monitoring and Logging (Моніторинг і Журналювання) використання інструментів моніторингу та журналювання, таких як Prometheus, ELK Stack, Splunk тощо, для відстеження стану системи та аналізу подій.

Ці технології допомагають автоматизувати і поліпшити різні аспекти розробки програмного забезпечення та управління інфраструктурою, сприяючи швидкому та ефективному розгортанню та обслуговуванню систем.

Вибір Ansible та його переваги над Chef та Puppet при виборі інструменту для автоматизації конфігурації та управління серверами, розробники та системні адміністратори стикаються з важливим вибором між різними інструментами, такими як Ansible, Chef та Puppet. У цій частині розглянемо вибір Ansible та визначимо його переваги перед Chef та Puppet.

Простота використання. Ansible відомий своєю легкістю використання. Основні конфігурації та завдання в Ansible описуються у простому форматі YAML. Це робить Ansible дуже доступним для новачків та забезпечує швидкий старт. Chef та Puppet використовують більш складні мови та синтаксис, що може вимагати більше часу на навчання та розгортання.

Агент-less архітектура. Однією з ключових переваг Ansible є його агентна архітектура. Ansible не вимагає встановлення агентів на керованих серверах, щоб керувати ними. У випадку Chef і Puppet, потрібно встановлювати агенти на кожному сервері, що може бути більш складним та затратним завданням.

Швидкість виконання. Ansible відомий своєю швидкістю виконання завдань. Він працює за принципом SSH для зв'язку з серверами та виконання

команд. Це дозволяє Ansible запускати завдання швидше порівняно з Chef і Puppet, які можуть вимагати більше часу на обробку через свою архітектуру та агенти.

Підтримка для різних ОС. Ansible має велику кількість модулів та плагінів для різних операційних систем, що робить його універсальним із можливістю підтримки різних конфігурацій та середовищ. Chef та Puppet також підтримують різні ОС, але вони можуть вимагати більше зусиль для налаштування [11].

Активна спільнота та підтримка. Ansible має активну спільноту користувачів та розробників, яка регулярно вносить нові функції та виправлення помилок. Це гарантує актуальність та підтримку Ansible в майбутньому. Chef та Puppet також мають спільноти, але їхні розвиток та підтримка можуть бути менш активними.

Підсумувавши усі преваги було обрано Ansible так як – це потужний та легкий інструмент для автоматизації конфігурації та управління серверами, який пропонує багато переваг над Chef та Puppet. Його простота використання, агентна архітектура, швидкість виконання, підтримка для різних ОС та активна спільнота роблять Ansible чудовим вибором для розробників та системних адміністраторів, які шукають потужний інструмент для автоматизації [11].

2.5 Аналіз та порівняння обраних технологій

Перш за все порівняно технології, щоб зрозуміти яка апаратна та програмна база потрібна для їх впровадження та реалізацію.

Для даної роботи є декілька основних критерії по котрим проводилося порівняння. А сама:

– мова конфігурації;

– агент або безагентний підхід;

– спрямованість;

– спільнота та підтримка;

– особливості.

Мова конфігурації [11]:

– ansible використовує YAML–подібну мову для опису конфігурації. Синтаксис Ansible дещо простіший для сприйняття;

– puppet використовує мову конфігурації Puppet DSL (Puppet Domain Specific Language). Ця мова більш потужна і гнучка, але може бути складніше для новачків.

– chef використовує мову конфігурації Chef DSL, а також можливість використовувати Ruby для складних завдань. Для розробників Ruby це може бути зручним.

Агент або безагентний підхід:

– ansible безагентний. ansible взаємодіє з мережевими пристроями та серверами через ssh або winrm, що полегшує встановлення та конфігурацію;

– puppet вимагає агента puppet, який встановлюється на цільових серверах або мережевих пристроях;

– chef також вимагає агента chef, який встановлюється на цільових системах.

Спрямованість [11]:

– ansible більше акцентує робиться на автоматизації задач управління системними налаштуваннями і інфраструктурою, а не на створенні програмного коду;

– puppet і chef спрямовані на автоматизацію інфраструктури та конфігурації серверів, а також на управління програмами та пакунками.

Спільнота та підтримка:

– ansible має широку спільноту та активний розвиток. документація багатомовна і детальна;

– puppet і chef також мають активні спільноти і документацію, але можуть бути менш популярними в деяких випадках.

Особливості [11]:

– ansible повністю безагентний, має більш простий синтаксис, більш спрямований на оркестрацію задач;

– puppet має механізм відстеження стану системи та здатність до декларативного програмування;

– chef дозволяє використовувати ruby для складних завдань і має більше гнучкості в налаштуванні.

Тож виходячи з поставлених вимог та професійних навичок персоналу буде доцільніше та більш ефективно використовувати Ansible, так як даний інструмент буде і більш доцільний так і навчання персоналу потребує мінімальних вкладень і вони вже мають уявлення роботи зі схожим синтаксисом.

Ansible є більш доцільним виходячи з переднього порівняння так як не потребує додаткового встановлення та розгортання окремих програмних застосунків так і вживаність мови на якому виконується написання скрипотів є споріднені особливості з мовою Python, що у свою чергу спрощує інтеграцію у робочу конфігурацію так як деякі окремі функції автоматизації теж використовують програми написані на мові Python для отримання даних через використання API запитів на окремі ресурси в середині мережі такі як середовища моніторингу так і на зовнішні ресурси які потрібні для інтеграції в випускаємій програмні продукти чи сервіси направлені на зовнішніх користувачів [2].

2.6 Функціональна класифікація автоматизації комп'ютерної мережі

Функціональна класифікація автоматизації комп'ютерної мережі передбачає поділ автоматизаційних засобів та рішень за їхніми функціональними ролями та завданнями в мережевому середовищі. Ось декілька основних функціональних класів автоматизації в комп'ютерних мережах.

Управління конфігурацією:

– керування конфігураціями мережевих пристроїв та сервісів;

– резервне копіювання конфігурацій;

– автоматична розгортка налаштувань на нових пристроях.

Моніторинг та аналіз:

– спостереження за роботою мережі та пристроїв;

– виявлення інцидентів та аномалій;

– збір та аналіз даних про використання ресурсів.

Автоматизація процесів відновлення:

– автоматичне виявлення та реакція на збої та помилки в мережі;

– запуск процесів відновлення або переходу на резервні шляхи.

Скасування та оптимізація:

– автоматичне масштабування мережевих ресурсів за потребою;

– оптимізація роботи мережі для підвищення продуктивності.

Автоматичне управління безпекою:

– виявлення та відстеження загроз безпеці мережі;

– автоматичні заходи щодо усунення потенційних загроз.

Управління обліком та інвентаризацією:

– ведення обліку мережевих пристроїв та програмного забезпечення;

– автоматичне оновлення інвентарних списків.

Управління мережевими послугами:

– автоматизація розгортки та управління послугами, такими як VPN, мережеві файли, DNS, DHCP тощо.

Оркестрація ресурсів:

– керування розподілом ресурсів та задач між серверами та обчислювальними вузлами.

Інтеграція з хмарними службами:

– автоматизація з'єднання мережі з хмарними ресурсами та сервісами.

Ця класифікація відображає різні аспекти автоматизації комп'ютерних мереж і може включати в себе різноманітні рішення та інструменти, спрямовані на полегшення управління, забезпечення безпеки, підвищення

продуктивності та підвищення якості обслуговування мережі. Вибір підходящих засобів автоматизації залежить від конкретних вимог та цілей вашої мережі [5].

2.7 Контейнеризація як частина нових технологій розгортання додатків у віртуальних середовищах

Контейнеризація – це технологія віртуалізації, яка дозволяє упаковувати, доставляти та запускати додатки та їхні залежності разом в ізольованих контейнерах. Кожен контейнер має власну операційну систему (часто називається "легковаговою ОС") та включає всі необхідні бібліотеки та середовище виконання, що робить його переносимим та консистентним навіть на різних середовищах.

Основні методи реалізації контейнеризації включають [6]:

– docker – найпопулярніший інструмент для контейнеризації. Він надає можливість створювати, розгортати та управляти контейнерами. Docker використовує docker engine для управління контейнерами та використовує dockerfile для опису конфігурації контейнера;

– kubernetes – це оркестратор контейнерів, який дозволяє автоматизувати розгортання, керування та масштабування контейнеризованих додатків. Він забезпечує оркестрацію контейнерів на кластері серверів;

– containerd – це менеджер контейнерів, який використовується багатьма платформами та оркестраторами. Він надає базовий рівень функціональності для роботи з контейнерами;

– openshift – це розширена платформа контейнеризації на основі kubernetes, яка включає в себе додаткові функції для автоматизації розгортання та керування контейнерами;

– lxc (linux containers) – це технологія віртуалізації на рівні ОС для створення системних контейнерів. Вона використовує віртуальні простори імен і управляє ресурсами на рівні ядра linux;

– rkt (Rocket) – це інший менеджер контейнерів, який надає альтернативу Docker. Він був розроблений за ідеєю більшої простоти та безпеки.

Контейнеризація дозволяє розробникам та адміністраторам створювати і розгортати додатки більш швидко та ефективно, забезпечуючи ізоляцію між контейнерами і забезпечуючи консистентність середовища від розробки до виробництва.

3 СИНТЕЗ СИСТЕМИ КОНТРОЛЮ КОНТЕЙНЕРНИХ СЕРЕДОВИЩ

3.1 Вибір і обґрунтування принципів побудови функціональної схеми проектованого об’єкта

Контейнеризація представляє собою передову технологію віртуалізації, яка дозволяє збирати, транспортувати та запускати програмні додатки разом із їхніми залежностями в ізольованих контейнерах. Кожен контейнер має свою власну легковажку операційну систему та включає всі необхідні бібліотеки та середовище виконання, забезпечуючи переносимість та консистентність його роботи навіть в різних середовищах.

Контейнеризація дозволяє розробникам та адміністраторам створювати і розгортати додатки більш швидко та ефективно, забезпечуючи ізоляцію між контейнерами і забезпечуючи консистентність середовища від розробки до виробництва.

Виходячи с переліка обраних наявних програм було проведено аналіз та зіставлено схему яка демонструє їх методу розгортання яка відображена на рисунку 3.1 та демонструє відмінність віртуалізації та контейнеризації та принци реалізації обраних застосунків [13].

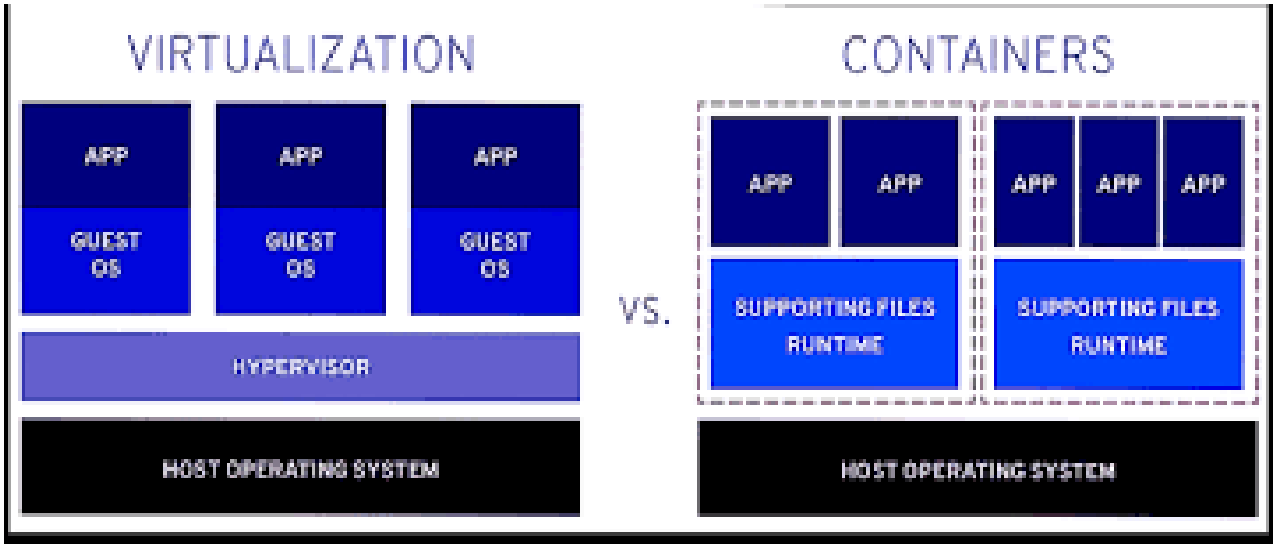


Рисунок 3.1 – Схема роботи розгортання при використання контейнеризації та віртуалізації

Також для впровадження контейнеризації в ігровій студії необхідно вибрати такі принципи побудови функціональної схеми, які забезпечать:

– масштабованість. Контейнери можна легко масштабувати, додаючи або видаляючи їх у міру необхідності. Це дозволяє ігровій студії швидко реагувати на зміни в попиту на її послуги;

– ефективність використання ресурсів. Контейнери можна ефективно використовувати ресурси інфраструктури, наприклад, виділяючи кожному контейнеру лише ті ресурси, які він потребує. Це дозволяє заощадити кошти на хостингу;

– безпеку. Контейнери можна використовувати для ізоляції програм від один одного, що допомагає захистити їх від взломів.

На основі цих принципів можна запропонувати таку функціональну схему проектованого об’єкта ігрової студії:

– сервер контейнерів. Сервер контейнерів відповідає за розгортання та управління контейнерами. Він може бути фізичним сервером або віртуальною машиною;

– репозиторій контейнерів. Репозиторій контейнерів зберігає контейнери, які використовуються в ігровій студії. Він може бути локальним або хмарним;

– система керування контейнерами. Система керування контейнерами дозволяє ігровій студії управляти контейнерами, наприклад, створювати, запускати, зупиняти та видаляти їх.

Обґрунтування принципів [13]:

– масштабованість. Контейнери можна легко масштабувати, додаючи або видаляючи їх у міру необхідності. Це дозволяє ігровій студії швидко реагувати на зміни в попиту на її послуги. Наприклад, якщо кількість гравців у грі різко збільшується, ігрову студію може просто додати більше контейнерів, щоб обробити додатковий трафік;

– ефективність використання ресурсів. Контейнери можна ефективно використовувати ресурси інфраструктури, наприклад, виділяючи кожному

контейнеру лише ті ресурси, які він потребує. Це дозволяє заощадити кошти на хостингу. Наприклад, ігрову студію може виділити кожному контейнеру лише ту кількість пам’яті, яка необхідна для його роботи;

– безпека. Контейнери можна використовувати для ізоляції програм від один одного, що допомагає захистити їх від взломів. Наприклад, ігрову студію може розгорнути різні ігри в окремих контейнерах, щоб запобігти взлому однієї гри через іншу.

Виходячи с потреб та доцільності перевага відходить у більшій мірі Docker цей продукт виходячи зі специфіки організації є потреба використання стандартних контейнерів. Це дозволить їй використовувати готові рішення для розгортання та управління контейнерами [13].

3.2 Обґрунтування прийнятих способів проектування і дослідження

Основні методи реалізації контейнеризації включають такі найбільш вживані програмні застосунки які приведені далі с пояснювальною характеристикою наведено нижче:

– docker – найпопулярніший інструмент для контейнеризації. Він надає можливість створювати, розгортати та управляти контейнерами. Docker використовує docker engine для управління контейнерами та використовує dockerfile для опису конфігурації контейнера;

– kubernetes – це оркестратор контейнерів, який дозволяє автоматизувати розгортання, керування та масштабування контейнеризованих додатків. він забезпечує оркестрацію контейнерів на кластері серверів;

– containerd – це менеджер контейнерів, який використовується багатьма платформами та оркестраторами. Він надає базовий рівень функціональності для роботи з контейнерами;

– openshift – це розширена платформа контейнеризації на основі kubernetes, яка включає в себе додаткові функції для автоматизації розгортання та керування контейнерами;

– lxc (linux containers) – це технологія віртуалізації на рівні ОС для створення системних контейнерів. Вона використовує віртуальні простори імен і управляє ресурсами на рівні ядра linux;

– rkt (rocket) – це інший менеджер контейнерів, який надає альтернативу docker. Він був розроблений за ідеєю більшої простоти та безпеки;

Переваги Docker над іншими програмами котрі були перелічені раніше. Docker має ряд переваг перед іншими платформами контейнеризації, включаючи:

– простота використання docker є одним із найпростіших у використанні застосунків для контейнеризації. Він пропонує простий інтерфейс командного рядка, який дозволяє розробникам швидко та легко створювати, розгортати та керувати контейнерами;

– широкий спектр функцій docker пропонує широкий спектр функцій, включаючи управління контейнерами, розгортання контейнерів і масштабування контейнерів. Це дозволяє розробникам створювати складні контейнерні системи;

– відкрита платформа docker є відкритою платформою, що означає, що вона підтримується спільнотою розробників. Це забезпечує стабільність та підтримку платформи.

Docker є хорошим вибором для розробників і організацій, які хочуть отримати швидкий старт у контейнеризації. Він простий у використанні, пропонує широкий спектр функцій і підтримується спільнотою розробників [11, 13].

Таблиця 3.1 – Порівняння Docker з іншими платформами контейнеризації

Функція	Docker	Kubernetes	Containerd	Open Shift	LXC	Rocket
Простота використання	Простий	Складний	Простий	Складний	Простий	Складний
Функціональність	Широкий спектр	Широкий спектр	Обмежений	Широкий спектр	Обмежений	Широкий спектр
Стабільність	Стабільна	Стабільна	Стабільна	Стабільна	Стабільна	Стабільна
Підтримка	Широка	Широка	Широка	Широка	Широка	Широка
Вартість	Безкоштовний	Безкоштовний	Безкоштовний	Безкоштовний	Безкоштовний	Безкоштовний

Docker є хорошим вибором для розробників і організацій, які хочуть отримати швидкий старт у контейнеризації. Він простий у використанні, пропонує широкий спектр функцій і підтримується спільнотою розробників. Однак для більш складних потреб контейнеризації можуть знадобитися більш потужні платформи, такі як Kubernetes або OpenShift.

Тож для більш детального розуміння було розібрано принципи функціонування платформи Docker, які відображено на рисунку 3.2.

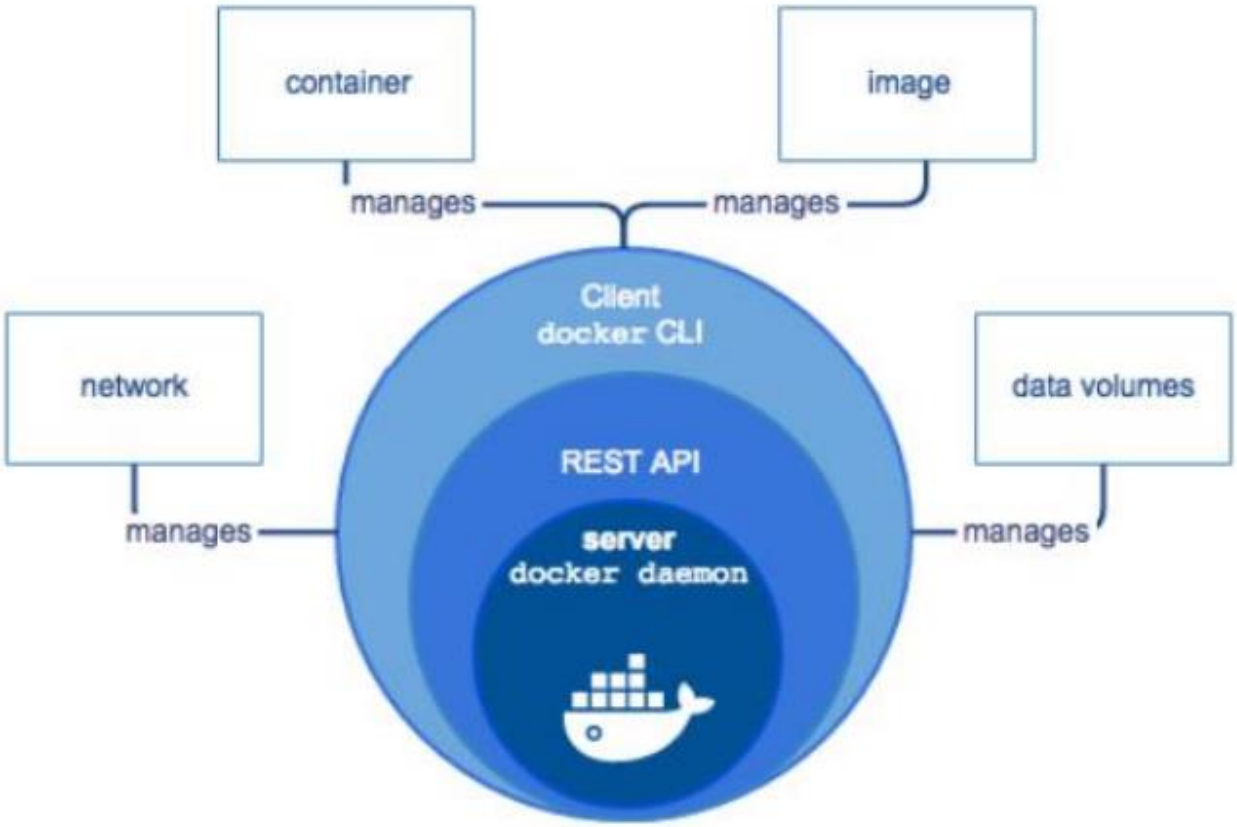


Рисунок 3.2 – Схема розгортання моделі с використанням Docker

Дослідження було проведено після реалізації створення та налаштувань контейнерів та створення тестового додатку, для розгортання так як умови дослідження потребують використання однакового додатку, для збору більш достовірних показників тож для забезпечення цих вимог було використано ізольовані контейнери. Відповідно Docker контейнери, яка дозволяє розробникам пакувати, розгортати та запускати програми в ізольованому середовищі.

Контейнери представляють собою легкі, ізольовані середовища, які містять все необхідне для запуску програми, включаючи операційну систему, бібліотеки, інструменти та код програми.

При впровадженні Docker в ігровій студії слід враховувати такі способи проектування і дослідження.

Необхідно розробити архітектуру контейнерних систем, яка буде відповідати потребам ігрової студії. Ця архітектура повинна визначати, як

контейнери будуть взаємодіяти один з одним, як вони будуть масштабуватися і як вони будуть керуватися.

Необхідно вжити заходів безпеки для захисту контейнерів від взломів і інших проблем. Ці заходи безпеки можуть включати в себе використання брандмауерів, шифрування та аутентифікації.

Необхідно вжити заходів для покращення продуктивності контейнерних систем. Ці заходи можуть включати в себе оптимізацію контейнерів, використання хмарних технологій і використання високопродуктивних серверів [13].

Нижче наведено кілька конкретних прикладів способів проектування і дослідження для Docker у впровадженні ігрової студії:

Архітектура контейнерних систем:

– ігрова студія може використовувати архітектуру мікросервісів, щоб розбити свою гру на невеликі, незалежні частини, які можуть бути упаковані в контейнери. Це дозволить студії легко масштабувати гру і додавати нові функції.

Безпека:

– ігрова студія може використовувати брандмауер для захисту контейнерів від несанкціонованого доступу. Студія також може використовувати шифрування для захисту даних, які зберігаються в контейнерах.

Покращена продуктивність:

– ігрова студія може використовувати хмарну технологію для запуску контейнерів. Хмарні технології можуть забезпечити ігровій студії доступ до високопродуктивних серверів і ресурсів.

3.3 Проектування системи розгортання при використані технологій контейнеризації та автоматизації

Система розгортання при використані Docker складається з таких компонентів [12]:

– інструмент для створення контейнерів цей інструмент використовується для створення контейнерів з кодом програми, бібліотеками та іншими необхідними ресурсами;

– інструмент для розгортання контейнерів цей інструмент використовується для розгортання контейнерів на серверах;

– інструмент для управління контейнерами цей інструмент використовується для управління контейнерами, такими як запуск, зупинка та видалення.

Приклад системи розгортання при використані Docker нижче наведено приклад системи розгортання при використані Docker:

– інструмент для створення контейнерів: Docker;

– інструмент для розгортання контейнерів: Docker swarm;

– інструмент для управління контейнерами: Docker engine.

Ця система розгортання використовує Docker Swarm для розгортання контейнерів на кластері серверів. Docker Engine використовується для управління контейнерами після їх розгортання.

Ось деякі додаткові частини щодо проектування системи розгортання при використані Docker:

– використано стандартизовані процеси і інструменти це допоможе забезпечити послідовність і відтворюваність розгортання;

– автоматизовано всі можливі завдання це допоможе звільнити час для розробників і операційних команд;

– використовується систему контролю версій для зберігання контейнерів це допоможе відстежувати зміни в контейнерах і відновлюватися після невдалих розгортань;

– впроваджується систему моніторингу для відстеження стану контейнерів це допоможе виявити і усунути проблеми до того, як вони призведуть до перерв у роботі.

Створення коду–інструкції для реалізації розгортання починається с написання інструкцій на мові Bah [12].

Для реалізації контейнерного середовища при використані Linux сервера надає змогу виконувати розгортання багатьох контейнерів паралельно, що допомагає використовувати ефективно апаратні можливості сервера розгортки.

Тож було зіставлено короткий код автоматизації. Спочатку було створено файл з назвою user-input.sh та потім для реалізації були внесені правки в функціональність виконання та змін режиму роботи а на виконуваний за допомогою команди chmod. Встановив параметри на a+x, щоб код виконувався – x усіма користувачами – a. Після використання chmod потрібно відстежити, що дозволи були змінені для користувачів, груп та інших осіб, додалось x – виконуваний файл ці виконані кроки відображені на рисунку 3.3.

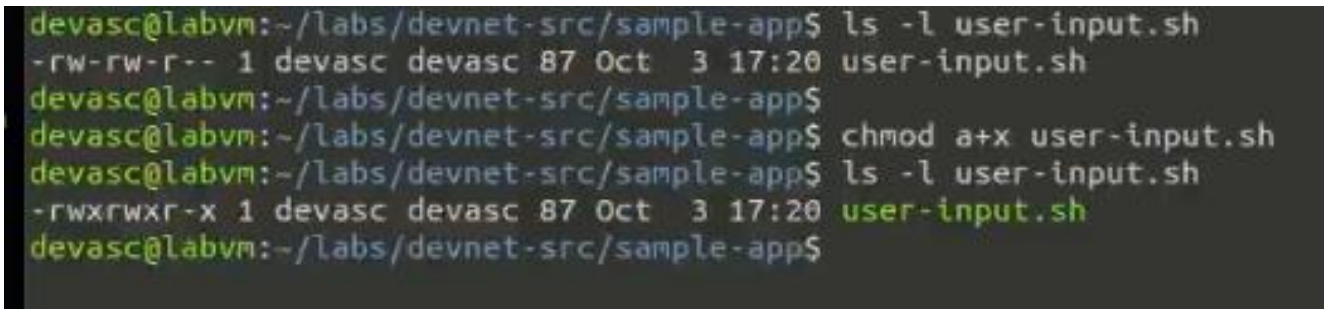


Рисунок 3.3 – Реалізація присвоєння політик для задання режиму роботи

Для багаторазового використання при подальшому використані цього короткого коду в автоматичному режимі подальших створень контейнерів було виконано видалення розширення файлу .sh за допомогою маніпуляцій які наведено нижче на рисунку 3.4.

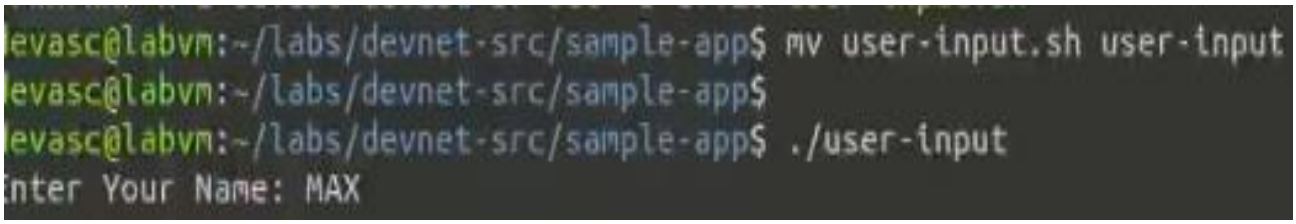


Рисунок 3.4 – Видалення закінчення для коректної роботи коду

На цій частині роботи для реалізації створюваних контейнерів потрібно було розроблено тестовий додаток який в свої суті повинен нести відображення доступності до застосунку у разі його коректного розгортання в контейнеризованому середовищі. Відповідно веб–застосунок несе суто демонстраційний характеру без наявного реального фунціоналу кінцевого продукту.

Для проведення розгортання та розробки веб–застосунків, які використовують Python, зазвичай використовують фреймворк. Framework – це бібліотека коду, яка спрощує розробникам створення надійних, масштабованих і прийнятних для підтримки веб–застосунків. Flask – це веб–застосунок, написаний на Python. Інші фреймворки це Tornado і Pyramid.

Flask отримує запити, а потім надає відповідь користувачеві у веб–застосунку. Це зручно для динамічних веб–застосунків, оскільки дозволяє взаємодіяти з користувачем та динамічним вмістом. Те, що робить зразок веб–застосунку динамічним, полягає в тому, що він буде відображати IP–адресу клієнта у разі успішного розгортання в контейнері [12].

Встановлення фреймворку Flask було виконано за допомогою такої команди.

devasc@labvm:~/labs/devnet-src/sample-app$ pip3 install flask

Потім потрібно створити файл sample_app.py для інтеграції фреймворк flask де було додано бібліотеки які наведено нижче:

– from flask import Flask;

– from flask import request.

Та потім було додано екземпляр класу Flask з його новим найменування sample, реалізація цієї команди наведено нижче.

```
sample = Flask(__name__)
```

Був створений метод відображення IP-адреси клієнта за допомогою налаштування Flask таким чином, щоб коли користувач відвідуватиме сторінку за замовчуванням (кореневий каталог), відображалося повідомлення з IP-адресою клієнта реалізація цього методу наведено нижче.

```
@sample.route("/")
def main():
    return "You are calling me from " + request.remote_addr + "\n"
```

Налаштування застосунку для розгортання на локальному сервері. Відладка Flask для запуску застосунку локально за адресую http://0.0.0.0:8080, яка аналогічна http://localhost:8080 та для реалізації у інструкції розгортання потрібно задати ці параметри такі як хост адресу та порт наведено нижче.

```
if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=8080)
```

Запуск та розгортання застосунку для перевірки доступності користувачам відображено на риску 3.5.

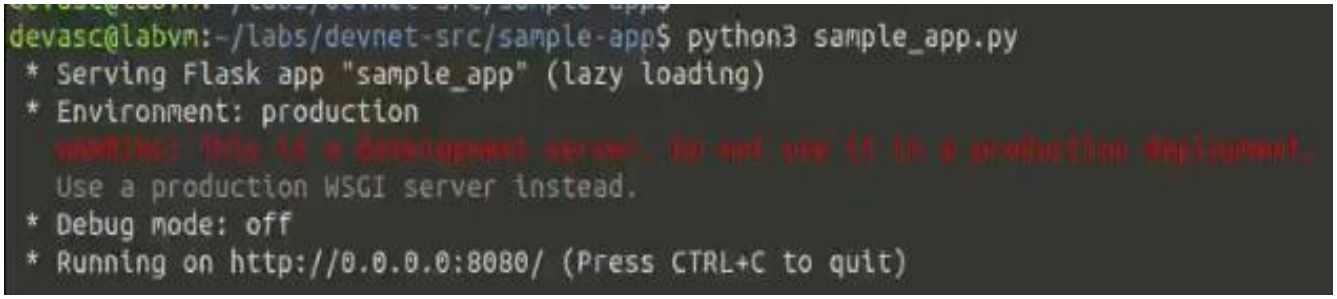


Рисунок 3.5 – Демонстрація розгортання тестового застосунку

Перевірка працездатності розгорнутого сервера в контейнері наочним способом продемонструвала доступність користувача що наведено на рисунку 3.6.

Рисунок 3.6 – Перевірка роботи та доступності сервера котрий був розгорнутий в контейнері

Для оптимізації роботи розгортання будо додано додаткові частини коду. За допомогою функції render_template файл HTML може бути відтворений у Flask автоматично. Для цього було імпортовано метод render_template з бібліотеки flask та відредагувати його у функцію return. Зміни внесені в код наведено нижче [11].

```
from flask import Flask
from flask import request
from flask import render_template
sample = Flask(__name__)
@sample.route("/")
def main():
        return render_template("index.html")
if __name__ == "__main__":
         sample.run(host="0.0.0.0", port=8080)
```

На рисунку 3.7 наведено повторне розгортання сервера але вже з внесеними правками оптимізації при розміщені застосунка.

Рисунок 3.7 – Оптимізоване розгортання веб застосунку

Використавши команду curl для перевірки відповіді сервера. Там можливо поспостерігати результат автоматичного відтворення HTML–коду за допомогою функції render_template. В цьому випадку отримано весь HTML–контент. Однак динамічний код Python буде замінений із значенням для {{request.remote_addr}}. Крім того, створене запрошення буде в тому ж рядку, що і останній рядок HMTL-виведення. Результат спрацювання наведено нижче на рисунку 3.6 [11].

Після створення та відладки експериментального веб–застосунка у взаємодії з сервером обробки цього застосунку. У подальшому було виконано реалізацію зі створення коду Bash для збірки та запуску контейнера Docker с використанням технологій паралельного розгортання декількох контейнерних середових для реалізації експериментального застосунку.

Застосунок можна розгорнути на сервері з «голого заліза» (фізичний сервер, виділений для одноклієнтського середовища) або на віртуальній машині, яку зробили у попередній частині проведених робіт. Його також можна розгорнути у контейнерному рішенні, такому як Docker. В цій частині буде створено код програми bash і додано до нього команди, які виконують наступні завдання для збірки та запуску контейнера Docker:

– створення тимчасових каталогів для зберігання файлів веб-сайту;

– копіювання каталогів веб-сайту та sample_app.py до тимчасового каталогу;

– збірка dockerfile;

– збірка контейнера docker;

– запуск контейнера і перевірка його роботи.

Було проведено створення тимчасового каталогу для зберігання файлів веб-сайту. Використовуючи файл програми bash sample-app.sh в каталозі ~/labs/devnet-src/sample-app. Було додано 'shebang' і команди, щоб створити структуру каталогів з tempdir як батьківський каталог що відображено на рисунку 3.8.

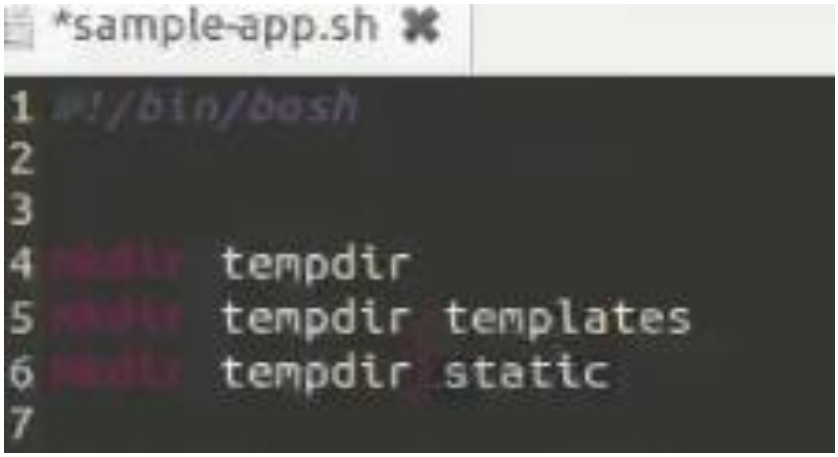


Рисунок 3.8 – Створення батьківського каталогу за допомого bash

Проведення додавання копіювання автоматичного копіювання структури каталоги веб–сайту та sample_app.py у тимчасовий каталог. У файл sample-app.sh додано команди для копіювання каталогу експериментального веб–сайту та програми в tempdir, що продемонстровано на рисунку 3.9.

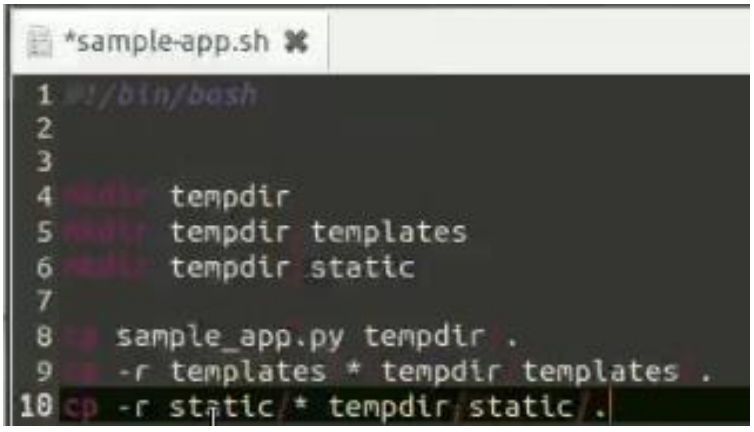


Рисунок 3.9 – Зіставлена програма автоматичного створення / копіювання експериментального веб–застосунку для збірника Dockerfile

Виконання збірки Dockerfile. На цьому кроці необхідно додати команди bash echo в файл sample-app.sh для створення Dockerfile в tempdir. Цей Dockerfile буде використаний для збірки подальших контейнерів при використання експериментального веб–додака [12, 13].

Потрібно запустити Python в контейнері, тому була додана команда Docker FROM, щоб встановити Python в контейнер.

echo "FROM python" >> tempdir/Dockerfile

Додавання команду Docker RUN, щоб встановити Flask у контейнер.

echo "RUN pip install flask" >> tempdir/Dockerfile

Для запуску застосунку розроблюваному контейнері знадобляться папки веб–додатку та скрипт sample_app.py, тому додається команда Docker COPY, щоб додати їх до каталогу в контейнері Docker. В цьому випадку створиться /home/myapp як батьківський каталог всередині контейнера Docker. Окрім копіювання файлу sample_app.py у Dockerfile, також скопіюєте файл index.html із каталогу templates та файл style.css із каталогу static.

echo "COPY ./static /home/myapp/static/" >> tempdir/Dockerfile

echo "COPY ./templates /home/myapp/templates/" >> tempdir/Dockerfile

echo "COPY sample_app.py /home/myapp/" >> tempdir/Dockerfile

За допомогою команди Docker EXPOSE відкрийте порт 8080 для використання веб-сервером.

echo "EXPOSE 8080" >> tempdir/Dockerfile

Додання команди Docker CMD для виконання скрипта Python

echo "CMD python3 /home/myapp/sample_app.py" >> tempdir/Dockerfile

Після внесення всіх налаштувань для автоматичної розгортки контейнерів для тестового веб–застосунку було виконану збірку контейнера Docker для подальшого використання що зображено на рискну 3.10.

Додання команди до файлу sample-app.sh, щоб перейти до каталогу tempdir і виконати збірку контейнера Docker. Параметр -t команди docker build дозволяє вказати ім'я контейнера, а точка вкінці (.) вказує, щоб контейнер був вбудований у поточний каталог.

cd tempdir

docker build -t sampleapp .

Запуск контейнера та перевірка, чи він запущений. Додання команду docker run до файлу sample-app.sh, щоб запустити контейнер. docker run -t -d -p 8080:8080 --name samplerunning sampleapp.

Параметри docker run вказують на наступне:

– -t вказує на те, що буде створений термінал для контейнера, щоб була змога отримати до нього доступ у командному рядку;

– -d вказує на те, щоб контейнер працював у фоновому режимі та друкував ідентифікатор контейнера під час виконання команди docker ps -a;

– -p вказує на те, що буде опублікуватися внутрішній порт контейнера на вузлі. Перший "8080" посилається на порт для застосунку, що працює в контейнері docker (створений sample-app). Другий "8080" вказує docker використовувати цей порт на вузлі. Ці значення не повинні обов’язково бути однаковими. Наприклад, внутрішній порт 80 на зовнішній 800 (80:800);

– --name вказує спочатку те, що викликати екземпляр контейнера (samplerunning), а потім образ контейнера, на якому буде заснований екземпляр (sampleapp). Ім'я екземпляра може бути довільним. Натомість ім'я образу має співпадати з іменем контейнера, яке було вказано в команді docker build (sampleapp);

Додання команди docker ps -a, щоб відобразити всі запущені в даний момент контейнери Docker. Ця команда буде останньою, виконаною скриптом bash. Виконане зіставлення інструкції с потрібними функціями розгортання продемонстровано на рисунку 3.10.

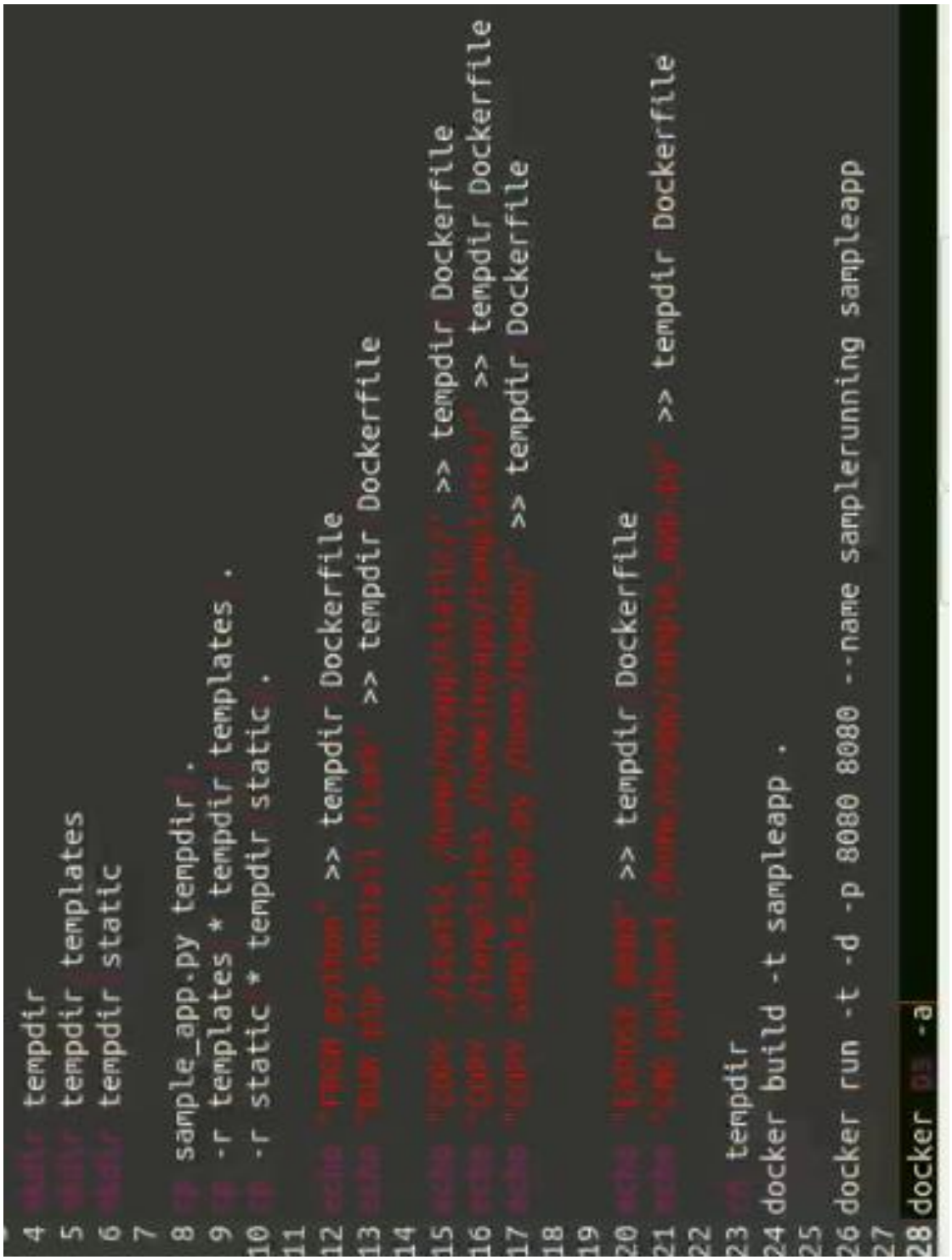


Рисунок 3.10 – Зіставлення інструкції для створення контейнера

Виконання запуску та розгортання контейнерного середовища у автоматичному режимі при використані Docker та програми bash відображено на рисунку 3.11 – 3.12.

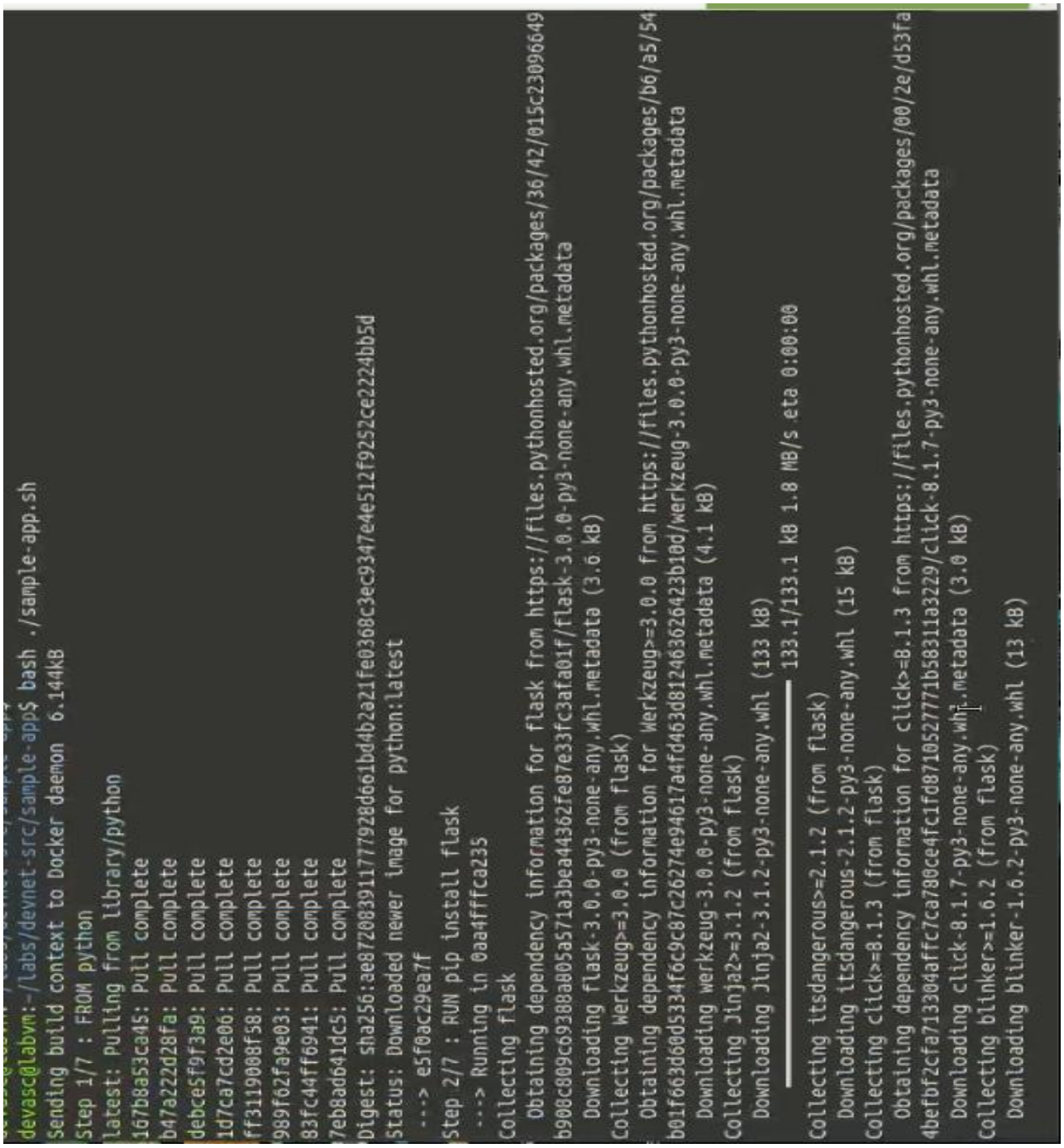


Рисунок 3.11 – Демонстрація пройдених підготовчих етапів проведеної розгортки контейнерного середовища

Рисунок 3.12 – Демонстрація пройдених етапів розгортання контейнера в автоматичному режимі

При успішному розгортанні можна спостерігати наявність файл під назвою bash Dickerfile цей файл містить пройдені важливі етапи розгортання в контейнерному середовищі усіх компонентів тестового веб–додатка що продемонстровано на рисунку 3.13.

```
devasc@labvm:~/labs/devnet-src/sample-app$ cat tempdir/Dockerfile
FROM python
RUN pip install flask
COPY ./static /home/myapp/static/
COPY ./templates /home/myapp/templates/
COPY sample_app.py /home/myapp/
EXPOSE 8080
CMD python3 /home/myapp/sample_app.py
devasc@labvm:~/labs/devnet-src/sample-app$
```

Рисунок 3.13 – Демонстрація завершених етапів розгортки контейнера та розгорнутого в ньому застосунку

Перевірка роботи та коректного доступу експериментального застосунку в контейнерному середовищі відображено на рисунку 3.14.

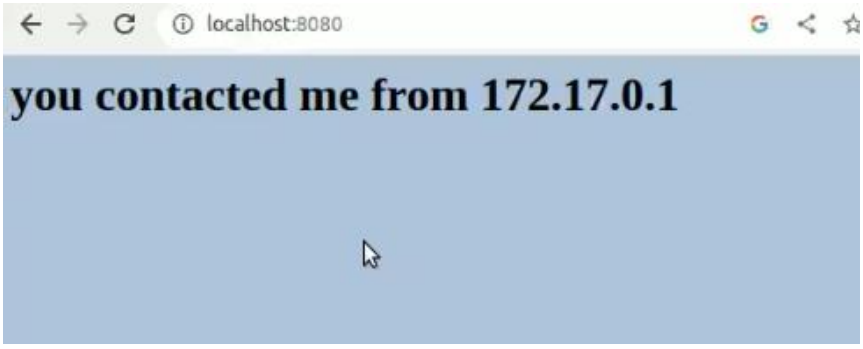


Рисунок 3.14 – Результат роботи демонстраційного веб–застосунку у контейнерному середовищі

Щоб отримати доступ до запущеного контейнера, потрібно використати команду docker exec -it із зазначенням імені запущеного контейнера (samplerunning) та потрібної оболонки bash (/bin/bash). Параметр -i вказує на те, щоб він був інтерактивним, а параметр -t визначає те, щоб отримати доступ до терміналу. Запрошення змінюється на root@containerID.

Є можливість використання кореневого доступу для контейнерів Docker samplerunning. Звідси дозволяється використовувати команди Linux для

дослідження та модифікації контейнера Docker що продемонстровано за допомогою рискну 3.15.

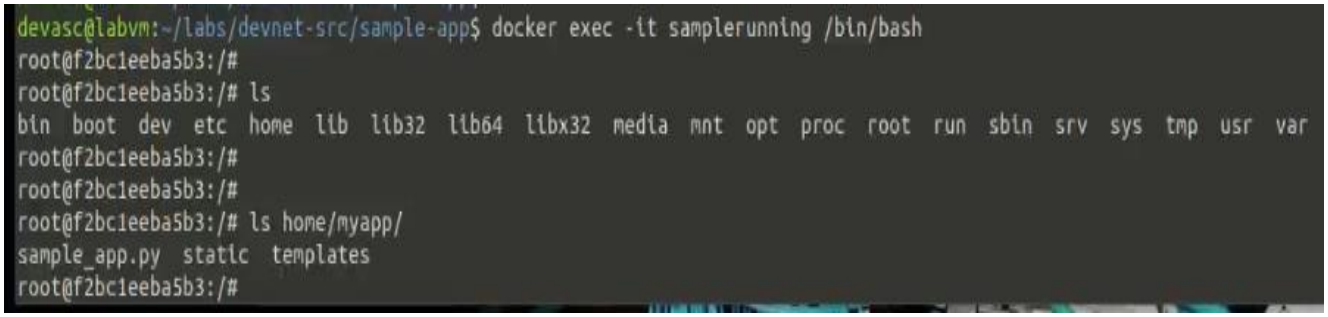


Рисунок 3.15 – Демонстрація наявності коректних каталогів розгорнутої системи та тестованого додатку

Після проведених етапів розгортання одержано контейнерні середовищ перелік створених та розгорнутих контейнерів наведено на рисунку 3.16 з включенням демонстрації роботи на одному із них веб–сервера застосунка наведено на рисунку 3.17.

Рисунок 3.16 – Перелік створених та працюючих контейнерів с їх параметрами

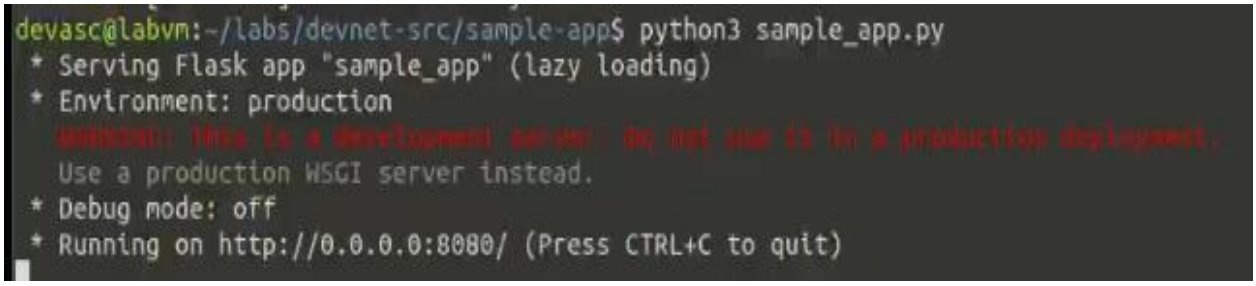
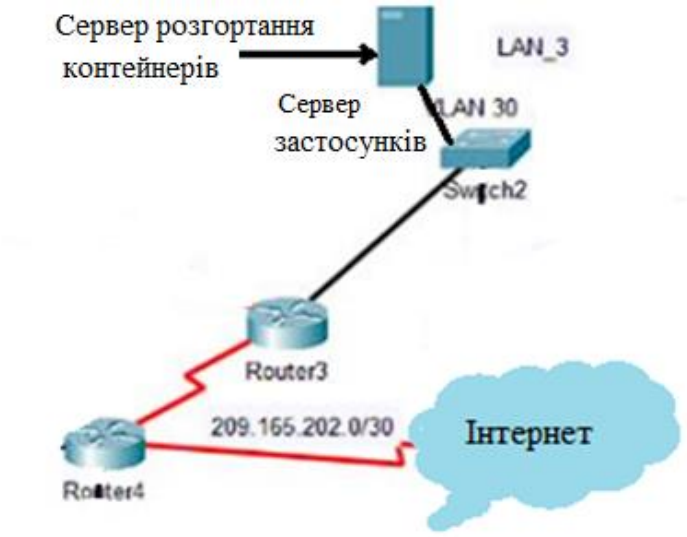


Рисунок 3.17 – Демонстрація роботи запушеного веб сервера в середині контейнера у локальному доступі

Тож на фінальному етапі реалізації було одержано та досліджено тестову модель експериментального веб–додатка для подальшого експерименту в наступних розділах роботи та створений та відлагодженої програми процедурної автоматизації та зібраний шаблон докер контейнера котрий може масштабуватися та паралельно розгортатися на одному й тому сервері застосунків. Та перевірений на працездатність та доступність для зовнішніх користувачів [11].

3.3.1 Розробка функціональних схем вузлів проектованої системи

Створення функціональних схем вузлів системи ігрової студії з використанням Docker передбачає розгляд діючих компонентів системи та їхніх взаємодій в контексті контейнеризації і розміщення на одному головному сервері і його параметри наведено в розділі 3.3.2 та наведено структура розміщення використовуваної апаратної частини на рисунку 3.18.



Рисунку 3.18 – Архітектура впровадження серверу розгортання контейнерів

Є декілька технологій та сервісів котрі будуть використовуватись та розгортатися для кожного проекту вони можуть слугувати як головним по типу веб–сервера котрий використовується для доступу вкладених сторінок

продукту нових та старих проектів. Ось декілька складових вузлів та їх функціональна загальна схема вузлів [13]:

– веб-сервер (Web Server) та контейнер з веб-сервером, який обробляє HTTP–запити від клієнтів та містити веб–сайт з переліком та доступом до усіх продуктів компанії, API, адміністративний інтерфейс;

– база даних (Database) контейнер з базою даних містить систему управління базами даних (наприклад, MySQL, PostgreSQL, MongoDB) та базу даних з ігровою інформацією;

– ігровий сервер (Game Server) контейнер ігрового сервера містить ігровий сервер, який взаємодіє з клієнтами гри та базою даних для обробки гравців і ігрової інформації;

– автоматизація розгортання (Deployment Automation) контейнер автоматизації розгортання може містити інструменти для автоматичного розгортання нових версій програмного забезпечення або ігрового контенту.

Так як розгортання та демонстрація дослідження проводиться с використанням розробленими тестовим веб–застосунком на сервері застосунків то більш детально потрібно провести розбір функціональних вузлів котрі були використані.

Контейнер з веб–сервером відповідає за обробку запитів HTTP від клієнтів і надсилає відповіді. При цьому в середині веб–сервер слухає на певному порту (наприклад, порт 80 для HTTP або порт 443 для HTTPS) і реагує на вхідні запити. Відповідно для демонстрації проведеного роботи продемонстровано на рисунках 3.19 – 3.21 такі частини як:

– демонстрація розгорнутих контейнерів з їх іменами та даними роботи;

– демонстрація пройдених конфігураційних кроків;

– демонстрація перегляду залежностей конкретного пакета веб–сервера;

– демонстрація шарів розгорнутого контейнера.

```
devasc@labvm:~/labs/devnet-src/sample-app/TTT$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
sampleapp1   latest    92a58c840704   7 weeks ago   1.03GB
sampleapp    latest    92a58c840704   7 weeks ago   1.03GB
python       latest    e5f0ac29ea7f   7 weeks ago   1.02GB
devasc@labvm:~/labs/devnet-src/sample-app/TTT$ docker ps -a
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS                      PORTS      NAMES
5a6915579cd3   sampleapp   "/bin/sh -c 'python3…"   43 seconds ago   Up 43 seconds               8080/tcp   container3
4107c8bbc93f   sampleapp   "/bin/sh -c 'python3…"   44 seconds ago   Up 43 seconds               8080/tcp   container2
4dd1af57e79e   sampleapp   "/bin/sh -c 'python3…"   44 seconds ago   Up 44 seconds               8080/tcp   container1
f2bc1eeba5b3   sampleapp   "/bin/sh -c 'python3…"   7 weeks ago      Exited (137) 7 weeks ago               samplerunning
devasc@labvm:~/labs/devnet-src/sample-app/TTT$
devasc@labvm:~/labs/devnet-src/sample-app/TTT$
```

Рисунок 3.19 – Демонстрація створених образів та розгорнутих контейнерів

```
root@4dd1af57e79e:/#
root@4dd1af57e79e:/# apt-cache depends python3
python3
  PreDepends: python3-minimal
  Depends: python3.11
  Depends: libpython3-stdlib
  Suggests: <python3-doc>
  Suggests: <python3-tk>
  Suggests: <python3-venv>
  Replaces: python3-minimal
root@4dd1af57e79e:/#
```

Рисунок 3.20 – Демонстрація залежностей контейнера веб–сервера

```
devasc@labvm:~/labs/devnet-src/sample-app/TTT$ docker history sampleapp
IMAGE          CREATED        CREATED BY                                      SIZE      COMMENT
92a58c840704   7 weeks ago    /bin/sh -c #(nop)  CMD ["/bin/sh" "-c" "pyth…  0B
9910894bb7ac   7 weeks ago    /bin/sh -c #(nop)  EXPOSE 8080                  0B
9f6cdda36a02   7 weeks ago    /bin/sh -c #(nop) COPY file:612335cb0cff7635…   298B
9f17dd9c37bd   7 weeks ago    /bin/sh -c #(nop) COPY dir:d165617e4859f9bd6…   198B
add417cc6c7c   7 weeks ago    /bin/sh -c #(nop) COPY dir:68371ba8b8fb4f547…   35B
637bffe5ba0a   7 weeks ago    /bin/sh -c pip install flask                    15MB
e5f0ac29ea7f   8 weeks ago    CMD ["python3"]                                 0B        buildkit.dockerfile.v0
<missing>      8 weeks ago    RUN /bin/sh -c set -eux;   wget -O get-pip.p…  10MB      buildkit.dockerfile.v0
<missing>      8 weeks ago    ENV PYTHON_GET_PIP_SHA256=45a2bb8bf2bb5eff16…   0B        buildkit.dockerfile.v0
<missing>      8 weeks ago    ENV PYTHON_GET_PIP_URL=https://github.com/py…   0B        buildkit.dockerfile.v0
<missing>      8 weeks ago    ENV PYTHON_PIP_VERSION=23.2.1                   0B        buildkit.dockerfile.v0
<missing>      8 weeks ago    RUN /bin/sh -c set -eux;  for src in idle3 p…   32B       buildkit.dockerfile.v0
<missing>      8 weeks ago    RUN /bin/sh -c set -eux;   wget -O python.ta…   60.3MB    buildkit.dockerfile.v0
<missing>      8 weeks ago    ENV PYTHON_VERSION=3.12.0                       0B        buildkit.dockerfile.v0
<missing>      8 weeks ago    ENV GPG_KEY=7169605F62C751356D054A26A821E680…   0B        buildkit.dockerfile.v0
<missing>      8 weeks ago    RUN /bin/sh -c set -eux;  apt-get update;  a…   18.6MB    buildkit.dockerfile.v0
<missing>      8 weeks ago    ENV LANG=C.UTF-8                                0B        buildkit.dockerfile.v0
<missing>      8 weeks ago    ENV PATH=/usr/local/bin:/usr/local/sbin:/usr…   0B        buildkit.dockerfile.v0
<missing>      2 months ago   /bin/sh -c set -ex;  apt-get update;  apt-ge…   587MB
<missing>      2 months ago   /bin/sh -c apt-get update && apt-get install…   177MB
<missing>      2 months ago   /bin/sh -c set -eux;  apt-get update;  apt-g…   48.4MB
<missing>      2 months ago   /bin/sh -c #(nop)  CMD ["bash"]                 0B
<missing>      2 months ago   /bin/sh -c #(nop) ADD file:ce04d6a354feaef93…   116MB
devasc@labvm:~/labs/devnet-src/sample-app/TTT$
```

Рисунок 3.21 – Демонстрація шарів розгорнутого контейнера

Веб–сайт та додатки:

– у контейнері містяться файли веб-сайту та веб-додатків, які доступні через веб-сервер;

– це може включати html–файли, зображення, стилі css, javascript та інші статичні ресурси, а також виконувані програми або серверний код.

Залежності та конфігурація:

– в контейнер також можуть бути включені залежності, необхідні для виконання веб-додатків, такі як фреймворки або бібліотеки;

– конфігураційні файли, такі як файл конфігурації веб–сервера (наприклад, nginx.conf), також можуть бути включені в контейнер.

Системи логування та моніторингу:

– у веб-сервері були налаштовані системи логування та моніторингу, що допомагають відстежувати діяльність сервера та виявляти помилки або аномалії.

HTTPS та SSL-сертифікати:

– якщо вимагається безпека, то в контейнер є включені ssl–сертифікати для підтримки https. це забезпечує зашифровану комунікацію між сервером і клієнтами.

Автоматизоване розгортання:

– для спрощення розгортання та оновлення веб-сервера можна використовувати інструменти автоматизації, такі як docker compose або оркестратори контейнерів. Це дозволяє легко впроваджувати нові версії веб-сервера або швидко створювати кілька інстанцій для масштабування.

Наведена модель відображає складові кроки взаємодії веб-сервера та їх розгортання етапи в контейнерному середовищі представлено на рисунку 3.22. Реалізовані контейнери дозволяють ізолювати веб–застосунок та веб–сервер від інших компонентів системи, полегшує розгортання та масштабування, і дозволяє швидко створювати та керувати веб–додатком схема проведених розгортань в середовищі контейнеризації відображено на рисунку 3.23 [11, 13].

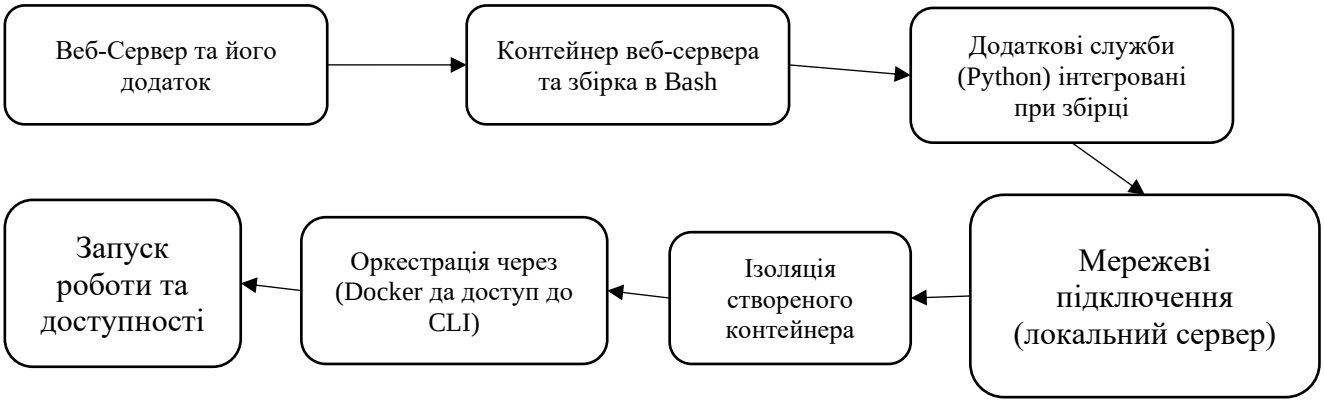


Рисунок 3.22 – Етапи розгортання веб–сервера в середині контейнера

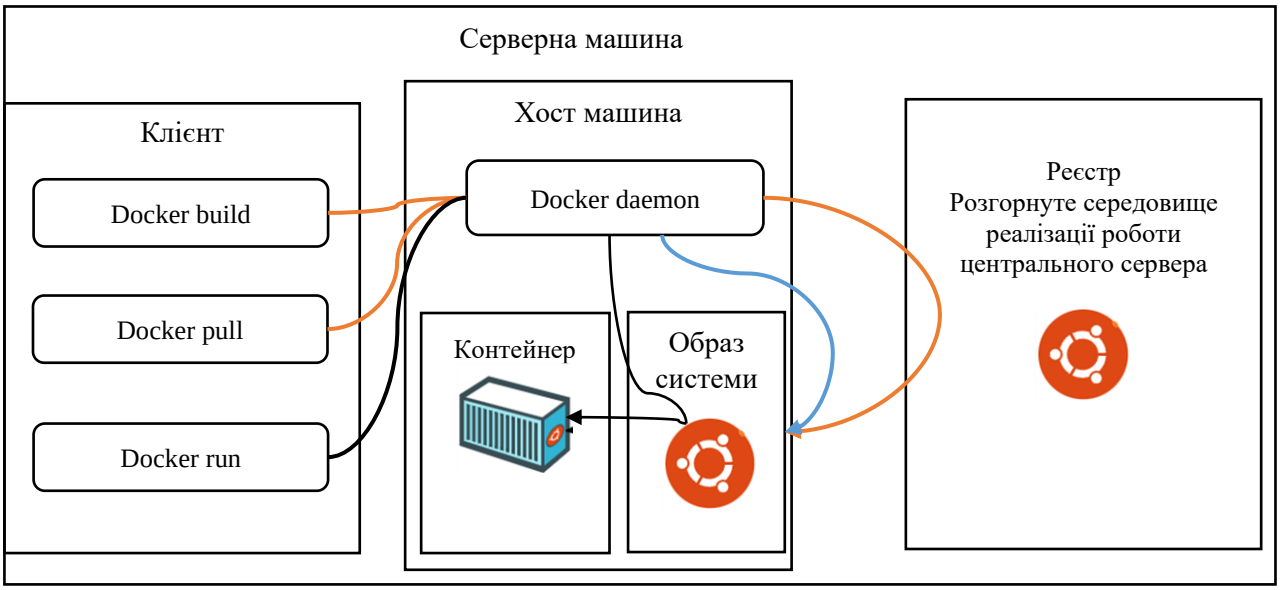


Рисунок 3.23 – Послідовність взаємодії веб–додатка на сервері

Докер–демон (Docker–daemon) – це сервер, який управляє контейнерами та компонентами Docker, такими як мережі, сховища, образи і контейнери. Він є невід'ємною частиною Docker і дозволяє взаємодіяти з Docker–системою.

Докер–клієнт (Docker–client / CLI) – це інтерфейс користувача для взаємодії з Docker–демоном. Він дозволяє користувачам виконувати команди та налаштування Docker. Важливою особливістю є те, що один клієнт може взаємодіяти з різними Docker–демонами.

Докер–образ (Docker–image) – це файл, який містить всю необхідну інформацію, залежності і конфігурацію для створення контейнера. Він

служить основою для контейнера і може бути створений або завантажений з Docker–реєстру.

Докер-файл (Docker–file) – це текстовий файл, що містить інструкції для створення Docker–образу. Він визначає кроки зі збирання образу, включаючи вибір базового образу, копіювання файлів та встановлення програм.

Докер-контейнер (Docker–container) – це невеликий та автономний пакет програмного забезпечення, який містить усе необхідне для запуску застосунку, включаючи код, середовище виконання, системні інструменти, бібліотеки та налаштування. Контейнери можуть бути запущені та виконуватися відокремлено один від одного.

Том (Volume) – це механізм для створення файлової системи для операцій читання і запису, яка може використовуватися в контейнері. Вона створюється автоматично та дозволяє зберігати дані, які можуть зберігатися поза контейнером.

Реєстр (Docker–registry) – це сервер, який використовується для зберігання Docker–образів. Він служить централізованим сховищем для обміну та розповсюдження образів між різними системами та користувачами Docker [13].

3.3.2 Вибір апаратних засобів і елементної бази

На підставі зібраних даних котрі були використані у експериментальному розділі було та спираючись на те що система у реальних робочих умовах використовує один головний сервер. На якому повинне працювати програмне забезпечення контейнерних технологій, та її працездатність на пряму залежить від якості апаратного забезпечення так як велика кількість зовнішніх користувачів одночасно висуває значну потребу у якісному обладнані з великими обрахунковими можливостями. Тому на цій підставі було обрано найбільш коректне апаратне забезпечення на базі двох центральних процесорів та великого сховища даних, так як користувачі будуть генерувати велику кількість даних з кожного аккаунта прогресії. Відповідно

потрібен значний об’єм оперативної пам’яті для більшої пропускної здатності та завантаженості процесорів у даному сервері.

Нижче приведені підібрані характеристики які покриватимуть усі потреби студії у даний час та ще перспективі від 2 до 4 років підтримки проектів на цьому сервері [2]:

– обрання даного сервера Cisco UCS-SPR-C240M4-P1 так як ігрова студія працює над великими проектами з високими потребами в ресурсах. Пам'ять оперативна (256 Гб), обчислювальна потужність (2 процесори Intel Xeon E5-2696 V4) і простір на жорсткому диску (100 Тв в SSD носіях) важливі для забезпечення швидкості та продуктивності контейнерів та урахуванням того що ще буде рейд масив для надійності даних;

– кластери і оркестратори якщо потрібно буде масштабувати наявні контейнери та керувати ними, варто розглянути використання контейнерних оркестраторів, таких як kubernetes або docker swarm. вони дозволяють розподілити контейнери між декількома серверами але це більше запас на майбутній розвиток компанії то і потрібно другий процесор та така наявна кількість 256 ОЗУ та накопичувачів від 50 Тв;

– мережева інфраструктура забезпечення надійної та швидкої мережевої інфраструктури важливе для ігрових продуктів. Було розглянуто можливість використання високошвидкісних комутаторів і маршрутизатор для забезпечення мінімальної затримки для користувачів то використовуються пограничний маршрутизатор марки Cisco 2811 ISR 4 він має такі характеристики пам'ять: RAM 512 Мб; флеш пам'ять 256 Мб; інтерфейси: 2 порти 100Base-TX / 1000Base-T, роз'єм RJ-45, 1 консольний порт управління, роз'єм RJ-45, 2 слотами HWIC, 1 порт USB 4-пін USB тип A; ОС базова Cisco IOS IP Base;

– сховище даних представляє собою два SSD ExaDrive DC 50 Тб. На перспективу ці сховища будуть розширюватися, але на дані вимоги цього об’єму вистачить для зберігання інформація користувачів та для системи контейнерно розвернутих середовищ, також розглянуто можливість

використання мережевих сховищ або файлових систем з підтримкою масштабованості. Також забезпечено резервне копіювання даних;

– балансування навантаження використання балансувальників навантаження, таких як nginx або haproxy, може допомогти розподілити трафік між різними контейнерами або серверами гри;

– безпека забезпечення безпеки ізоляції між контейнерами та системами є критично важливим для гри, потрібно розглянути можливість використання інструментів моніторингу та захисту від потенційних загроз;

– моніторинг і журналювання для відстеження продуктивності та виявлення помилок важливо використовувати інструменти моніторингу та журналювання, такі як prometheus, grafana, elk stack тощо;

– автоматизація і ci/cd використання систем автоматизації, таких як jenkins або gitlab ci/cd, може спростити розгортання, тестування і оновлення ваших контейнерів.

Загалом, вибір апаратних засобів і елементної бази повинен бути адаптованим до конкретних потреб нового створюваного проекту студії та враховувати плани щодо масштабування та надійності. Але було враховано при виборі апаратних компонентів останні та приблизному майбутньому потреби у нових стороніх продуктів та скільки вони можуть потребувати для стабільної роботи без додаткових оновлень елементної бази у перспективі наступних 2 років з моменту релізу продукту. Також важливо враховувати засоби для забезпечення безпеки та моніторингу створюваних контейнерного середовища[2].

3.3.3 Розроблення автоматизованих інструкцій для забезпечення контейнеризації

Згідно висунутих вимог впровадження системи контейнеризації Docker а саме розроблений код програми для реалізації повинен містити такі виконувані інструкції як [4]:

– розгортання веб–сервера для трестованого додатку да поміщення його в контейнер в автоматичному режимі;

– створення файлу докер задання йому інструкцій розгортання в динамічному режимі за допомогою коду програми автоматизації збирання Docker–контейнера;

– створення коду автоматизації збирання докер зі створенням каталогів розгортанням та створення автоматичного копіювання веб–додатку для розгортання та задання можливості доступу до налаштувань контейнера через CLI.

Відповідно до вимог було розроблено три файли з кодом програми котра виконує перелічені умови. Це зроблено для зручної модифікації уразі розгортання нового додатку, та у разі оновлення та додавання нових функцій і динамічного розширення контейнера [11].

Код програми 3.1 – Код для розгортання сервера додатку

```
from flask import Flask #Імпортує клас Flask
from flask import request # Імпортує об'єкт request для роботи з HTTP-запитами
from flask import render_template # Імпортує функцію render_template для відображення HTML-шаблонів.
sample = Flask(__name__) # Створює об'єкт Flask з іменем sample.
@sample.route("/") # Декоратор маршруту для головної сторінки
def main(): # Функція, яка буде виконана при переході на головну сторінку
    return render_template("index.html")
if __name__ == "__main__": # Перевірка, чи файл запущено як головний.
    sample.run(host="0.0.0.0", port=8080) # Запуск веб-додатка на сервері з IP-адресою "0.0.0.0" і портом 8080.
```

Код програми 3.2 – код для створення та розгортання контейнерів

```
mkdir tempdir # Створення каталогу з назвою "tempdir".
mkdir tempdir/templates #Створення підкаталогу "templates" у каталозі "tempdir".
mkdir tempdir/static #Створення підкаталогу "static" у каталозі "tempdir".
```

cp sample_app.py tempdir/. #Копіювання файлу "sample_app.py" до каталогу "tempdir".

cp -r templates/* tempdir/templates/. #Рекурсивне копіювання усіх файлів з каталогу "templates" в підкаталог "templates" у "tempdir".

cp -r static/* tempdir/static/. #Рекурсивне копіювання усіх файлів з каталогу "static" в підкаталог "static" у "tempdir".

echo "FROM python" >> tempdir/Dockerfile #Додавання рядка "FROM python" до файлу "Dockerfile" у "tempdir".

echo "RUN pip install flask" >> tempdir/Dockerfile #Додавання рядка "RUN pip install flask" до файлу "Dockerfile" у "tempdir".

echo "COPY ./static /home/myapp/static/" >> tempdir/Dockerfile #Додавання рядка для копіювання вмісту каталогу "static" у "Dockerfile".

echo "COPY ./templates /home/myapp/templates/" >> tempdir/Dockerfile #Додавання рядка для копіювання вмісту каталогу "templates" у "Dockerfile".

echo "COPY sample_app.py /home/myapp/" >> tempdir/Dockerfile #Додавання рядка для копіювання файлу "sample_app.py" у "Dockerfile".

echo "EXPOSE 8080" >> tempdir/Dockerfile #Додавання рядка "EXPOSE 8080" до файлу "Dockerfile" для вказання порту, на якому буде доступний додаток.

echo "CMD python3 /home/myapp/sample_app.py" >> tempdir/Dockerfile #Додавання рядка "CMD python3 /home/myapp/sample_app.py" до файлу "Dockerfile" для вказання команди, яка виконується при запуску контейнера.

cd tempdir #Перехід до каталогу "tempdir".

docker build -t sampleapp . #Створення Docker-образу з іменем "sampleapp" з використанням файлу "Dockerfile" у поточному каталозі (".").

docker run -t -d -p 8080:8080 --name samplerunning sampleapp #Запуск Docker-контейнера з образом "sampleapp" з вказаними параметрами.

docker ps –a #Перевірка стану всіх Docker-контейнерів, включаючи завершені.

Код програми 3.3 – Dockerfail код для коректного запуску розгорнутого додатку в створеному контейнері для доступу зовнішніх користувачів

FROM python # Використовує базовий образ Python для побудови вашого контейнера.

RUN pip install flask #Виконує команду для встановлення бібліотеки Flask в контейнері.

COPY ./static /home/myapp/static/ #Копіює вміст каталогу "static" з вашого проекту в каталог "/home/myapp/static/" в контейнері.

COPY ./templates /home/myapp/templates/ #Копіює вміст каталогу "templates" з вашого проекту в каталог "/home/myapp/templates/" в контейнері.

COPY sample_app.py /home/myapp/ #Копіює файл "sample_app.py" з вашого проекту в каталог "/home/myapp/" в контейнері.

EXPOSE 8080 #Вказує, що контейнер буде слухати на порту 8080.

CMD python3 /home/myapp/sample_app.py #Задає команду, яка буде виконуватися при запуску контейнера. У цьому випадку, це запуск Python-сценарію "/home/myapp/sample_app.py", що є створеним Flask-додатком.

3.4 Висновки по розділу синтезу системи

Згідно до поставленої задачі кваліфікаційної роботи у даному розділі у повному об’ємі виходячи із потреб розгорнутої системи було проведено аналіз і розробка виділених ключових частин системи котрій потрібно звернути увагу для подальшої автономної роботи, та проведення якісного експериментального дослідження. Згідно вимог розділів було у повному об’ємі досліджено принципи побудови функціональних схем проекту. Обрані методи створення та використання сервера з віртуальною машиною та в подальшому на цій базі розгортати потрібну кількість тестованих контейнерів, було проведено проектування та практичне розгортання системи. На основі одержаних практичних показників було підібрано доцільне апаратне обладнання для подальшого впровадження.

Було отримано працюючі контейнери та перелік їх даних наведені на рисунку 3.16 та також було протестовано роботу і перевірку доступності веб– сервера в середині контейнера що приведено на рисунку 3.17.

4 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗГОРТАННЯ КОНТЕЙНЕРНОГО СЕРЕДОВИЩ

4.1 Призначення й сфера застосування програми

Дані програмні інструкції призначені для розгортання контейнера та розміщення програмного продукту в середині середовища створеного контейнера. В подальшому для розгортання в автоматному порядку ідентичних програмних застосунків студії після проведена попередньої збірки контейнерного середовища.

4.2 Обґрунтування технічних характеристик

4.2.1 Постановка завдання на розробку програми

Розроблювана інструкція – програма повинна виконувати збірку та створення контейнера та управління контейнерного середовища в середовищі Bash та за допомогою фреймворк Framework у середовищі Python для проведення розгортання та розробки веб–застосунків. Тож кінцева програма повинна бути написана у вигляді лінійного коду с поетапним виконанням розгортання сервера та контейнера з тестовано–експериментальним застосунком у середині. Також висунуте припущення, що завдяки алгоритмам програмного продукту Docker і його апаратним потребам розгортання він може себе характеризувати як більш швидкий та гнучкий у подальших модифікаціях порівняно с подібними продуктами.

4.2.2 Опис алгоритму і функціонування програми

Під час розгортання Docker контейнера програма використовує конфігураційний файл для визначення параметрів середовища. Заданий образ створюється як ізольоване середовище для додатка, забезпечуючи переносимість та цілісність. Docker налаштовує мережеві підключення та пробрасує порти для доступу до додатка. Кожен контейнер ізольований та має власний файловий простір, забезпечуючи ефективне та безпечне середовище

для додатків. Після розгортання веб-застосунок доступний через визначений мережевий порт, а Docker забезпечує інструменти для моніторингу та управління контейнерами, що робить процес розгортання ефективним та керованим.

Для функціонування програми в середовищі Docker потрібно пройти наступні етапи:

– створення коду інструкції для розгортання локального сервера;

– виконання збірки докер файла dockerfile;

– виконати збірку контейнера docker;

– та провести розгортання.

Після виконання розгортання одержується доступ до розгорнутого додатку та до консолі самого середовища для подальшої модифікації самого контейнера.

4.2.3 Опис і обґрунтування вибору методу організації вхідних і вихідних даних

Організація вхідних і вихідних даних є ключовою частиною розгортання контейнерів Docker і визначає, як система взаємодіє з оточуючим середовищем.

Організація вхідних даних полягає у монтажі томів (Volumes) використання Docker Volumes дозволяє зберігати вхідні дані в окремому томі, що монтується в контейнер. Це забезпечує стабільність даних і легкість обміну даними між контейнерами.

docker run -v /host/path:/container/path my_image

Використання томів дозволяє зберігати дані поза контейнером, забезпечуючи зручний доступ і збереження важливих даних при перезапуску контейнера.

Організація вихідних даних полягає у використанні зовнішніх сховищ або систем логування. Вихідні дані можуть бути направлені на зовнішні сховища або системи логування для аналізу та моніторингу. Це дозволяє легко

відслідковувати та аналізувати вихідні дані, надаючи попередження щодо працездатності контейнерів та додатків. Результати роботи додатків можуть бути збережені в Docker Volumes або зовнішніх сховищах для подальшого використання або аналізу.

Це дозволяє легко обмінюватися вихідними даними між контейнерами або забезпечувати постійний доступ до результатів роботи.

4.3 Опис розробленої програми

4.3.1 Загальні відомості

Створена програма являється невід'ємною частиною комплексу системи контейнеризації. Інструкції розгортання пишуться в середовищі Bash та Python і завантажуються у систему контейнеризації docker і відповідно с попереднім налаштуванням docker файл керування контейнерами.

4.3.2 Функціональне призначення

Розроблені програмні інструкції представляють собою повну реалізацію лінійного процесу розгортання контейнерів.

Інструкції використовуються для забезпечення функціонування та розгортання за функціонування контейнерних додатків. Програма виконується в серверному середовищі на базі Ubuntu Server із використанням середовища контейнеризації docker.

Програма надає можливість виконання таких операцій системи контейнеризації:

– підготовка контейнера для розгортання;

– підготовка локального сервера;

– підготовка docker файлам для збірки версії розгортання.

4.3.3 Опис логічної структури

Docker Daemon (Служба) демон Docker, що відповідає за управління контейнерами та ресурсами на сервері, слухає запити Docker API.

Docker Client (Клієнт) клієнт Docker, який служить інтерфейсом для взаємодії з Docker Daemon, дозволяючи користувачеві керувати контейнерами та ресурсами.

Docker Compose (Оркестрація) використовується для конфігурації та запуску багатоконтейнерних застосунків за допомогою YAML-файлів, спрощуючи процес оркестрації.

Docker Images (Образи), які містять код, залежності та конфігурацію веб-додатка, можуть бути створені вручну або автоматизовано за допомогою Dockerfile.

Docker Containers (Контейнери), робочі екземпляри образів, які містять веб-сервер, базу даних та інші компоненти веб-додатка.

Docker Network (Мережа) забезпечує засоби взаємодії між контейнерами та зовнішніми ресурсами для забезпечення комунікації.

Docker Registry (Реєстр), де зберігаються та обмінюються образами веб-додатка, може бути як публічний, так і приватний.

Інші Сервіси (тестований веб–додаток) Додаткові сервіси, такі як веб–додатки, можуть бути окремо запущені як контейнери.

Ця архітектура дозволяє ефективно управляти та масштабувати веб-додаток за допомогою Docker, забезпечуючи ізоляцію та переносимість середовища виконання. Загальне відображення архітектури реалізації Docker продемонстровано нижче у рискну 4.1.

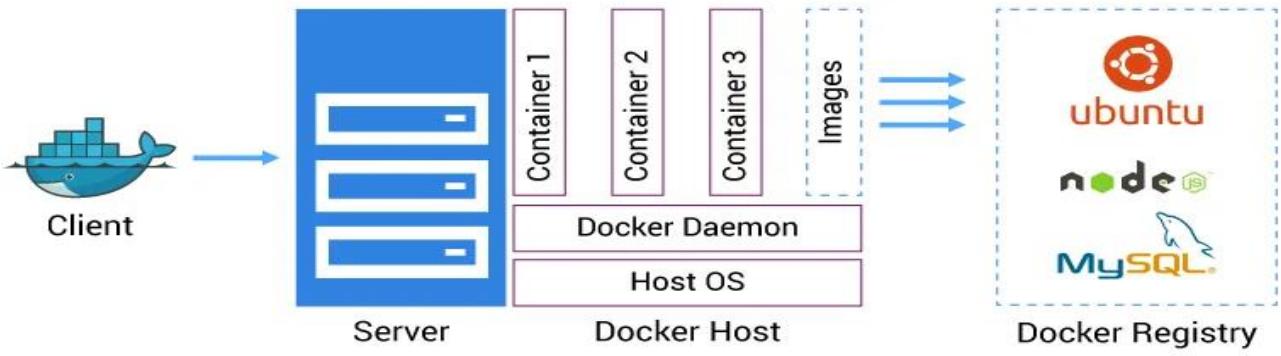


Рисунок 4.1 – Демонстрація обраної архітектури реалізації контейнерного середовища Docker

4.3.4 Використовувані технічні засоби

Обрання даного сервера Cisco UCS-SPR-C240M4-P1 так як ігрова студія працює над великими проектами з високими потребами в ресурсах. Пам'ять (256 Гб), обчислювальна потужність (2 процесори Intel Xeon E5-2696 V4) і простір на жорсткому диску (100 Тв в SSD носіях) важливі для забезпечення швидкості та продуктивності контейнерів та урахуванням того що ще буде рейд масив для надійності даних.

Мережева інфраструктура забезпечення надійної та швидкої мережевої інфраструктури важливе для Ігорових продуктів. Було розглянуто можливість використання високошвидкісних комутаторів і маршрутизатор для забезпечення мінімального лагу для користувачі то використовуються пограничний маршрутизатор марки Cisco 2811 ISR 4 він має такі характеристики пам'ять: RAM 512 Мб; флеш пам'ять 256 Мб; інтерфейси: 2 порти 100Base-TX / 1000Base-T, роз'єм RJ-45, 1 консольний порт управління, роз'єм RJ-45, 2 слоти HWIC, 1 порт USB 4-пін USB тип A; ОС базова Cisco IOS IP Base.

4.4 Загальні відомості

Позначення і найменування програми docker представляє собою відкрите програмне забезпечення, яке забезпечує автоматизоване розгортання, управління та ізоляцію застосунків у контейнерах. Воно дозволяє запускати програми в уніфікованому середовищі, незалежно від налаштувань хост–системи.

Програмне забезпечення для належної роботи docker необхідно встановити docker engine, що включає docker daemon та docker client. Також можна користуватися інструментами, такими як docker compose для оркестрації багатоконтейнерних додатків, і docker swarm чи kubernetes для управління контейнерами у кластері.

Мови Програмування для інструкції Dockerfile, які використовуються для побудови контейнерів, зазвичай написані мовою специфікації Docker DSL

(Domain Specific Language). Це власна мова для опису конфігурації та зіставлення лінійної інструкції для розгортання. Та для використання зовнішньої інтеграції використовується код на основі Bash.

5 ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ

В цьому розділі проводилося експериментальне дослідження, для встановлення якісного використання ресурсів апаратної частини та затраченого часу на розгортання одного та декількох контейнерних середовищ. Це допомагає уникнути помилок і забезпечити, щоб програмні продукти відповідали потребам користувачів. Для цього були розроблена тестова модель, яка представляє вимоги у вигляді експериментального веб–додатку який розміщується на середовищі із можливістю зовнішнього користування. Ці моделі можливо відслідкувати кореляцію якості та часу затрат і ефективності розгортання та паралельного розгортання нових контейнерів у різних середовищах контейнеризації. Також були розроблені технології, які накопичують, аналізують та управляють даними про вимоги. Ці технології допомагають відстежувати зміни в вимогах та забезпечувати їхню цілісність. Цей підхід можна використовувати лише для одного контрено трестованого продукту. Він не може застосовуватися для аналізу вимог у різних продуктах так як буде похибки котрі потрібно закладавати у розрахунок в самому початку.

5.1 Сутність експерименту (мета, умови)

Мета цього дослідження – розробити підхід, який можна використовувати для аналізу вимог у різних продуктах на етапах проходження створення, масштабування та розгортання контейнерного середовища. Для цього будуть використовуватися альтернативні сучасні технології для контейнеризації застосунків.

Для досягнення об’єктивних результатів відповідно до проведеного аналізу технологій було обрано Docker та Vagrant так як вони дуже схожі за призначенням використання і майже однакові у методі створення контейнерів. Наступне що було використано так це один і той самий тестовий додаток с метою розгортання веб–сервера та з веб–ресурсом що забезпечив найбільш

достовірний результати без значних похибок зі сторони створених контейнерних середовищ.

Це означає, що необхідно розробити метрики, які можна використовувати для оцінки того, наскільки добре альтернативні технології контейнеризації та віртуалізації проектів дозволяють розгортати програмні продукти [2].

5.2 Результат експерименту

Результатом проведення аналітично – експериментальних досліджень з використанням Docker та Vagrant для реалізації розгортання тестового застосунку потрібно дослідити наступні якісні показники [2]:

– адаптованість (Portability – Prb);

– оброблюваність (Performance – Prm).

Адаптованість визначається тим, наскільки легко можна перенести відповідне програмне забезпечення з однієї операційної платформи на іншу. Іншими словами, це може бути виміряно як величина, яка обернено пропорційна часовим та ресурсним витратам, необхідним для здійснення такої переносу. Для обрахування даного параметра Prb було використано таку формулу

$$Prb = \frac{1}{\mathrm{T}(Deployment)}, \tag{5.1}$$

де $\mathrm{T}(Deployment)$ – це потрібний час на створення та розгортання проекту

З урахуванням окремих характеристик процесу контейнеризації, розрахунок з використанням формули (1) включає в себе визначення двох видів витрат часу: часу, необхідного для розгортання операційного середовища (контейнера) «T(Environment deployment)», і часу, необхідного для налаштування та підготовки до розгортання сервера додатків «T(Config. and preparation)». У свою чергу, час «T(Environment deployment)» складається

з двох частин: часу створення (білда) контейнера чи віртуальної машини та часу їх безпосереднього виконання (рану).

Отже, у випадку використання технології Docker, часові витрати на процес розгортання виглядають наступним чином [2]:

$$T\ (Environment\ deployment)_{Docker} = T(Docker\ build) + T(Docker\ run) \quad (5.2)$$

у випадку використання технології Vagrant, розрахунок виконується за наступною формулою:

$$T\ (Environment\ deployment)_{Vagrant} = T(Vagrant\ setup) \quad (5.3)$$

де «T(Vagrant setup)» – це виділений час потрібний для запуску та розгортання Vagrant машини.

Таким чином, враховуючи вирази (5.1) – (5.3), остаточний вираз для розрахунку метрики оцінки показника якості адаптованості (Prb) представлений наступною формулою [2]:

$$Prb = \frac{1}{T((Env\ deployment))\ +\ T((Config.and\ preparation))} \quad (5.4)$$

Для оцінки загальної оброблюваності в ході розгортання в цьому дослідженні використовуються такі критичні параметри:

– метрика для визначення ступеня використання центрального процесора (M(CPU)).

– метрика для визначення обсягу оперативної пам'яті (M(RAM)).

– метрика для визначення середнього навантаження операційної системи (M(LOADAVG)).

Метрика M(CPU) визначається як середнє навантаження центрального процесора, викликане виконанням конкретних дій.

$$M(CPU) = 1 - \frac{\sum_{i=1}^{N}(Uavg(CPU)i)}{N} \quad (5.5)$$

де $Uavg(CPU)$ – ця метрика представляє собою середнє використання центрального процесора, яке виникає під час виконання кожної окремої операції, де N – кількість операцій, що виконані.

Метрика M(RAM) допомагає визначити потрібний обсяг оперативної пам'яті для розгортання компонентного програмного рішення (КПР) за допомогою одного з альтернативних методів, таких як Docker чи Vagrant, і вона представляється наступним чином [2]:

$$M(\text{RAM}) = \frac{1}{V_2(RAM) - V_1(RAM)} \quad (5.6)$$

Тут V_1(RAM) представляє об'єм оперативної пам'яті, використаної до початку процесу розгортання відповідного компонентного програмного рішення (КПР), і V_2(RAM) – об'єм оперативної пам'яті, що використовується після завершення процесу розгортання.

Також може бути визначена метрика середнього завантаження операційної системи M(Load Avg) під час процесу розгортання КПР, і ця метрика буде визначатися наступним чином [2]:

$$M(Load\ Avg) = \frac{1}{L_2 - L_1} \quad (5.7)$$

де L_1 – це середнє навантаження системи перед початком процесу розгортання відповідного КПР, а L_2 – середнє навантаження після завершення розгортання [2].

Для розрахунку остаточного значення метрики Prm на основі окремих метрик (5) – (7), пропонується об'єднати їх у вигляді:

$$Prm = k_1 M(CPU) + k_2 M(RAM) + k_3 M(load\ Avg) \quad (5.8)$$

Вагові коефіцієнти k_1, k_2, k_3 можуть бути визначені з допомогою експертного підходу, такого як метод аналізу ієрархій (MAI).

У Таблиці 5.1 наведені результати цих експериментів для обох технологій: Docker і Vagrant. Тут T1 та T2 позначають час початку та закінчення експерименту відповідно, і вони вказані в Unix–часі який було одержано за допомогою команди data +%s. ΔT вказує на час, який був витрачений на розгортання застосунку у відповідному контейнер. Відповідно для більш демонстраційного розуміння як проходив експеримент приведено рисунок 5.1 який відображає послідовність дій експериментів.

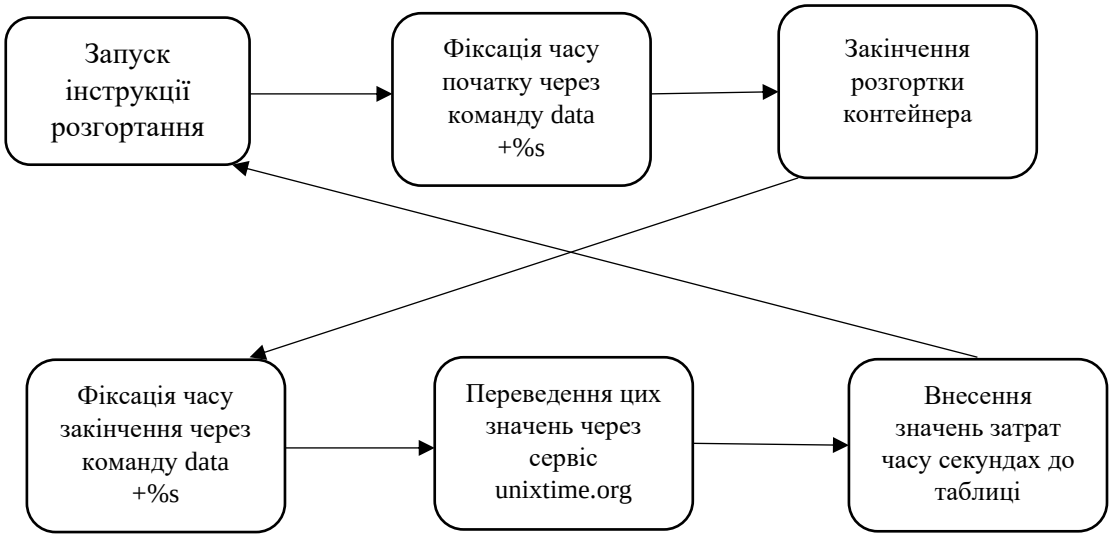


Рисунок 5.1 – Алгоритм проведення експериментів для встановлення затрат часу на розгортання контейнерного середовища

Таблиця 5.1 – Визначення часу на створення та розгортання проекту з використанням Docker та Vagrant у секундах.

№	T1(Docker), c	T2(Docker), c	ΔT (Docker), c	T1(Vagrant), c	T2(Vagrant), c	ΔT (Vagrant), c
1	1696485004	1696485262	253	1696489505	1696489850	327
2	1696485315	1696485560	243	1696489880	1696490180	289
3	1696485850	1696486100	237	1696490282	1696490600	311
4	1696486145	1696486430	267	1696490718	1696491373	333
5	1696486443	1696487351	307	1696491610	1696491915	279
Середній час Docker			261	Середній час Vagramt		290

З використанням інформації з таблиці 5.1 та формул (5.1) – (5.4) були визначені результати для оцінки ефективності процесу створення контейнерів тестового КПР за допомогою метрики Prb.

$$Prb\ (Docker) = 0.0038;\ Prb\ (Vagrant) = 0.0035 \quad (5.9)$$

Отже, можна відзначити, що значення метрики переносимості Prb при використанні технології Docker перевищує значення цієї ж метрики при

використанні технології Vagrant лише на приблизно 8%, що є досить невеликим відхиленням.

Експеримент для визначення значення метрики продуктивності Prm був проведений за наступною методикою: було виконано 10 запитів на обробку даних за допомогою функціоналу тестового КПР, зібрані відповідні дані, і потім були розраховані показники метрик з використанням формул (5.5) – (5.7). Отримані результати представлені у таблиці 5.2.

Таблиця 5.2 – Демонстрація результатів експерименту

Метрика / Технології	Docker	Vagrant
M(CPU)	0.63	0.56
M(RAM)	0.00049	0.00041
M(Load AVG)	0.14	0.083

Наступним кроком для визначення остаточного значення метрики продуктивності Prm для варіантів використання Docker та Vagrant є використання методу аналізу ієрархій (MAI) згідно з формулою (5.8). Для цього спочатку була створена матриця попарних порівнянь пріоритетів для альтернатив Docker / Vagrant продемонстровано в таблиця 5.3.

Таблиця 5.3 – Порівняння з методом MAI

	M(Load AVG)	M(RAM)	M(CPU)
M(Load AVG)	1	3/1	5/4
M(RAM)	1/3	1	3/6
M(CPU)	4/5	6/3	1

Далі, наступним кроком є визначення значень вагових коефіцієнтів k1, k2, k3 для використання в формулі (5.8) детальні продемонстровано в таблиці 5.4.

Таблиця 5.4 – Вагові значення для MAI

	M(Load AVG)	M(RAM)	M(CPU)	Сума по рядкам	Вагові коефіцієнти k1 , k2 , k3
M(Load AVG)	1	3	1,25	5,25	0,48
M(RAM)	0,33	1	0,5	1,83	0,17
M(CPU)	0,8	2	1	3,8	0,35

Використовуючи результати продуктивності які наведені в таблиці 4.3, вагових значень з таблиці 5.4 та використовуючи формулу (5.8), було отримано порівняльні результати оцінки для Prm до трестованих технологій Docker і Vagrant відповідно.

$$\begin{aligned} Prb\ (Docker) &= 0.48 \cdot 0.14 + 0.17 \cdot 0.00049 + 0.35 \cdot 0.63 \approx 0.29 \\ Prb\ (Vagrant) &= 0.48 \cdot 0.083 + 0.17 \cdot 0.00041 + 0.35 \cdot 0.56 \approx 0.24 \end{aligned} \quad (5.10)$$

Отже, аналізуючи результати метрики Prm з виразу (5.9), можна зробити обґрунтований висновок, що продуктивність функціонування тестового компонентного програмного рішення (КПР), який було розгорнуто за допомогою контейнерної технології Docker, що є приблизно на 17% вищою, ніж у випадку його розгортання з використанням технології віртуалізації Vagrant [2].

5.3 Аналіз відповідності теоретичних та експериментальних досліджень

У відповідності с теоретично виведеними та розглянутими гіпотезами котрі пов’язані з такими відсліджуваними параметрами як адаптивність та продуктивністю обраних середовищ контейнерного розгортання програмних продуктів, а саме Docker та Vagrant було проведено теоретичне очікування роботи цих моделей. Ці моделі були в подальшому реалізовані в експериментальній частині у вигляді створених контейнерів у віртуальному

середовищі котре імітувало задані параметри у теоретичному досліджені, а саме образ серверної системи та апаратних характеристик котрі максимально наближені до розгодованих у теоретичній частині. Згідно отриманих результатів в практичній реалізації було підтверджено теоретичне дослідження котре демонструвало приблизно на 15–17% ефективності роботи та використаних ефективних ресурсів і швидкості розгортання контейнерів для тестового додатку на технології Docker, котра демонструє хоч і незначну але критичну перевагу у разі впровадження в повній мірі в систему ігрової судії на технологією Vagrant котра показала себе незначно але все одно менше ефективною у роботі с додатками котрі випускає дане підприємство.

5.4 Характеристика новизни результатів

Обґрунтовано та згідно з результатами проведених експериментів, контейнеризація за допомогою Docker виявляє значну перевагу з точки зору переносимості. У відношенні до аналізу продуктивності було розраховано кількісні значення, порівнюючи різні критерії з використанням методу аналізу ієрархій, і отримані результати також підтвердили перевагу Docker контейнеризації [5].

Відповідно до одержаних результатів у ході виконання дослідження було виявлено, що технології Docker контейнерів більш ефективні у разі їх використання невеликими проектами, що в свою чергу надає новий погляд для прикладного використання Docker в ігрових студіях. Які потребують універсального та максимальної ефективності використання апаратної та програмної інфраструктури. Можливе використання даної розробки позитивним чином позначиться на якості роботи та доступності зовнішніх користувачів. Відповідно, так як час затрачений для розгортання та модифікації створених контейнерів у разі потреби буде затрачено менше на 17%, та легко реалізуємо у разі потреби в майбутньому при клонуванні.

Це вказує на те, що використання Docker дозволяє оптимізувати ресурси та прискорює роботу додатків порівняно із традиційними методами віртуалізації, такими як віртуальні машини або подібні контейнери.

ВИСНОВКИ

Кваліфікаційна робота є завершеною науковою роботою, в якій вирішена науково–практична задача аналізу, порівняння, дослідження та обґрунтування використання контейнерних середовищ та практичне використання та покриття більшості потреб для впровадження у структуру Ігорової студії та забезпечення використання автоматизованої схеми розгортання контейнерних середовищ для релізних проектів котрі будуть доступні для кінцевих користувачів.

Основні висновки і результати роботи полягають у наступному:

1. Проведено аналіз та дослідження наявних інструментів та методів контейнеризації та контейнерної обробки і було розглянуто в роботі інструменти і технології котрі підтримують часткову чи повну автоматизацію з найбільш поширеними мовами такими як Bash та Ansible так як вони відносно прості в інтеграції та найбільш задовольняють потреби відібраних інструментів.

2. В результаті дослідження різних моделей віртуалізації була обрана Docker, оскільки вона вражає своєю гнучкістю та ефективністю. Модель роботи Docker дозволяє ефективно розгортати безліч контейнерів на одній віртуальній машині чи фізичному сервері, забезпечуючи велику ступінь ізоляції між додатками та сервісами. Це особливо корисно для проектів, де потрібна гнучкість у додаванні та зміні окремих додатків та служб у майбутньому. Аналіз та перевірка теорій, висунутих під час дослідження, підтвердили високу продуктивність та ефективність Docker у реалізації зазначених завдань.

3. Практично реалізовано та протестовано систему контейнерного розгортання на імітаційній машині, використовуючи емулятор сервера Ubuntu та автоматизоване створення контейнерного середовища. Це було зроблено для проведення експерименту та збору даних, необхідних для розділу роботи, що включає порівняння ефективності різних систем контейнеризації.

4. Були встановлені та ретельно вивчені теоретичні та практичні переваги Docker у порівнянні з аналогічними програмами для контейнерного розгортання. Це дозволило розробити методику для проведення оцінки ефективності Docker в порівнянні з конкуруючим продуктом – Vagrant. Експериментальне дослідження підтвердило теоретичні розрахунки стосовно вигідності Docker при розгортанні веб-застосунків з використанням моделі обслуговування зовнішніх користувачів. У порівнянні з конкурентами, Docker відзначився як поширена та досить гнучка середа контейнеризації. Його легкість використання та можливості інтеграції з додатковими бібліотеками стануть додатковою перевагою для подальших розробок та розширення параметрів інтелектуального портфеля продуктів компанії.

5. Обґрунтування використання Docker у геймінг-студії "Ubisoft Ukraine (Kyiv)" є повністю обґрунтованим та відповідає визначеним вимогам. У контексті розробки додатків та подібних проектів, які випускає та реалізує підприємство, використання Docker є доречним та ефективним. Застосування Docker дозволить ефективно управляти та ізолювати різні частини проектів, забезпечуючи при цьому високу гнучкість та масштабованість.

Зокрема, у теоретичному та експериментальному дослідженні встановлено, що Docker продемонстрував результати роботи, які на 15–17% перевищують ефективність порівнянних з ним рішень. Це вказує на його значний потенціал для оптимізації розгортання та управління контейнерами у великих та складних проектах, які характерні для геймінг-індустрії.

6. Реальне використання отриманих результатів відкриває можливість для успішного впровадження та реалізації подібних проектів у інших геймінг-студіях, які мають аналогічні вимоги до серверної контейнеризації та масштабування системи. Розроблена система контейнерної розгортки дозволяє зекономити кошти та підвищити продуктивність розробки та тестування продуктів, полегшуючи інтеграцію нововведень та розширення потужностей для забезпечення ефективного обслуговування великої кількості зовнішніх користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Методичні рекомендації до виконання кваліфікаційної роботи магістра студентами галузі знань 12 інформаційні технології спеціальності 123 комп'ютерна інженерія/ Л.І. Цвіркун, В.В. Гнатушенко, С.М. Ткаченко. – Д.М.: НТУ «Дніпровська політехніка», 2022. – 46 с.

2. Мережева академія Cisco курс DevNet: [Електронний ресурс] – Режим доступа: URL: https://www.netacad.com.

3. Джон Купер Архітектура комп'ютерних мереж: [Навчальний посібник] / Netskills Ver 1.0, 2014. – 265 с.

4. Автоматизація DevOps: [Електронний ресурс] – Режим доступа: URL: https://habr.com/ru/companies/ruvds/articles/525840/.

5. DevOps: Призначення та Основні Функції: [Електронний ресурс] – Режим доступа: URL: https://buki.com.ua/blogs/devops-priznacennya-ta-osnovni-funkciyi.

6. Десять інструментів DevOps: [Електронний ресурс] – Режим доступа: URL: https://nt.ua/blog/10-devops-tools.

7. Cisco DevNet як платформа для навчання, можливості для програмістів: [Електронний ресурс] – Режим доступа: URL: https://dou.ua/forums/topic/26036/.

8. Витрати на інформаційні технології (IT) на корпоративне програмне забезпечення в усьому світі з 2009 по 2023 рік: [Електронний ресурс] – Режим доступа: URL: https://www.statista.com/statistics/203428/total-enterprise-software-revenue-forecast/.

9. Ступінь впровадження DevOps розробниками програмного забезпечення в усьому світі в 2017 і 2018 роках: [Електронний ресурс] – Режим доступа: URL: https://www.statista.com/statistics/673505/worldwide-software-development-survey-devops-adoption/.

10. Еволюція DevOps і впровадження пропозицій самообслуговування на внутрішніх платформах у всьому світі у 2020 році за етапами еволюції:

[Електронний ресурс] – Режим доступа: URL: https://www.statista.com/statistics/1229785/devops-evolution-self-service-adoption/.

11. Використовуємо Docker для розгортання web–додатка bitrix: [Електронний ресурс] – Режим доступа: URL: https://avivi.pro/ua/blog/vikoristovu-mo-docker-dlya-rozgortannya-web-dodatka-bitrix/.

12. 20 корисних команд для роботи з Docker в Linux: [Електронний ресурс] – Режим доступа: URL: https://blog.iteducenter.ua/ratings/docker_linux/.

13. Вивчаємо Docker, частина 3: файли Dockerfile: [Електронний ресурс] – Режим доступа: URL: https://habr.com/ru/companies/ruvds/articles/439980/.

Додаток А

Програмне забезпечення налаштування системи розгортання контейнерних середовищ

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ НАЛАШТУВАННЯ КОНТЕЙНЕРНОГО СЕРЕДОВИЩА

Текст програми

804.02070743.23007-01 12 01

Листів 9

2023

АНОТАЦІЯ

Дана програма містить в собі програмний код для налаштування компонентів комп’ютерної системи та системи Docker на базі Ubuntu сервера. Програма призначена для реалізації налаштування трестованого продукту (додатку імітаційного) для подальшого розгортання у системі контейнерів Docker, налаштування Docker файл с заданими структури проекту, Docker файлом системи розгортання, автоматизованими інструкціями котрі несуть призначення для розгортання подібних машин та динамічного розширення шляхом розвантаження потоків на паралельні контейнери

ЗМІСТ

1. Налаштування контейнерного середовища 4
1.1 Налаштування розгортання тестового веб-додатка 4
1.2 Налаштування докер файла для створення контейнера 4
1.3 Налаштування файл трестованого застосунку 4
1.4 Налаштування коду програми автоматизації 5

1 Налаштування контейнерного середовища

1.1 Налаштування розгортання тестового веб-додатка

```
from flask import Flask
from flask import request
from flask import render_template

sample = Flask(__name__)

@sample.route("/")
def main():
    return render_template("index.html")

if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=8080)
```

1.2 Налаштування докер файлам для створення контейнера

```
FROM python
RUN pip install flask
COPY ./static /home/myapp/static/
COPY ./templates /home/myapp/templates/
COPY sample_app.py /home/myapp/
EXPOSE 8080
CMD python3 /home/myapp/sample_app.py
```

1.3 Налаштування файл трестованого застосунку

```
<html>
<head>
```

```
    <title>Ubisoft Kiev market</title>
    <link rel="stylesheet" href="/static/style.css" />
</head>
<body>
    <h1>you contacted me from {{request.remote_addr}}</h1>
</body>
</html>
```

1.4 Налаштування коду програми автоматизації

```
mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static

cp sample_app.py tempdir/.
cp -r templates/* tempdir/templates/.
cp -r static/* tempdir/static/.

echo "FROM python" >> tempdir/Dockerfile
echo "RUN pip install flask" >> tempdir/Dockerfile

echo "COPY ./static /home/myapp/static/" >> tempdir/Dockerfile
echo "COPY ./templates /home/myapp/templates/" >> tempdir/Dockerfile
echo "COPY sample_app.py /home/myapp/" >> tempdir/Dockerfile


echo "EXPOSE 8080" >> tempdir/Dockerfile
echo "CMD python3 /home/myapp/sample_app.py" >> tempdir/Dockerfile
```

cd tempdir

docker build -t sampleapp .

docker run -t -d -p 8080:8080 --name samplerunning sampleapp

docker ps –a