НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут телекомунікаційних систем

Кафедра телекомунікацій

«На правах рукопису»

УДК _______________

«До захисту допущено»

Завідувач кафедри

__________ Сергій КРАВЧУК

«___»_____________2022 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-професійною програмою «Інженерія та програмування інфокомунікацій»
зі спеціальності 172 «Телекомунікації та радіотехніка»

на тему: «Визначення оптимального типу мережі Docker»

Виконав:
студент ІІ курсу, групи ТЗ-11мп
Казарін Юрій Вікторович __________

Керівник:
Старший викладач кафедри ТК НН ІТС, к.т.н.
Маньківський В.Б. __________

Рецензент:
Доцент кафедри ІКТС НН ІТС, к.т.н., доцент
Созонник Г.Д. __________

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____________

Київ – 2022 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут телекомунікаційних систем
Кафедра телекомунікацій

Рівень вищої освіти – другий (магістерський)

Спеціальність – 172 «Телекомунікації та радіотехніка»

Освітньо-професійна програма «Інженерія та програмування інфокомунікацій»

ЗАТВЕРДЖУЮ

Завідувач кафедри

__________ Сергій КРАВЧУК

«___»_____________2022 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Казаріну Юрію Вікторовичу

1. Тема дисертації «Визначення оптимального типу мережі Docker», науковий керівник дисертації Маньківський Володимир Броніславович, к.т.н., затверджені наказом по університету від «28» жовтня 2022 р. № 3995-с.

2. Термін подання студентом дисертації 12.12.2022 р.

3. Об'єкт дослідження: Docker-технологія.

4. Предмет дослідження: використання та тестування Docker-мереж.

5. Перелік завдань, які потрібно розробити: здійснити літературний та теоретичний огляд, розкрити поняття, Docker-технології, контейнеризації та мережі Docker, поставити задачу на дослідження, побудувати мережі Docker з контейнерами і навести їх характеристики, продемонструвати готову мережу з контейнерами і налаштуваннями, протестувати мережі, зробити висновки.

6. Орієнтовний перелік ілюстративного матеріалу: тема, мета, актуальність і задачі досліджень, аналіз мереж Docker, результат дослідження, висковки по роботі.

7. Дата видачі завдання “ 15 ” жовтня 2021 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Обрання теми магістерської дисертації	01.11.2021-25.01.2022	виконано
2	Збір та аналіз інформації	01.02.2022-28.02.2022	виконано
3	Оформлення вимог	01.03.2022-30.04.2022	виконано
4	Огляд та аналіз предметної області	01.05.2022-30.06.2022	виконано
5	Початок роботи з середовищем	01.07.2022-30.07.2022	виконано
6	Знайомство з основними особливостями середовища Docker	01.08.2022-28.08.2022	виконано
7	Тестова побудова контейнерів та мереж	01.09.2022-30.09.2022	виконано
8	Дослідження мереж	01.10.2022-07.11.2022	виконано
9	Тестування та опис дослідження	10.11.2022-03.12.2022	виконано
10	Оформлення розділу стартап-проєкту	05.12.2022-10.12.2022	виконано
11	Оформлення і захист роботи	12.12.2022	виконано

Студент Юрій КАЗАРІН

Науковий керівник дисертації Володимир МАНЬКІВСЬКИЙ

РЕФЕРАТ

Робота містить 81 сторінку, 47 рисунків, 25 таблиць, 13 джерел.

Актуальність теми полягає в тому, що Docker – відносно нова та ще не зовсім вивчена технологія. Цій технології лише 14 років і постійно відкриваються все нові можливості, які дає нам робота з Docker та технологією контейнеризації. На сьогодні це в основному контейнерування ПЗ, робота з ПЗ поза ядром, тобто елемент віртуалізації, об'єднання певних контейнерів та процесів в мережі. Побудова мережі за допомогою цієї технології та винесення роботи за межі ядра допомогає у вирішенні певного спектру проблем, наприклад втручання сторонніх процесів (таких, що знаходяться поза певними типами мережі) у роботу системи. Саме дослідження полягає у пошуку оптимального типу мереж для базових задач організації роботи мережі та контейнерів в ній, оцінюватиметься оптимальність тестуванням навантаження в мережі, характеристиками та простотою використання.

Метою роботи є вивчити середовище Docker Desktop на Windows, побудувати системи мереж та контейнерів Docker та порівняти їх, визначивши оптимальний тип мережі. Для досягнення мети можна поставити такі задачі:

- побудувати мережі Docker з контейнерами і навести їх характеристики;
- продемонструвати готові мережі з контейнерами і налаштуваннями;
- протестувати мережі;
- порівняти результати тестування та визначити оптимальний тип мережі.

Об'єкт дослідження: Docker-технологія.

Предмет дослідження: використання та тестування Docker-мереж.

Наукова новизна одержаних результатів полягає в дослідженні різних типів мереж Docker в середовищі Windows Docker Desktop, не використовуючи віртуальну машину та ОС Linux за певними показниками, основним з яких є робота мережі під навантаженням. Уперше проведено порівняльні експерименти

всередині одного середовища, дістало подальший розвиток уявлення про оптимальність різних типів мережі.

Практичне значення одержаних результатів полягає в наочному тестуванні та порівнянні систем з різними типами мереж Docker в однакових умовах і визначення оптимального типу мережі з точним визначенням по меті дослідження та рекомендації щодо подальшого впровадження.

Ключові слова: Docker, контейнер, контейнеризація, мережа, тип мережі.

ABSTRACT

The work contains 81 pages, 47 figures, 25 tables, 13 sources.

The relevance of the topic is that Docker is a relatively new and not yet fully explored technology. This technology is only 14 years old, and new opportunities are constantly opening up that work with Docker and containerization technology gives us. Today, this is mainly software containerization, working with software outside the core, that is, an element of virtualization, combining certain containers and processes in the network. Building a network using this technology and moving the work outside the kernel helps to solve a certain range of problems, such as the interference of third-party processes (such as those outside certain types of networks) in the operation of the system. The research itself consists in finding the optimal type of networks for the basic tasks of organizing the operation of the network and containers in it, optimality will be evaluated by testing the network load, characteristics and ease of use.

The goal of the work is to build systems of networks and Docker containers and compare them, determining the optimal type of network. To achieve the goal, you can set the following tasks:

- build Docker networks with containers and list their characteristics;
- demonstrate ready-made networks with containers and settings;
- test networks;
- compare test results and determine the optimal type of network.

Research object: Docker technology.

Research subject: use and testing of Docker networks.

The scientific novelty of the obtained results lies in the study of different types of Docker networks in the Windows Docker Desktop environment, without using a virtual machine and Linux OS according to certain indicators, the main of which is the operation of the network under load. For the first time, comparative experiments were conducted within the same environment, and the concept of the optimality of different types of networks was further developed.

The practical value of the obtained results consists in the visual testing and comparison of systems with different types of Docker networks under the same conditions and the determination of the optimal type of network with an accurate definition for the purpose of research and recommendations for further implementation.

Keywords: Docker, container, containerization, network, network type.

ЗМІСТ

ВСТУП 10

РОЗДІЛ 1 12

ТЕОРЕТИЧНІ ВІДОМОСТІ 12

1.1. Контейнеризація 12

1.2. Відомості про Docker 14

1.3. Docker engine 24

1.4. Мережі Docker. 30

РОЗДІЛ 2 34

ДОСЛІДЖЕННЯ 34

2.1. Першочергові пояснення 34

2.2. Налаштування контейнерів Docker 34

2.3. Налаштування та тестування мереж. 37

РОЗДІЛ 3. 61

РОЗРОБКА СТАРТАП-ПРОЄКТУ 61

3.1. Інформаційна карта проєкту. 61

3.2. Формування команди стартапу. 64

3.3. Розроблення ринкової стратегії проєкту. 66

3.4. Аналіз ринкових можливостей 69

3.5. Реалізація стартап-проєкту 74

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ 80

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 81

ПЕРЕЛІК СКОРОЧЕНЬ

API	Application Programming Interface
CLI	Common Language Infrastructure
DTR	Docker Trusted Registry
ОС	Операційна система
CPU	Central Processing Unit

ВСТУП

Актуальність теми полягає в тому, що Docker – відносно нова та ще не зовсім вивчена технологія. Цій технології лише 14 років і постійно відкриваються все нові можливості, які дає нам робота з Docker та технологією контейнеризації. На сьогодні це в основному контейнерування ПЗ, робота з ПЗ поза ядром, тобто елемент віртуалізації, об’єднання певних контейнерів та процесів в мережі. Побудова мережі за допомогою цієї технології та винесення роботи за межі ядра допомагає у вирішенні певного спектру проблем, наприклад втручання сторонніх процесів (таких, що знаходяться поза певними типами мережі) у роботу системи. Саме дослідження полягає у пошуку оптимального типу мереж для базових задач організації роботи мережі та контейнерів в ній, оцінюватиметься оптимальність тестуванням навантаження в мережі, характеристиками та простотою використання.

Дисертація була підготовлена відповідно до науково-дослідного плану кафедри телекомунікацій» Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського.

Метою роботи є вивчити середовище Docker Desktop на Windows, побудувати системи мереж та контейнерів Docker та порівняти їх, визначивши оптимальний тип мережі.

Об’єкт дослідження – Docker-технологія.

Предмет дослідження – використання та тестування Docker-мереж.

Були проведені такі методи дослідження: теоретичний науковий метод (опис явищ, роботи мережі, теоретичне моделювання) та емпіричний науковий метод (експеримент, наукове дослідження та спостереження.)

Наукова новизна одержаних результатів полягає в дослідженні різних типів мереж Docker в середовищі Windows Docker Desktop, не використовуючи віртуальну машину та ОС Linux за певними показниками, основним з яких є робота мережі під навантаженням. Уперше проведено порівняльні експерименти

всередині одного середовища, дістало подальший розвиток уявлення про оптимальність різних типів мережі.

Практичне значення одержаних результатів полягає в наочному тестуванні та порівнянні систем з різними типами мереж Docker в однакових умовах і визначення оптимального типу мережі з точним визначенням по меті дослідження та рекомендації щодо подальшого впровадження.

Дана дисертація містить 3 розділи, 81 сторінку загального обсягу (з них 71 сторінка основного тексту), 47 рисунків та 13 джерел.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1. Контейнеризація

Технологія контейнеризації - це ще одна форма віртуалізації ОС, що пропонує ізоляцію додатків в просторах користувача (контейнерах). Всі контейнери використовують ту саму операційну систему. Завдяки технології контейнеризації можна запускати додаток із потрібними бібліотеками у типовому контейнері, який з'єднується з хостом або іншою зовнішньою компонентою за допомогою простого інтерфейсу[1].

Усі компоненти, необхідні для роботи програми (код, середовище запуску, системні інструменти, бібліотеки та налаштування), упаковуються в один образ і можуть бути використані повторно в рамках поточного завдання або будь-яких інших. Контейнер незалежний від ресурсів та архітектури хоста. Він створює ізольоване середовище для програми, не використовуючи CPU, RAM або сховища хостової ОС. Усі процеси йдуть усередині.

Навіть кілька контейнерів використовують ядро однієї хостової ОС. Дозволяє створювати одному комп'ютері лише однорідні обчислювальні середовища. Набагато легше ВМ, розмір вимірюється Мб. Здатний запускатися майже миттєво.

Під різні завдання вибираються різні методи віртуалізації.

Для чого потрібні контейнери? Це зручне рішення для тестування та розробки. Коли розробник запускає свій чи чийсь код у тестовому середовищі, можуть виникати помилки через зміну середовища програми. Топологія мережі, політики безпеки, обчислювальні ресурси можуть впливати на роботу програми. Контейнер самодостатній та легко перестворюється, якщо потрібно відкотити зміни або провести ще одне тестування.

Що таке контейнеризація з погляду використовуваних технологій? Для створення контейнерів використовуються технології, як Linux XC, OpenVZ, Linux VServer, BSD Jails і Solaris. Перша популярна технологія контейнеризації в Linux - це OpenVZ, що пізніше перетворилася на більш досконалий комерційний продукт Virtuozzo.

Плюси контейнеризації:

Швидкість створення. Контейнер можна створити швидше, ніж ВМ. При цьому середовище контейнеризації деяких завдань дає більше можливостей.

Економічність. Контейнер займає менше місця у сховищі, що зменшує накладні витрати.

Висока продуктивність. Відсутність міжмережевих залежностей та конфліктів підвищує продуктивність розробки. Кожен контейнер фактично є мікросервісом, який можна незалежно оновлювати, не задаючись питанням синхронізації.

Управління версіями. Моніторити версійність контейнерів, стежити за відмінностями між ними.

Можливість міграції середовища обчислень. Усі залежності додатків та ОС, необхідних роботи програми, інкапсулюються. Це дозволяє легко переносити образ контейнера з одного середовища в інше. Так, один образ можна запускати серед Windows і Linux або dev/test/stage.

Стандартизація. Як правило, контейнери створюються на основі відкритих стандартів. Тому з ними можна працювати у більшості дистрибутивів Linux, Microsoft, MacOS.

Безпека. Контейнери ізольовані один від одного та базової інфраструктури. Зміна/відновлення/видалення одного контейнера не впливає на інший.

Недоліки технології:

Висока складність. Зростання кількості контейнерів, що працюють із додатком, впливає на складність керування ними. У виробничому середовищі для

роботи з безліччю контейнерів варто використовувати оркестратори. Наприклад, Kubernetes та Mesos.

Розростання. Нерідко в контейнери пакується набагато більше ресурсів, ніж реально потрібно. Через це образ розростається, займаючи більше місця на диску.

Підтримка Native Linux. Docker та багато інших контейнерних технологій засновані на Linux-контейнерах (LXC). Через це запуск контейнерів у середовищі Windows не завжди зручний, а щоденне використання складніше, ніж при роботі в Linux[2].

Недостатня зрілість. Технології контейнеризації додатків з'явилися над ринком порівняно недавно. Не завжди вдається відразу вирішити проблеми, що виникли. Іноді потрібний час на пошук рішення.

Незважаючи на те, що контейнерів може бути багато, вони, як правило, короткоживуть. Docker-контейнери, наприклад, часто називають одноразовими. Можна використовувати його, отримати результат, а потім видалити і запустити такий самий.

1.2. Відомості про Docker

Docker – це відкрита платформа для розробки, доставки та експлуатації програм[3]. Використовуючи контейнери Docker, можна розгортати, копіювати, переносити та робити резервні копії інформації швидше та легше, ніж за допомогою віртуальної машини. Docker ізолює контейнери, змушуючи їх працювати як єдиний процес. Якщо оточення програми складається з X одночасних процесів, Docker запустить X контейнерів, кожен із своїм процесом.

У своєму ядрі Docker дозволяє запускати практично будь-яку програму, яка безпечно ізольована в контейнері. Безпечна ізоляція дозволяє запускати на одному хості кілька контейнерів одночасно. Легка природа контейнера, що запускається без додаткового навантаження гіпервізора, дозволяє оптимально використовувати апаратне забезпечення.

Платформа та засоби контейнерної віртуалізації використовуються у таких випадках:

упаковка програми (і всі залежності) в контейнери Docker;

доставка упакованих контейнерів командам для розробки та тестування;

розгортання упакованих контейнерів на сервери: дата-центри, хмари тощо.

Проект Docker розпочато як внутрішню розробку компанії dotCloud, заснованої Соломоном Хайксом у 2008 році з метою побудови публічної PaaS-платформи з підтримкою різних мов програмування. Поряд з Хайксом у початковій підготовці значну участь взяли інженери dotCloud Андреа Лудзарді та Франсуа-Ксав'є Бурле.

У березні 2013 року код Docker був опублікований під ліцензією Apache 2.0. У червні 2013 року генеральним директором у dotCloud запрошено Бена Голуба, який раніше керував фірмою Gluster (розробляла технологію розподіленого зберігання GlusterFS і поглинуту за $136 млн Red Hat у 2011 році). У жовтні 2013 року, наголошуючи на зміщенні фокусу до нової ключової технології, dotCloud перейменована в Docker (при цьому PaaS-платформа збережена під колишньою назвою – dotCloud).

У жовтні 2013 року випущено реліз Havana тиражованої IaaS-платформи OpenStack, в якому реалізовано підтримку Docker (як драйвер для OpenStack Nova). З листопада 2013 року часткова підтримка Docker включена до дистрибутиву Red Hat Enterprise Linux версії 6.5 і повна – до 20-ї версії дистрибутива Fedora, раніше було досягнуто згоди з Red Hat про включення з 2014 року Docker до тиражованої PaaS-платформи Open Shift. У грудні 2013 року було оголошено про підтримку розгортання Docker-контейнерів у середовищі Google Compute Engine.

З 2014 року ведуться роботи із включення підтримки Docker у середу управління фреймворку розподілених додатків Hadoop. За результатами тестування варіантів платформи віртуалізації для Hadoop, проведеному в травні

2014 року, Docker показав на основних операціях (з масового створення, перезапуску та знищення віртуальних вузлів) значно більш високу продуктивність, ніж KVM, зокрема, на тесті масового створення віртуальних обчислювальних вузлів. процесорних ресурсів у Docker зафіксовано у 26 разів нижче, ніж у KVM, а приріст споживання ресурсів оперативної пам'яті – втричі нижчий.

Docker чудово підходить для організації циклу розробки. Docker дозволяє розробникам використовувати локальні контейнери з програмами та сервісами. Що дозволяє інтегруватися з процесом постійної інтеграції та викладання (continuous integration and deployment workflow).

Наприклад, розробники пишуть код локально і діляться своїм стеком розробки (набором образів docker) з колегами. Коли вони готові, отруюють код та контейнери на тестовий майданчик та запускають усі необхідні тести. З тестового майданчика вони можуть оправити код та образи на продакшен.

Заснована на контейнерах платформа Docker дозволять легко перенести корисне навантаження. Docker контейнери можуть працювати на локальній машині як реальної, так і на віртуальній машині в дата центрі, так і в хмарі.

Імпортування та легка природа docker дозволяє легко динамічно керувати вашим навантаженням. Можна використовувати docker, щоб розгорнути або погасити програму або послуги. Швидкість docker дозволяє робити це в режимі реального часу.

Docker легковагий та швидкий. Він дає стійку, рентабельну альтернативу віртуальним машинам з урахуванням гіпервізора. Він особливо корисний в умовах високих навантажень, наприклад, при створенні власної хмари або платформа-як-сервіс (platform-as-service). Але він також корисний для маленьких і середніх програм, коли вам хочеться отримувати більше наявних ресурсів.

Основний принцип роботи Docker – контейнеризація програм. Цей тип віртуалізації дозволяє упаковувати програмне забезпечення ізольованими

середовищами – контейнерами. Кожен із цих віртуальних блоків містить усі необхідні елементи роботи програми. Це дозволяє одночасного запуску великої кількості контейнерів на одному хості.

Docker використовує клієнт-серверну архітектуру.

Docker клієнт взаємодіє із фоновим процесом – сервером Docker, який у свою чергу запускає контейнери. Клієнт і фоновий процес можуть виконуватися в одній операційній системі, також можна підключити клієнта до віддаленого фонового процесу. Докер клієнт та фоновий процес взаємодіють через API REST поверх сокетів або через мережний інтерфейс.

Фоновий процес Docker приймає запити та керує об'єктами Docker. Фоновий процес може взаємодіяти з іншими фоновими процесами. Фоновий процес керує такими об'єктами:

образи;

контейнери;

мережна взаємодія;

томи.

Docker клієнт є основним способом взаємодії із сервером. При виконанні команди клієнт надсилає цю команду фоновому процесу, який у свою чергу її виконує. Docker клієнт може взаємодіяти з безліччю різних серверів

Щоб розуміти, з чого складається docker, вам потрібно знати про три компоненти:

образ;

реєстр;

контейнер.

Образ – це незмінний шаблон, що містить інструкції створення контейнера. Найчастіше образ базується іншому образі, доповнюючи його нової функціональністю. Наприклад, можна побудувати образ, що базується на образі

Linux Ubuntu, розширивши його встановивши веб-сервер Apache з розробленим додатком, а також налаштований сервер для правильної роботи програми.

Є можливість створювати нові образи або перевикористовувати вже створені та розміщені у реєстрі. Щоб створити новий образ, необхідно створити конфігураційний файл, який називається DockerFile. Цей файл містить інструкції щодо створення та запуску образу. Кожна інструкція у конфігураційному файлі створює шар в образі. Коли конфігураційний файл змінюється і запущено процес повторного збирання образу, тільки шари, що змінилися, будуть перезібрані. Саме ця технологія робить контейнери легковагими та швидкими порівняно з іншими технологіями віртуалізації[4].

Контейнер – це запущений екземпляр образу. Контейнери можна створювати, запускати, зупиняти та видаляти за допомогою API Docker або інтерфейсу командного рядка. Контейнер може бути підключений до однієї чи кількох мереж. До нього може бути змонтована частина файлової системи хост машини. Також можна створити новий образ із поточного стану контейнера.

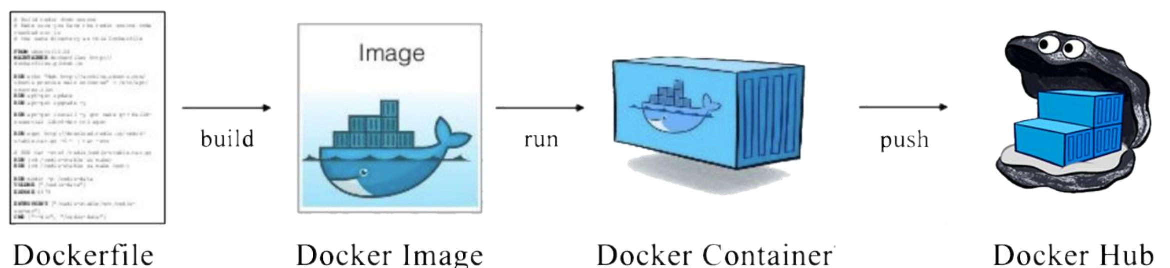

Рис 1.1. Опис циклу роботи в Docker

Реєстр контейнерів зберігає види. Цей реєстр може бути як громадським, так і приватним. Docker Hub є відкритим офіційним регістром, в якому зберігаються офіційні образи від різних постачальників технологій, наприклад, ubuntu. Додавати образи до публічного регістру може будь-хто. За промовчанням Docker налаштований для пошуку образів у цьому публічному реєстрі. Також можна створювати приватні регістри образів. У разі використання команд docker pull або docker run необхідні образи будуть завантажені зі сконфігурованого

реєстру. При використанні команди docker push, вказаний образ буде поміщений у регістр

Попередниками контейнерів Docker були віртуальні машини. Віртуальна машина, як і контейнер, ізолює від зовнішнього середовища додаток та його залежність. Однак контейнери Docker мають переваги перед віртуальними машинами. Так, вони споживають менше ресурсів, їх дуже легко переносити, вони швидше запускаються та приходять у працездатний стан.

Контейнери схожі на директорію. У контейнерах міститься все необхідне роботи програми. Кожен контейнер створюється із образу. Контейнери можуть бути створені, запущені, зупинені, перенесені або вилучені. Кожен ізольований контейнер є безпечною платформою для застосування. Контейнери – це компонент роботи.

Як працює контейнер?

Контейнер складається з операційної системи, файлів користувача та метаданих. Як знаємо, кожен контейнер створюється з образу. Цей образ каже docker-у, що у контейнері, який процес запустити, коли запускається контейнер та інші конфігураційні дані. Docker образ доступний лише для читання. Коли docker запускає контейнер, він створює рівень для читання/запису зверху образу (використовуючи union file system, як було сказано раніше), у якому може бути запущена програма.

Docker-образ - це read-only шаблон. Наприклад, образ може містити операційну систему Ubuntu з Apache та додаток на ній. Образи використовуються створення контейнерів. Docker дозволяє легко створювати нові образи, оновлювати існуючі, або ви можете завантажити образи, створені іншими людьми. Образи – це компонент збирання docker-а.

Як працює образ?

Ми вже знаємо, що образ – це read-only шаблон, з якого створюється контейнер. Кожен образ складається із набору рівнів. Docker використовує union

file system суміщення цих рівнів в один образ. Union file system дозволяє файлам та директоріям з різних файлових систем (різних гілок) прозоро накладатися, створюючи когерентну файлову систему.

Одна з причин, через яку docker легковажний – це використання таких рівнів. При зміні образу, наприклад, оновлюєте програму, створюється новий рівень. Так, без заміни всього образу або його перекладання, як вам можливо, доведеться зробити з віртуальною машиною, тільки рівень додається або оновлюється. І вам не потрібно роздавати весь новий образ, що звучить тільки поновлення, що дозволяє поширювати образи простіше і швидше.

В основі кожного образу є базовий образ. Наприклад, ubuntu, базовий образ Ubuntu або fedora, базовий образ дистрибутива Fedora. Також ви можете використовувати образи як базу для створення нових образів. Наприклад, якщо у вас є образ apache, ви можете використовувати його як базовий образ для ваших веб-додатків.

Docker образи можуть бути створені з цих базових образів, кроки для створення цих образів ми називаємо інструкціями. Кожна інструкція створює новий образ чи рівень. Інструкціями будуть наступні дії:

запуск команди;

додавання файлу або директорії;

створення змінного оточення;

вказівки що запускати коли запускається контейнер цього способу;

Ці інструкції зберігаються у файлі Dockerfile. Docker читає це Dockerfile, коли ви збираєте образ, виконує ці інструкції та повертає кінцевий образ.

Docker-демон (Docker-daemon) - сервер контейнерів, що входить до складу програмних засобів Docker. Демон управляє Docker-об'єктами (мережі, сховища, образи та контейнери). Демон також може зв'язуватися з іншими демонами для керування Docker.

Docker-клієнт (Docker-client/CLI) – інтерфейс взаємодії користувача з Docker-демоном. Клієнт та Демон – найважливіші компоненти «движка» Докера (Docker Engine). Клієнт Docker може взаємодіяти з кількома демонами.

Docker-образ (Docker-image) - файл, що включає залежності, відомості, конфігурацію для подальшого розгортання та ініціалізації контейнера.

Docker-файл (Docker-file) - опис правил зі збирання образу, в якому перший рядок вказує на базовий образ. Наступні команди виконують копіювання файлів та встановлення програм для створення певного середовища для розробки.

Docker-контейнер (Docker-container) - це легкий, автономний виконуваний пакет програмного забезпечення, який включає в себе все необхідне для запуску програми: код, середу виконання, системні інструменти, системні бібліотеки та налаштування.

Том (Volume) - емуляція файлової системи для здійснення операцій читання і запису. Вона створюється автоматично з контейнером, оскільки деякі додатки здійснюють збереження даних.

Реєстр (Docker-registry) - зарезервований сервер, який використовується для зберігання docker-образів. Приклади реєстрів:

Центр Docker - реєстр, який використовується для завантаження docker-image. Він забезпечує їх розміщення і інтеграцію з GitHub і Bitbucket.

Контейнери Azure - призначений для роботи з образами та їх компонентами в директорії Azure (Azure Active Directory).

Довірений реєстр Docker або DTR - служба docker-реєстру для інсталяції на локальному комп'ютері або мережі компанії.

Docker-хаб (Docker-hub) або сховище даних - репозиторій, призначений для зберігання образів з різним програмним забезпеченням. Наявність готових елементів впливає на швидкість розробки.

Docker-хост (Docker-host) - машинне середовище для запуску контейнерів з програмним забезпеченням.

Docker-мережі (Docker-networks) - застосовуються для організації мережевого інтерфейсу між додатками, розгорнутими в контейнерах.

Docker - реєстр зберігає образи. Після створення образу ви можете опублікувати його на державному реєстрі Docker Hub або на вашому особистому реєстрі.

За допомогою Docker клієнта ви можете шукати вже опубліковані образи і завантажувати їх на вашу машину з Docker для створення контейнерів.

Docker Hub надає публічні і приватні сховища образів. Пошук і скачування образів з публічних сховищ є для всіх. Вміст приватних сховищ не влучає у результатів пошуку. Тільки Ви і ваші користувачі можуть отримувати ці образи і створювати з них контейнери.

Реєстр Docker являє собою віддалену платформу, яка використовується для зберігання образів Docker. В ході роботи з Docker образи відправляють до реєстру і завантажують з нього. Подібний реєстр може бути організований тим, хто користується Docker. Крім того, постачальники хмарних послуг можуть підтримувати і власні реєстри. Наприклад, це стосується AWS і Google Cloud.

У контейнерах Docker організувати роботу з тимчасовими даними можна двома способами.

За замовчуванням файли, створювані додатком, що працює в контейнері, зберігаються в шарі контейнера, що підтримує запис. Для того щоб цей механізм працював, нічого спеціально налаштовувати не потрібно. Виходить дешево і сердито. Додатком досить просто зберегти дані і продовжити займатися своїми справами. Однак після того як контейнер перестане існувати, зникнуть і дані, збережені таким ось нехитрим способом.

Для зберігання тимчасових файлів в Docker можна скористатися ще одним рішенням, відповідним для тих випадків, коли потрібно більш високий рівень продуктивності, в порівнянні з тим, який можна досягти при використанні стандартного механізму тимчасового зберігання даних. Якщо вам не потрібно, щоб ваші дані зберігалися б довше, ніж існує контейнер, ви можете підключити до контейнера tmpfs - тимчасове сховище інформації, яке використовує оперативну пам'ять хоста. Це дозволить прискорити виконання операцій по запису і читання даних.

Часто буває так, що дані потрібно зберігати і після того, як контейнер припинить існувати. Для цього нам знадобляться механізми постійного зберігання даних.

Існують два способи, що дозволяють зробити термін життя даних великим терміну життя контейнера. Один із способів полягає у використанні технології bind mount. При такому підході до контейнера можна примонтировать, наприклад, реально існуючу папку. Працювати з даними, що зберігаються в такій папці, зможуть і процеси, що знаходяться за межами Docker. Ось як виглядають монтування tmpfs і технологія bind mount.

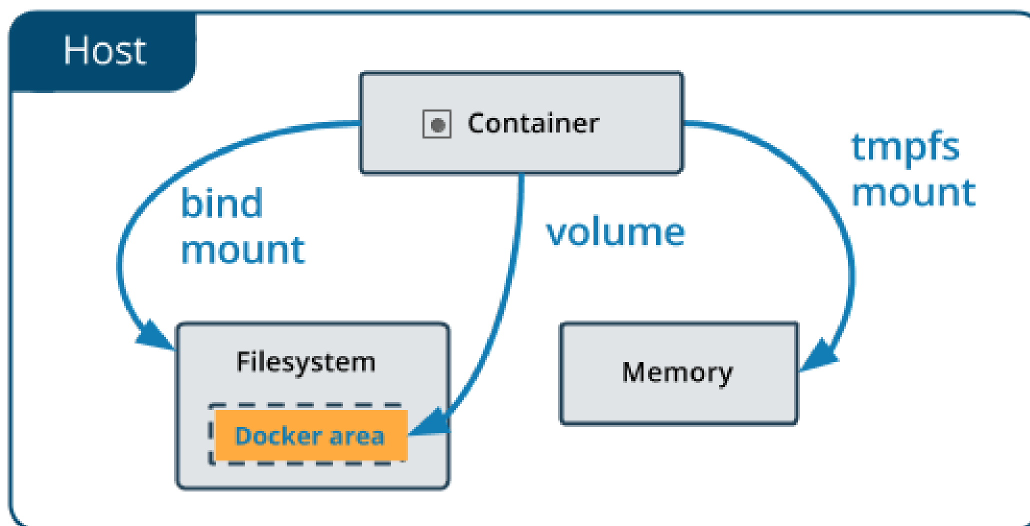

Рис. 1.2. Опис принципу зберігання даних

Мінуси використання технології bind mount полягають в тому, що її використання ускладнює резервне копіювання даних, міграцію даних, спільне використання даних декількома контейнерами. Набагато краще для постійного зберігання даних використовувати томи Docker.

1.3. Docker engine

Docker Engine («Движок» Docker) - ядро механізму Докера. «Движок» відповідає за функціонування та забезпечення зв'язку між основними Docker-об'єктами (реєстром, образами і контейнерами)[6].

Сервер виконує ініціалізацію демона (фонової програми), який застосовується для управління і модифікації контейнерів, образів і тмів.

REST API - механізм, який відповідає за організацію взаємодії Докер-клієнта і Докер-демона.

Клієнт - дозволяє користувачеві взаємодіяти з сервером за допомогою команд, що набираються в інтерфейсі (CLI).

ПК-користувач віддає команду за допомогою клієнтського інтерфейсу Docker-демона, розгорнутому на Docker-хості. Наприклад, скачати готовий образ з реєстру (сховища Docker-образів) за допомогою команди docker pull. Взаємодія між клієнтом і демоном забезпечує REST API. Демон може використовувати публічний (Docker Hub) або приватний реєстри.

Виходячи з команди, заданої клієнтом, демон виконує різні операції з образами на основі інструкцій, прописаних в файлі Dockerfile. Наприклад, виробляє їх автоматичну збірку за допомогою команди docker build.

Робота образу в контейнері. Наприклад, запуск docker-image, за допомогою команди docker run або видалення контейнера через команду docker kill.

Docker-image - шаблон тільки для читання (read-only) з набором деяких інструкцій, призначених для створення контейнера. Він складається з шарів, які Docker комбінує в один образ за допомогою допоміжної файлової системи

UnionFS. Так вирішується проблема нераціонального використання дискової пам'яті. Параметри способу визначаються в Docker-file.

Для багаторазового застосування Docker-image слід користуватися реєстром образів або Докер-реєстром (Docker-registry), що дозволяє закачувати готові образи з зовнішнього сховища сервісу і зберігати їх в реєстрі Докер-хоста. Рекомендований варіант - офіційний реєстр компанії Docker Trusted Registry (DTR).

Якщо потрібний файл з розширенням, то скачиваться будуть тільки потрібні шари. Наприклад, розробник вирішив доопрацювати програмне забезпечення і модифікувати образ, змінивши кілька файлів. Після завантаження на сервер будуть відправлені шари, що містять тільки модифіковані дані.

Як працюють контейнери

Кожен контейнер будується на основі Docker-образів. Контейнери запускаються безпосередньо з ядра операційної системи Linux. Завдяки цьому, вони споживають набагато менше ресурсів, ніж при апаратній віртуалізації.

Ізоляція робочого середовища здійснюється за допомогою технології namespace. Для кожного ізольованого простору (контейнера) створюється унікальний простір імен, яке і забезпечує до нього доступ. Будь-який процес, що виконується всередині контейнера, обмежується namespace.

Що відбувається при запуску контейнера

Відбувається запуск образу (Docker-image). Docker Engine перевіряє існування образу. Якщо образ уже існує локально, Docker використовує його для нового контейнера. При його відсутності виконується скачування з Docker Hub.

Мінімальне споживання ресурсів - контейнери не віртуалізують всю операційну систему (ОС), а використовують ядро хоста та ізолюють програму на рівні процесу. Останній споживає набагато менше ресурсів локального комп'ютера, ніж віртуальна машина.

Швидкісне розгортання - допоміжні компоненти можна не встановлювати, а використовувати вже готові docker-образи (шаблони). Наприклад, не має сенсу постійно встановлювати і налаштовувати Linux Ubuntu. Досить 1 раз її інсталювати, створити образ і постійно використовувати, лише оновлюючи версію при необхідності.

Зручне приховування процесів - для кожного контейнера можна використовувати різні методи обробки даних, приховуючи фонові процеси.

Робота з небезпечним кодом - технологія ізоляції контейнерів дозволяє запускати будь-який код без шкоди для ОС.

Просте масштабування - будь-який проект можна розширити, запровадивши нові контейнери.

Зручний запуск - додаток, що знаходиться всередині контейнера, можна запустити на будь-якому docker-хості.

Оптимізація файлової системи - образ складається з шарів, які дозволяють дуже ефективно використовувати файлову систему.

Які ж проблеми вирішує система віртуалізацій Docker?

Платформа Docker вирішує три основні проблеми розгортання сервісів:

доставка коду на сервер;

запуск коду;

однаковість оточення.

Docker дозволяє ізолювати сервіси від інфраструктури, таким чином досягається можливість доставляти їх набагато швидше. Docker дозволяє управляти інфраструктурою за допомогою тих же принципів, які використовуються при управлінні додатками. При використанні інструментів Docker для доставки, тестування і розгортання, значно скорочується затримка між фіксацією нового коду в системі контролю версій і запуском його на сервері промислової експлуатації.

Docker надає можливість упаковувати і запускати сервіси в ізольованому оточенні, яке і називається контейнером. Дана ізольованість і безпеку дозволяє запускати кілька контейнерів на одному хості одночасно. Контейнери легкі,оскільки не вимагають роботи гіпервізора. Вони запускаються безпосередньо на ядрі хостової машини. Таким чином,досягається можливість запуску більшої кількості контейнерів, ніж віртуальних машин, на одному і тому ж фізичному обладнанні. Також, Docker контейнери можна запускати в самих віртуальних машинах.

Docker надає наступні інструменти і платформу для управління життєвим циклом контейнерів. За допомогою Docker відбувається упаковка додатки і їх компонентів в контейнер. Після розробки, додаток може бути розгорнуто на сервері вручну. Дана техніка розгортання одноманітна незалежно від того де розгортається додаток, будь то в локальному дата центрі, хмарному провайдера або в гібридному оточенні.

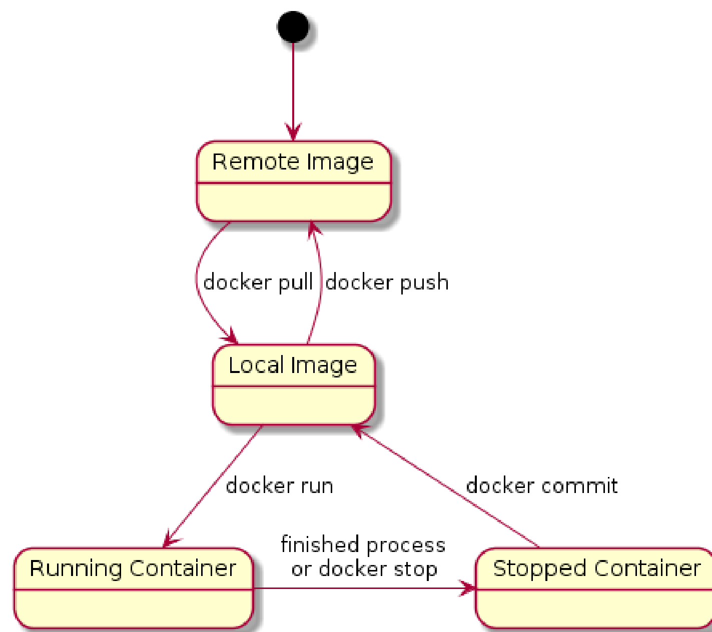

Рис. 1.3. Життєвий цикл образу

Для управління декількома контейнерами, з яких складається проект, використовують пакетний менеджер— Docker Compose.

Він застосовується не у всіх випадках. Якщо проект є простим додатком, що не вимагає використання сторонніх сервісів, то для його розгортання можна обмежитися тільки Docker. Docker Compose рекомендується використовувати при проектуванні складних програмних продуктів, що включають в себе безліч процесів і сервісів.

Кластер серверів (Server Cluster) - це певна кількість серверів, що об'єднані в групу та утворюють єдиний ресурс. Дане рішення дозволяє істотно збільшити надійність і продуктивність системи.

Згруповані в локальну мережу кілька комп'ютерів можна назвати апаратним кластером, проте, суть даного об'єднання - підвищення стабільності і працездатності системи за рахунок єдиного програмного забезпечення під управлінням модуля (Cluster Manager).

Загальна логіка кластера серверів створюється на рівні програмних протоколів і дає можливість:

керувати будь-якою кількістю апаратних засобів за допомогою одного програмного модуля;

додавати і вдосконалювати програмні і апаратні ресурси, без зупинки системи і масштабних архітектурних перетворень;

забезпечувати безперебійну роботу системи, при виході з ладу одного або декількох серверів;

синхронізувати дані між серверами - одиницями кластера;

ефективно розподіляти клієнтські запити по серверам;

використовувати загальну базу даних кластера.

По суті, головним завданням кластера серверів є виключення простою системи. В ідеалі, будь-який інцидент, пов'язаний із зовнішнім втручанням або внутрішнім збоєм в роботі ресурсу, повинен залишатися непоміченим для користувача.

При проектуванні систем за участю серверного кластера необхідно враховувати можливості клієнтського програмного забезпечення по ідентифікації кластера та спільної роботи з командним модулем (Cluster Manager). В іншому випадку можлива ситуація, при якій спроба програми-клієнта за допомогою модуля отримати доступ до даних ресурсу через інші сервера може отримати відмову (конкретні механізми в даному випадку залежать від можливостей і налаштувань кластера і клієнтського обладнання).

При перетворенні хостів в кластер потрібно скористатися утилітою кластеризації Docker Swarm. Хост, що знаходиться в його складі, називається «вузлом» (node), який буває керуючим або робочим. Один кластер містить тільки один керуючий «вузол».

Деякі можливості утиліти

Управління навантажувальними характеристиками - здійснюється оптимізація розсилки запитів між хостами, забезпечуючи на них рівномірне навантаження.

Динамічне управління - допускається додавання елементів в swarm-кластер без подальшого його перезапуску.

Можливість масштабування - дозволяє додавати або видаляти docker-образ для автоматичного створення контейнера.

Відновлення «вузла» після збою - працездатність кожного хоста постійно контролюється керівником «вузлом». При збої кластера відбувається його відновлення і перезапуск.

Rolling-update - виконує оновлення контейнерів. Процедура може виконуватися в певній послідовності і з тимчасовою затримкою для запуску іншого контейнера. Параметр вказується в налаштуваннях. Якщо станеться збій поновлення, то Docker Swarm видасть помилку і процес повториться заново.

Docker є важливим інструментом для кожного сучасного розробника, як основа апаратної віртуалізації додатків. Ця технологія має широкий функціонал і

можливостями для контролю процесів. Докер дозволяє не тільки розгортати контейнери, а й оперативно масштабувати їх екземпляри, працювати з многоконтейнерними додатками (Docker Compose), а також об'єднати кілька докер-хостів в єдиний кластер (Docker Swarm)[7].

Докер характеризується досить простим синтаксисом. Тому він досить простий в освоєнні як для досвідчених IT-фахівців, так і для новачків. Програмне забезпечення сумісне з усіма версіями операційних систем Linux і Windows, тому область застосування Docker практично не обмежена.

1.4. Мережі Docker.

Докер може працювати в мережах bridge, overlay, host і macvlan(ipvlan[10])[13].

Brigde це програмний режим моста. За замовчуванням Docker використовує мережу з назвою bridge для всіх контейнерів для спілкування в межах однієї машини, якщо для них не описуються інші мережі. Для кожного контейнера створюється свій віртуальний мережний інтерфейс, він підключається до мережі за допомогою bridge.

Відмінності мосту за замовчуванням та визначеного користувачем:

Описані користувачем контейнери надає найкращу ізоляцію контейнеризованих додатків. Контейнери, підключені до мосту користувача відкривають всі прокинуті порти всередині цього мосту один одному локально, але не в публічну мережу.

Визначені користувачем мости надають автоматичну роздільну здатність DNS імені між контейнерами. Контейнери в стандартній мережі мосту можуть звертатися один до одного тільки по IP-адресах доки не вказано параметр --link (так само підтримується в Compose, але він визнаний застарілим і використовувати його більше не рекомендується в жодному разі). У випадку з мережею типу міст можна звертатися до контейнера по його імені або заданому або.

Контейнери можуть бути підключені та відключені від мережі користувальницької мережі на льоту. Це дозволяє не перетворюючи контейнер налаштувати його параметри в мережі, наприклад, вказати іншу статичну IP-адресу.

Мости користувача можуть налаштовуватися. Користувальницькі мости налаштовуються та керуються через docker network create або в Docker Compose файлі. Налаштування можна змінювати на льоту. У рідного мосту змінювати налаштування треба змінюючи файл конфігурації daemon.json, а також він використовує єдині налаштування iptables і MTU.

Пов'язані прапором --link контейнери у рідній мережі "міст" ділять змінні оточення. Так як після впровадження мереж, що настроюються режим --link застарів і не рекомендується до використання, то розглядати подробиці цього пункту сенсу не має.

Overlay це розподілена мережа серед кількох хостів Docker. Назва мережі походить від слова "прошарок" (overlay), тому що оскільки мережа є прошарком для комунікації контейнерів у розподіленій мережі.

Мережа типу overlay вимагає, щоб хост був частиною мережі Swarm. У Swarm за замовчуванням використовується мережа типу overlay з ім'ям ingress для розподілу навантаження, а також мережа типу bridge під назвою docker_gwbridge для комунікації самих Docker daemon.

Для overlay мереж є також вимоги до фаєрволу, тобто відкритих в ньому портів:

TCP 2377 для комунікації управління кластера;

TCP та UDP 7946 для комунікації нод кластера;

UDP 4789 для трафіку мережі overlay.

У користувача overlay мережі є кілька цікавих параметрів:

--attachable: дозволяє спілкуватися як сервісам, а й окремим контейнерам коїться з іншими контейнерами не більше swarm;

--opt encrypted: включає шифрування AES у режимі GCM з ротацією ключів кожні 12 годин. На жаль, так як використовується віртуальна LAN (VxLAN), то навантаження на ЦП сильно зростає, тому опція може не підходити для продакшену. Windows хости також не підтримують опцію.

Host це режим мережі, при використанні якого стек контейнера не ізольований від хоста Docker. як якщо контейнер прив'язується до 80 порту (наприклад, nginx), він буде доступний порту 80 IP-адресу хоста.

При використанні режиму в Docker swarm використовується overlay мережа для керуючого трафіку, а контейнер доступний тільки з однієї машини, до якої прив'язався. Це створює обмеження, яке не дозволяє використовувати в swarm на машині із зайнятим портом додаток. Таким чином, якщо в Swarm використовується порт 80 в розподіленій мережі і на одній машині з swarm є контейнер з host режимом, то на цій машині не буде доступна програма з розподіленої мережі. Docker використовує технологію просторів імен Linux для ізоляції ресурсів, таку як процес ізоляції простору імен PID, файлова система ізоляції простору імен монтування, мережа ізоляції простору імен мережі і т.д. , маршрутизація, правила Iptable тощо ізольовані з іншого мережного простору імен. Контейнеру Docker зазвичай призначається незалежний простір імен мережі. Але якщо режим хоста використовується при запуску контейнера, контейнер не отримає незалежний простір імен мережі, а спільно використовуватиме простір імен мережі з хостом. Контейнер не віртуалізуватиме свою власну мережеву карту, налаштовуватиме власну IP-адресу і т. д., а використовуватиме IP-адресу хоста і порт. Коли ми виконуємо будь-яку команду ifconfig в контейнері для перегляду мережевого оточення, все, що ми бачимо, - це інформація про хост-машину. Щоб зовнішній світ міг отримати доступ до програм у контейнері, ви можете безпосередньо використовувати ip-адресу з портом без будь-якого перетворення NAT, так само, як і запуск безпосередньо на хост-комп'ютері. Однак

інші аспекти контейнера, такі як файлова система та список процесів, як і раніше, ізольовані від хоста.

Деякі програми, наприклад для моніторингу трафіку, повинні бути безпосередньо підключені до фізичної мережі. Це можна зробити за допомогою режиму macvlan.

Необхідно розуміти, що для цього режиму використовується фізичний інтерфейс мережі, який має доступ до фізичної мережі. Для ізоляції від основної мережі необхідно використовувати інший мережний інтерфейс або створити VLAN.

При використанні режиму слід розуміти, що

можна завдати шкоди своєї мережі вичерпавши запас IP-адресів або необдуманим створенням VLANів;

мережний інтерфейс повинен підтримувати використання кількох MAC-адрес;

Якщо програма може працювати з bridge або overlay режимами, то краще вибрати їх.

macvlan може працювати у двох режимах:

мережевий міст, де трафік проходить через фізичний адаптер хоста;

мережевий міст з використанням стандарту 802.1Q, де тегується трафік та показується приналежність до певного VLAN. Docker створює мережевий "підінтерфейс" над основним, дозволяючи налаштовувати маршрутизацію та фільтрацію на вищому рівні.

Висновок:

В даному розділі було описано теоретичні відомості про технологію Docker. Аналіз об’єкта та предмета дослідження, знайомство з основними поняттями та обробка теоретичних даних дозволили сформувати уявлення про майбутнє дослідження.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ

2.1. Першочергові пояснення

Для порівняння типів мережі Docker, побудови контейнерів та роботи з ними в мережі я перейшов на Docker Desktop останної на момент складання дисертації версії Docker Desktop 4.15.0.

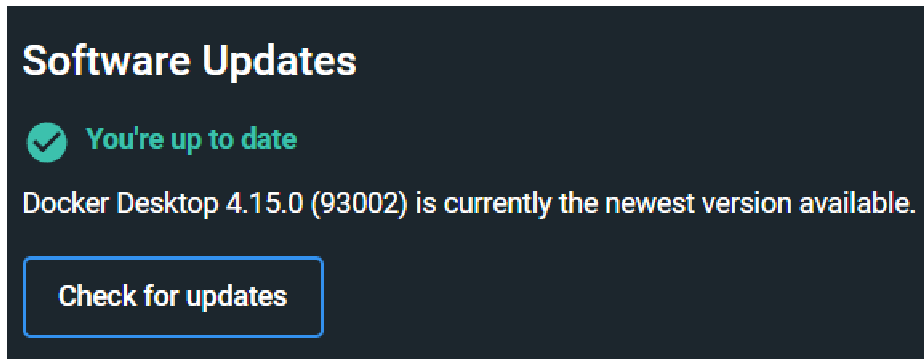

Рис. 2.1 Версія Docker Desktop

2.2. Налаштування контейнерів Docker

Для початку роботи з Docker Desktop я завантажую образ docker/getting-started командою:

docker run -d -p 80:80 docker/getting-started

що означає відкриття 80 порту для контейнерів, створюваного цим образом.

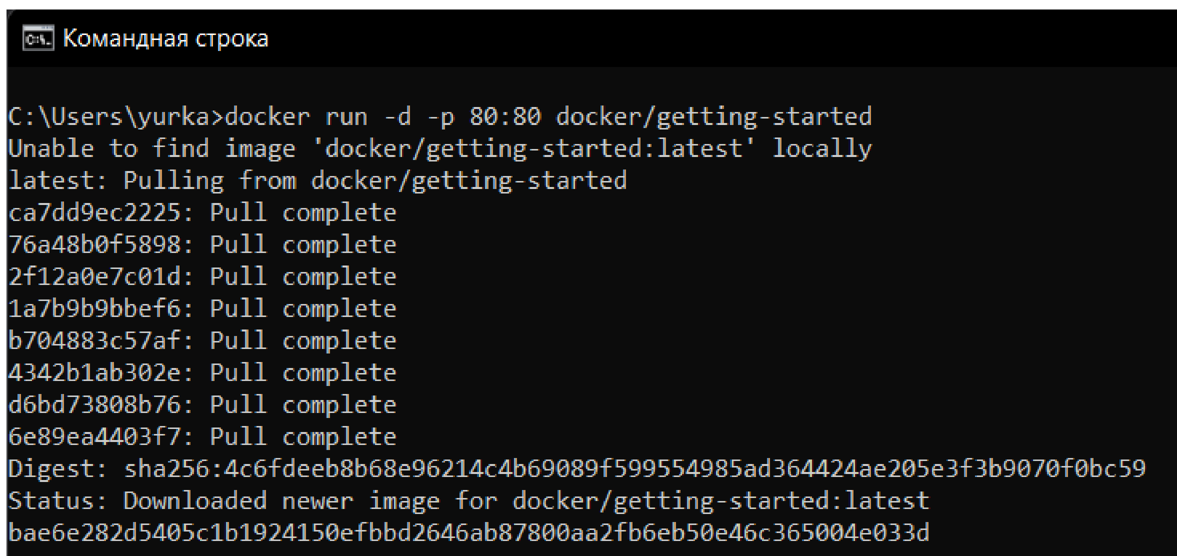

Рис. 2.2 Результат виконання команди

Рис. 2.3 Завантажений образ відображається у вікні образів Docker Decktop

Далі зроблю те ж саме ще з кількома образами і ось що вийде:

NAME	TAG	STATUS	CREATED	SIZE	ACTIONS
busybox 334e4a014c81	latest	In use	3 days ago	4.86 MB	
nginx ac8efec875ce	latest	In use	4 days ago	141.77 MB	
ntop/ntopng 5ca501354840	latest	In use	9 days ago	1.23 GB	
docker/getting-started e5be50c31cb9	latest	In use	14 days ago	29.79 MB	

Рис. 2.4 Перелік образів в Docker Desktop

Для цього я виконав такі команди:

docker pull nginx

```
C:\Users\yurka>docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
025c56f98b67: Pull complete
ca9c7f45d396: Pull complete
ed6bd111fc08: Pull complete
e25b13a5f70d: Pull complete
9bbabac55ab6: Pull complete
e5c9ba265ded: Pull complete
Digest: sha256:ab589a3c466e347b1c0573be23356676df90cd7ce2dbf6ec332a5f0a8b5e59db
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

Рис. 2.5 Результат виконання команди

docker run ntop/ntopng

```
Командная строка

C:\Users\yurka>docker run ntop/ntopng
Starting redis-server: redis-server.
09/Dec/2022 17:14:07 [Redis.cpp:151] Successfully connected to redis 127.0.0.1:6379@0
09/Dec/2022 17:14:07 [Redis.cpp:151] Successfully connected to redis 127.0.0.1:6379@0
09/Dec/2022 17:14:07 [NtopPro.cpp:311] [LICENSE] No license file found /etc/ntopng.license: reading license from redis
09/Dec/2022 17:14:07 [NtopPro.cpp:483] [LICENSE] Unable to validate license [Empty license file]
09/Dec/2022 17:14:07 [NtopPro.cpp:552] WARNING: [LICENSE] Invalid license [Empty license file]
09/Dec/2022 17:14:07 [NtopPro.cpp:569] WARNING: [LICENSE] ntopng will now run in Enterprise L edition for 10 minutes
09/Dec/2022 17:14:07 [NtopPro.cpp:571] WARNING: [LICENSE] before returning to community mode
09/Dec/2022 17:14:07 [NtopPro.cpp:573] WARNING: [LICENSE] You can buy a permanent license at http://shop.ntop.org
09/Dec/2022 17:14:07 [NtopPro.cpp:575] WARNING: [LICENSE] or run ntopng in community mode starting
09/Dec/2022 17:14:07 [NtopPro.cpp:576] WARNING: [LICENSE] ntopng --community
09/Dec/2022 17:14:08 [NetworkInterface.cpp:3278] Cleanup interface dummy
09/Dec/2022 17:14:08 [NetworkInterface.cpp:3278] Cleanup interface lo
09/Dec/2022 17:14:08 [main.cpp:240] Unable to open interface lo [97]: PF_RING not loaded. Falling back to pcap.
09/Dec/2022 17:14:08 [PcapInterface.cpp:96] Reading packets from  [id: 1]
09/Dec/2022 17:14:08 [Ntop.cpp:2589] Registered interface lo [id: 1]
09/Dec/2022 17:14:08 [NetworkInterface.cpp:3278] Cleanup interface eth0
09/Dec/2022 17:14:10 [main.cpp:240] Unable to open interface eth0 [97]: PF_RING not loaded. Falling back to pcap.
09/Dec/2022 17:14:10 [PcapInterface.cpp:96] Reading packets from  [id: 2]
09/Dec/2022 17:14:10 [Ntop.cpp:2589] Registered interface eth0 [id: 2]
09/Dec/2022 17:14:10 [main.cpp:337] PID stored in file /var/run/ntopng.pid
09/Dec/2022 17:14:10 [Geolocation.cpp:149] Loaded database dbip-city-lite.mmdb [/usr/share/ntopng/httpdocs/geoip//dbip-city-lite.mmdb][ip_ver
sion: 6]
09/Dec/2022 17:14:10 [Geolocation.cpp:149] Loaded database dbip-asn-lite.mmdb [/usr/share/ntopng/httpdocs/geoip//dbip-asn-lite.mmdb][ip_versi
on: 6]
09/Dec/2022 17:14:10 [Geolocation.cpp:95] Using geolocation provided by DB-IP (https://db-ip.com)
09/Dec/2022 17:14:11 [HTTPserver.cpp:1408] Found TLS certificate /usr/share/ntopng/httpdocs/ssl/ntopng-cert.pem
09/Dec/2022 17:14:11 [HTTPserver.cpp:1683] Web server dirs [/usr/share/ntopng/httpdocs][/usr/share/ntopng/scripts]
```

```
09/Dec/2022 17:14:11 [HTTPserver.cpp:1686] HTTP server listening on 3000
09/Dec/2022 17:14:11 [Utils.cpp:3672] WARNING: Capabilities cap_set_proc error: Operation not permitted
09/Dec/2022 17:14:11 [Utils.cpp:738] WARNING: Unable to retain privileges for privileged file writing
09/Dec/2022 17:14:11 [Utils.cpp:784] User changed to ntopng
09/Dec/2022 17:14:11 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface lo [id: 1]...
09/Dec/2022 17:14:11 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface lo [id: 1]...
09/Dec/2022 17:14:11 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface eth0 [id: 2]...
09/Dec/2022 17:14:11 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface eth0 [id: 2]...
09/Dec/2022 17:14:11 [main.cpp:407] Working directory: /var/lib/ntopng
09/Dec/2022 17:14:11 [main.cpp:409] Scripts/HTML pages directory: /usr/share/ntopng
09/Dec/2022 17:14:11 [Ntop.cpp:466] Welcome to ntopng x86_64 v.5.4.221130 (5.4-stable:150715ac4c8dfdac2978c2cd31c5ee081870116c:20221130) - (C
) 1998-22 ntop.org
09/Dec/2022 17:14:11 [Ntop.cpp:476] Built on Ubuntu 22.04.1 LTS
09/Dec/2022 17:14:11 [NtopPro.cpp:776] [LICENSE] System Id:     LBE140604820DAA8A--OL
09/Dec/2022 17:14:11 [NtopPro.cpp:777] [LICENSE] Edition:       Enterprise L (Bundle)
09/Dec/2022 17:14:11 [NtopPro.cpp:778] [LICENSE] License Type:  Time-Limited [Empty license file] License
09/Dec/2022 17:14:11 [NtopPro.cpp:798] [LICENSE] Validity:      Until Fri Dec  9 17:24:07 2022
09/Dec/2022 17:14:11 [Ntop.cpp:879] Adding 127.0.0.1/32 as IPv4 interface address for lo
09/Dec/2022 17:14:11 [Ntop.cpp:888] Adding 127.0.0.0/8 as IPv4 local network for lo
09/Dec/2022 17:14:11 [Ntop.cpp:879] Adding 172.17.0.2/32 as IPv4 interface address for eth0
09/Dec/2022 17:14:11 [Ntop.cpp:888] Adding 172.17.0.0/16 as IPv4 local network for eth0
09/Dec/2022 17:14:11 [PeriodicActivities.cpp:109] Started periodic activities loop...
09/Dec/2022 17:14:12 [startup.lua:35] Processing startup.lua: please hold on...
09/Dec/2022 17:14:12 [startup.lua:120] [lists_utils.lua:811] Refreshing category lists...
09/Dec/2022 17:14:13 [startup.lua:120] [lists_utils.lua:439] Updating list 'Abuse.ch URLhaus' [https://urlhaus.abuse.ch/downloads/hostfile/].
.. OK
09/Dec/2022 17:14:14 [startup.lua:120] [lists_utils.lua:439] Updating list 'Emerging Threats' [https://rules.emergingthreats.net/fwrules/emer
```

```
ging-Block-IPs.txt]... OK
09/Dec/2022 17:14:14 [startup.lua:120] [lists_utils.lua:439] Updating list 'Feodo' [https://feodotracker.abuse.ch/downloads/ipblocklist_recom
mended.txt]... OK
09/Dec/2022 17:14:15 [startup.lua:120] [lists_utils.lua:439] Updating list 'Feodo Tracker Botnet C2 IP Blocklist' [https://feodotracker.abuse
.ch/downloads/ipblocklist.txt]... OK
09/Dec/2022 17:14:15 [startup.lua:120] [lists_utils.lua:439] Updating list 'NoCoin Filter List' [https://raw.githubusercontent.com/hoshsadiq/
adblock-nocoin-list/master/hosts.txt]... OK
09/Dec/2022 17:14:16 [startup.lua:120] [lists_utils.lua:439] Updating list 'SSLBL Botnet C2 IP Blacklist' [https://sslbl.abuse.ch/blacklist/s
slipblacklist.txt]... OK
09/Dec/2022 17:14:17 [startup.lua:120] [lists_utils.lua:439] Updating list 'ThreatFox' [https://threatfox.abuse.ch/downloads/hostfile/]... OK
09/Dec/2022 17:14:18 [startup.lua:120] [lists_utils.lua:439] Updating list 'URLhaus' [https://urlhaus.abuse.ch/downloads/hostfile/]... OK
09/Dec/2022 17:14:18 [startup.lua:120] [lists_utils.lua:439] Updating list 'dshield 7 days' [https://raw.githubusercontent.com/firehol/blockl
ist-ipsets/master/dshield_7d.netset]... OK
09/Dec/2022 17:14:19 [startup.lua:120] [lists_utils.lua:739] Category Lists (20518 hosts, 2724 IPs, 0 JA3) loaded in 1 sec
09/Dec/2022 17:14:19 [startup.lua:124] Initializing device polices...
09/Dec/2022 17:14:19 [startup.lua:140] Initializing alerts...
09/Dec/2022 17:14:19 [startup.lua:149] Initializing timeseries...
09/Dec/2022 17:14:19 [startup.lua:212] Fetching latest ntop blog posts...
09/Dec/2022 17:14:20 [startup.lua:216] Completed startup.lua
09/Dec/2022 17:14:20 [PeriodicActivities.cpp:167] Found 10 activities
09/Dec/2022 17:14:20 [NetworkInterface.cpp:3245] Started packet polling on interface lo [id: 1]...
09/Dec/2022 17:14:20 [NetworkInterface.cpp:3245] Started packet polling on interface eth0 [id: 2]...

C:\Users\yurka>
```

Рис. 2.6-2.8 Результат виконання команди

Створюю декілька контейнерів з образів, які завантажились:

docker container run --name web1 -d -p 8080:80 nginx

```
C:\Users\yurka>docker container run --name web1 -d -p 8080:80 nginx
bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69
```

Рис. 2.9 Результат виконання команди

Створився контейнер з образу nginx з назвою web1 та відкритий у 8080 порту. Строка знизу це ідентифікатор контейнера в системі Docker. Таким самим чином створю ще два контейнери образу nginx з портами 8081 та 8082 та назвами web2 та web3.

```
C:\Users\yurka>docker container run --name web2 -d -p 8081:80 nginx
1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56

C:\Users\yurka>docker container run --name web2 -d -p 8082:80 nginx
docker: Error response from daemon: Conflict. The container name "/web2" is already in use by container "1e7ac20e4961a58a86d727c54c183a64b18d
3af7ac05ee8b6cea2c1b28d3ef56". You have to remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.

C:\Users\yurka>docker container run --name web3 -d -p 8082:80 nginx
3bea4a9ca15e3bf31c87e7aa4419bad07b925c2ac392c5fa10d693ce423525fa
```

Рис 2.10 Результат створення контейнерів

В середній команді я забув виправити назву створюваного контейнера з web2 на web3, тому виникла помилка.

2.3. Налаштування та тестування мереж.

В Docker за замовчуванням створені декілька мереж, але вони не налаштовані. Всі створювані контейнери доєднуються до мережі bridge за замовчуванням, якщо в команді створення контейнера не вказана конкретна мережа. Всі наявні мережі можна переглянути командою

docker network ls

```
C:\Users\yurka>docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
f4b2da7ee988   \                 macvlan   local
908962303bac   bridge            bridge    local
423afd1455c6   docker_gwbridge   bridge    local
22df8c8588a2   host              host      local
tyibmbmvqja9   ingress           overlay   swarm
20453e5227d9   network_name3     bridge    local
6a921d8fad07   network_name4     macvlan   local
8648f1ad086b   none              null      local

C:\Users\yurka>
```

Рис 2.11 Перелік наявних та створених мереж Docker

Червоною рискою відмічені ті мережі, що Докер створює автоматично за замовчуванням. Мережі bridge, host та null наявні з самого початку, а мережа overlay створюється при запуску мережевої служби swarm. Для порівняння мереж, їх перевірки та тестування створю нові мережі bridge, macvlan, overlay і host.

Почну з мережі bridge:

docker network create --subnet 10.0.3.0/24 --gateway=10.0.3.1 --ip-range 10.0.3.0/24 --driver=bridge --label=my_network network_name3

Цією командою створиться нова мережа bridge з підмережею 10.0.3.0/24, шлюзом 10.0.3.1 та назвою network_name3.

```
C:\Users\yurka>docker network create --subnet 10.0.3.0/24 --gateway=10.0.3.1 --ip-range 10.0.3.0/24 --driver=bridge --label=my_network networ
k_name3
20453e5227d94fe4765e61b1b3876a2ae90a10098041eaaaa0a94e4e54e21d6b
```

Рис. 2.12 Результат виконання команди

Командою

docker network inspect network_name3

передивлюсь інформацію про новостворену мережу.

```
C:\Users\yurka>docker network inspect network_name3
[
    {
        "Name": "network_name3",
        "Id": "20453e5227d94fe4765e61b1b3876a2ae90a10098041eaaaa0a94e4e54e21d6b",
        "Created": "2022-12-09T22:12:01.012248Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.0.3.0/24",
                    "IPRange": "10.0.3.0/24",
                    "Gateway": "10.0.3.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {
            "my_network": ""
        }
    }
]
```

Рис 2.13 Характеристики мережі network_name3

Так як контейнери були підключені до мережі bridge за замовчуванням, відключу їх від тієї мережі і підключу їх до мережі bridge network_name3:

docker network disconnect 908962303bac bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69

docker network disconnect 908962303bac ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f

docker network disconnect 908962303bac 1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56

docker network connect 20453e5227d9 ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f

docker network connect 20453e5227d9 bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69

docker network connect 20453e5227d9 1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56

```
C:\Users\yurka>docker network disconnect 908962303bac bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69

C:\Users\yurka>docker network disconnect 908962303bac ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f

C:\Users\yurka>docker network disconnect 908962303bac 1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56

C:\Users\yurka>docker network connect 20453e5227d9 ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f

C:\Users\yurka>docker network connect 20453e5227d9 bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69

C:\Users\yurka>docker network connect 20453e5227d9 1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56
```

Рис 2.14 Результат виконання команди

Синтаксис команди виглядає так: спочатку вказується ідентифікатор мережі, від якої від’єднується чи приєднується контейнер, потім ідентифікатор самого контейнеру.

За результатами цих команд контейнери підключені до мережі network_name3. Перевірю це командою

docker network inspect network_name3

Командная строка

```
C:\Users\yurka>docker network inspect network_name3
[
    {
        "Name": "network_name3",
        "Id": "20453e5227d94fe4765e61b1b3876a2ae90a10098041eaaaa0a94e4e54e21d6b",
        "Created": "2022-12-09T22:12:01.012248Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.0.3.0/24",
                    "IPRange": "10.0.3.0/24",
                    "Gateway": "10.0.3.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56": {
                "Name": "web2",
                "EndpointID": "078f4ce9325d5bc5ca29ffbf468695fc582a21ae289ce1b56eae1be467ca3f78",
                "MacAddress": "02:42:0a:00:03:04",
                "IPv4Address": "10.0.3.4/24",
                "IPv6Address": ""
            },
            "bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69": {
                "Name": "web1",
                "EndpointID": "fadf97a145a6ec345ddc411260c151456c14ed9c2b8dea3ce9f6ae49a32c035a",
                "MacAddress": "02:42:0a:00:03:03",
                "IPv4Address": "10.0.3.3/24",
                "IPv6Address": ""
            },
            "ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f": {
                "Name": "upbeat_mirzakhani",
                "EndpointID": "0c577a698e2e84798935d73429265ef8cf5a103297f80af822323088193478bd",
                "MacAddress": "02:42:0a:00:03:02",
                "IPv4Address": "10.0.3.2/24",
                "IPv6Address": ""
```

```
            }
        },
        "Options": {},
        "Labels": {
            "my_network": ""
        }
    }
]
```

Рис 2.15-2.16. Результат виконання команди

Як ми можемо побачити, контейнери дійсно під'єднані до мережі network_name3 і мають IPv4 адреси від цієї мережі. Тепер перевірю працездатність мережі і зв'язок контейнерів, пропінгувавши контейнер по одному з IP-адрес. Для цього введу команду

docker exec –it bae6e282d5405c1b1924150efbbd2646ab87800aa2fb6eb50e46c365004e033d ping 10.0.3.4

```
64 bytes from 10.0.3.4: seq=4 ttl=64 time=0.124 ms
64 bytes from 10.0.3.4: seq=5 ttl=64 time=0.137 ms
64 bytes from 10.0.3.4: seq=6 ttl=64 time=0.123 ms
64 bytes from 10.0.3.4: seq=7 ttl=64 time=0.117 ms
64 bytes from 10.0.3.4: seq=8 ttl=64 time=0.113 ms
64 bytes from 10.0.3.4: seq=9 ttl=64 time=0.143 ms
64 bytes from 10.0.3.4: seq=10 ttl=64 time=0.114 ms
64 bytes from 10.0.3.4: seq=11 ttl=64 time=0.131 ms
64 bytes from 10.0.3.4: seq=12 ttl=64 time=0.144 ms
64 bytes from 10.0.3.4: seq=13 ttl=64 time=0.120 ms
64 bytes from 10.0.3.4: seq=14 ttl=64 time=0.148 ms
64 bytes from 10.0.3.4: seq=15 ttl=64 time=0.137 ms
64 bytes from 10.0.3.4: seq=16 ttl=64 time=0.110 ms
64 bytes from 10.0.3.4: seq=17 ttl=64 time=0.112 ms
64 bytes from 10.0.3.4: seq=18 ttl=64 time=0.110 ms
64 bytes from 10.0.3.4: seq=19 ttl=64 time=0.113 ms
64 bytes from 10.0.3.4: seq=20 ttl=64 time=0.130 ms
64 bytes from 10.0.3.4: seq=21 ttl=64 time=0.143 ms
64 bytes from 10.0.3.4: seq=22 ttl=64 time=0.127 ms
64 bytes from 10.0.3.4: seq=23 ttl=64 time=0.119 ms
64 bytes from 10.0.3.4: seq=24 ttl=64 time=0.117 ms
64 bytes from 10.0.3.4: seq=25 ttl=64 time=0.132 ms
64 bytes from 10.0.3.4: seq=26 ttl=64 time=0.125 ms
64 bytes from 10.0.3.4: seq=27 ttl=64 time=0.120 ms
64 bytes from 10.0.3.4: seq=28 ttl=64 time=0.157 ms
64 bytes from 10.0.3.4: seq=29 ttl=64 time=0.145 ms
64 bytes from 10.0.3.4: seq=30 ttl=64 time=0.122 ms
64 bytes from 10.0.3.4: seq=31 ttl=64 time=0.118 ms
64 bytes from 10.0.3.4: seq=32 ttl=64 time=0.124 ms
```

Рис 2.17. Результат виконання команди

В цій колонці нас цікавить час відклику, бо робота мережі, якісний зв'язок контейнерів, завантаженість мережі запитами безпосередньо впливають на нього. Для того, щоб протестувати мережу під навантаженням, буду використовувати генератор трафіку Nsasoft Network Traffic Emulator. Він дозволяє генерувати трафік з підмережі до певної IP-адреси, можна вибрати число пакетів трафіку, які

доставляться до контейнера. Також є можливість вибрати тип трафіку за протоколом.

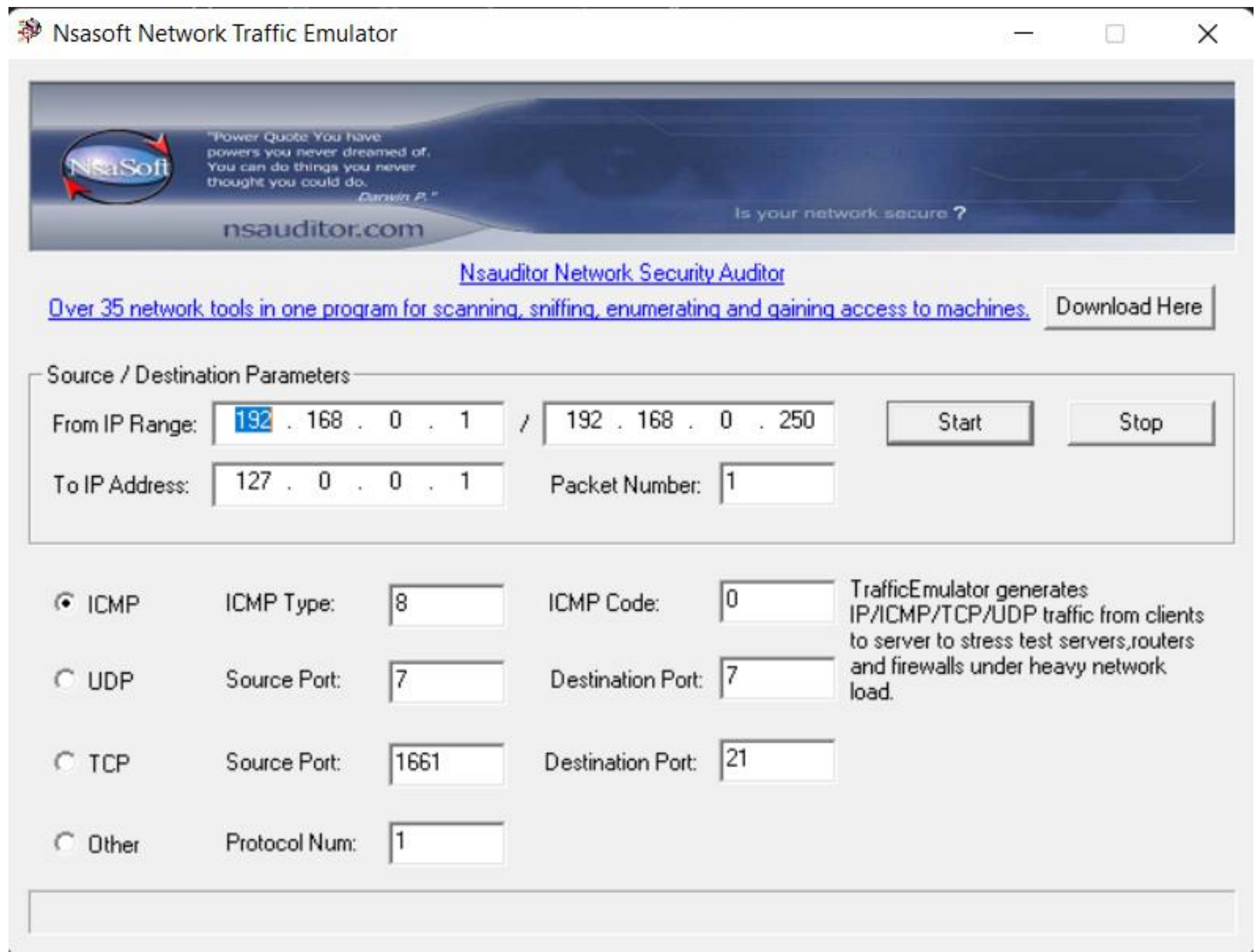

Рис. 2.18. Вікно програми для генерації трафіку після відкриття.

Для тестування мережі введемо підмережу нашої мережі та IP-адресу контейнера, який власне пінгується. Такий режим тестування не є дуже надійним та наочним, бо не видно графіків, рисунків та іншої корисної інформації, але навіть така проста перевірка може показати надійність використовуємої мережі на базовому рівні.

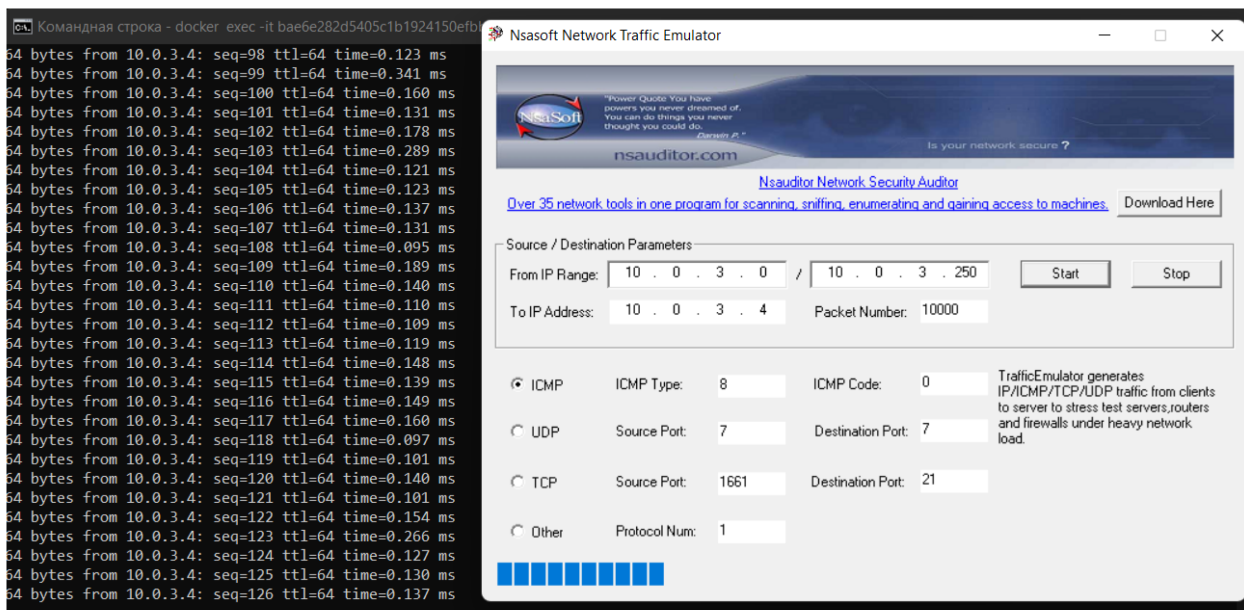

Рис 2.19. Результати після початку генерації трафіка

Як можна побачити, результати часу відгуку, який в стані спокою знаходився на одному рівні приблизно 0.120 мс, починає відхилятись від норми і збільшуватись. Через певний час під навантаженням контейнер всередині мережі пінгується з великими затримками:

```
64 bytes from 10.0.3.4: seq=54 ttl=64 time=0.430 ms
64 bytes from 10.0.3.4: seq=55 ttl=64 time=0.262 ms
64 bytes from 10.0.3.4: seq=56 ttl=64 time=0.368 ms
64 bytes from 10.0.3.4: seq=57 ttl=64 time=0.758 ms
64 bytes from 10.0.3.4: seq=58 ttl=64 time=0.183 ms
64 bytes from 10.0.3.4: seq=59 ttl=64 time=0.162 ms
64 bytes from 10.0.3.4: seq=60 ttl=64 time=0.188 ms
64 bytes from 10.0.3.4: seq=61 ttl=64 time=0.153 ms
64 bytes from 10.0.3.4: seq=62 ttl=64 time=0.195 ms
64 bytes from 10.0.3.4: seq=63 ttl=64 time=0.136 ms
64 bytes from 10.0.3.4: seq=64 ttl=64 time=0.176 ms
64 bytes from 10.0.3.4: seq=65 ttl=64 time=0.166 ms
64 bytes from 10.0.3.4: seq=66 ttl=64 time=0.537 ms
64 bytes from 10.0.3.4: seq=67 ttl=64 time=0.404 ms
64 bytes from 10.0.3.4: seq=68 ttl=64 time=0.285 ms
64 bytes from 10.0.3.4: seq=69 ttl=64 time=0.379 ms
64 bytes from 10.0.3.4: seq=70 ttl=64 time=0.194 ms
64 bytes from 10.0.3.4: seq=71 ttl=64 time=0.572 ms
64 bytes from 10.0.3.4: seq=72 ttl=64 time=0.265 ms
64 bytes from 10.0.3.4: seq=73 ttl=64 time=0.160 ms
64 bytes from 10.0.3.4: seq=74 ttl=64 time=0.451 ms
64 bytes from 10.0.3.4: seq=75 ttl=64 time=0.174 ms
64 bytes from 10.0.3.4: seq=76 ttl=64 time=0.179 ms
64 bytes from 10.0.3.4: seq=77 ttl=64 time=0.139 ms
64 bytes from 10.0.3.4: seq=78 ttl=64 time=0.221 ms
64 bytes from 10.0.3.4: seq=79 ttl=64 time=0.174 ms
64 bytes from 10.0.3.4: seq=80 ttl=64 time=0.232 ms
64 bytes from 10.0.3.4: seq=81 ttl=64 time=0.187 ms
64 bytes from 10.0.3.4: seq=82 ttl=64 time=0.180 ms
64 bytes from 10.0.3.4: seq=83 ttl=64 time=0.244 ms
```

Рис. 2.20. Результати після тривалого тестування

Як бачимо, чим більший час тестування, тим сильніше мережа реагує на трафік, який генерується в неї запитами.

Після того, як тестування мережі завершується, можемо бачити що час відгуку приходить до норми:

```
Командная строка
64 bytes from 10.0.3.4: seq=123 ttl=64 time=0.138 ms
64 bytes from 10.0.3.4: seq=124 ttl=64 time=0.140 ms
64 bytes from 10.0.3.4: seq=125 ttl=64 time=0.110 ms
64 bytes from 10.0.3.4: seq=126 ttl=64 time=0.137 ms
64 bytes from 10.0.3.4: seq=127 ttl=64 time=0.175 ms
64 bytes from 10.0.3.4: seq=128 ttl=64 time=0.130 ms
64 bytes from 10.0.3.4: seq=129 ttl=64 time=0.142 ms
64 bytes from 10.0.3.4: seq=130 ttl=64 time=0.138 ms
64 bytes from 10.0.3.4: seq=131 ttl=64 time=0.149 ms
64 bytes from 10.0.3.4: seq=132 ttl=64 time=0.180 ms
64 bytes from 10.0.3.4: seq=133 ttl=64 time=0.148 ms
64 bytes from 10.0.3.4: seq=134 ttl=64 time=0.124 ms
64 bytes from 10.0.3.4: seq=135 ttl=64 time=0.107 ms
64 bytes from 10.0.3.4: seq=136 ttl=64 time=0.132 ms
64 bytes from 10.0.3.4: seq=137 ttl=64 time=0.131 ms
64 bytes from 10.0.3.4: seq=138 ttl=64 time=0.131 ms
64 bytes from 10.0.3.4: seq=139 ttl=64 time=0.112 ms
64 bytes from 10.0.3.4: seq=140 ttl=64 time=0.114 ms
64 bytes from 10.0.3.4: seq=141 ttl=64 time=0.129 ms
64 bytes from 10.0.3.4: seq=142 ttl=64 time=0.118 ms
64 bytes from 10.0.3.4: seq=143 ttl=64 time=0.138 ms
64 bytes from 10.0.3.4: seq=144 ttl=64 time=0.140 ms
64 bytes from 10.0.3.4: seq=145 ttl=64 time=0.107 ms
64 bytes from 10.0.3.4: seq=146 ttl=64 time=0.112 ms
64 bytes from 10.0.3.4: seq=147 ttl=64 time=0.137 ms
64 bytes from 10.0.3.4: seq=148 ttl=64 time=0.142 ms
64 bytes from 10.0.3.4: seq=149 ttl=64 time=0.115 ms
64 bytes from 10.0.3.4: seq=150 ttl=64 time=0.112 ms
```

Рис. 2.21. Результати після вимкнення трафіку

Наступною створимо мережу macvlan. Для цього введемо команду

docker network create -d macvlan --subnet=10.0.2.0/24 --gateway=10.0.2.1 -o parent=eth0.10 network_name4,

якою створюється мережа з драйвером macvlan, підмережею 10.0.2.0/24, шлюзом 10.0.2.1, драйвером Ethernet-хосту eth0 та назвою network_name4.

```
C:\Users\yurka>docker network create -d macvlan --subnet=10.0.2.0/24 --gateway=10.0.2.1 -o parent=eth0.10 network_name4
6a921d8fad075a9bb21291b59af92b8d4a8fc654136f55bfceef309f2a5681f5
```

Рис. 2.22. Результат виконання команди.

Командою

docker network inspect network_name4

переглянемо характеристики мережі.

```
C:\Users\yurka>docker network inspect network_name4
[
    {
        "Name": "network_name4",
        "Id": "6a921d8fad075a9bb21291b59af92b8d4a8fc654136f55bfceef309f2a5681f5",
        "Created": "2022-12-09T22:22:34.0368854Z",
        "Scope": "local",
        "Driver": "macvlan",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.0.2.0/24",
                    "Gateway": "10.0.2.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "parent": "eth0.10"
        },
        "Labels": {}
    }
]

C:\Users\yurka>
```

Рис. 2.23. Характеристики мережі macvlan network_name4

Від’єднаю контейнери від мережі bridge та під’єднаю їх до мережі macvlan.

docker network disconnect 20453e5227d9 ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f

docker network disconnect 20453e5227d9 bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69

docker network disconnect 20453e5227d9 1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56

docker network connect
6a921d8fad075a9bb21291b59af92b8d4a8fc654136f55bfceef309f2a5681f5
ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f

docker network connect
6a921d8fad075a9bb21291b59af92b8d4a8fc654136f55bfceef309f2a5681f5
bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69

docker network connect
6a921d8fad075a9bb21291b59af92b8d4a8fc654136f55bfceef309f2a5681f5
1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56

```
C:\Users\yurka>docker network disconnect 20453e5227d9 ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f

C:\Users\yurka>docker network disconnect 20453e5227d9 bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69

C:\Users\yurka>docker network disconnect 20453e5227d9 1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56
```

```
C:\Users\yurka>docker network connect 6a921d8fad075a9bb21291b59af92b8d4a8fc654136f55bfceef309f2a5681f5 ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f

C:\Users\yurka>docker network connect 6a921d8fad075a9bb21291b59af92b8d4a8fc654136f55bfceef309f2a5681f5 bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69

C:\Users\yurka>docker network connect 6a921d8fad075a9bb21291b59af92b8d4a8fc654136f55bfceef309f2a5681f5 1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56
```

Рис. 2.24-2.25. Результати виконання команд.

Перевіряю мережу командою

docker network inspect network_name4

```
C:\Users\yurka>docker network inspect network_name4
[
    {
        "Name": "network_name4",
        "Id": "6a921d8fad075a9bb21291b59af92b8d4a8fc654136f55bfceef309f2a5681f5",
        "Created": "2022-12-09T22:22:34.0368854Z",
        "Scope": "local",
        "Driver": "macvlan",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "10.0.2.0/24",
                    "Gateway": "10.0.2.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "1e7ac20e4961a58a86d727c54c183a64b18d3af7ac05ee8b6cea2c1b28d3ef56": {
                "Name": "web2",
                "EndpointID": "3652078a85b35568fca0b95734c3ae0a2d2a314eb073167e706fdb56065861c6",
                "MacAddress": "02:42:0a:00:02:04",
                "IPv4Address": "10.0.2.4/24",
                "IPv6Address": ""
            },
            "bda58b5e0f7b1972a5784f3d46162734a1b582dc58b05b23ab77f6b60cf56e69": {
                "Name": "web1",
                "EndpointID": "1b74dbc024cf2dfc08332a9a20d022aa8c6587dbd8f0366ef2d8fe808307bc82",
                "MacAddress": "02:42:0a:00:02:03",
                "IPv4Address": "10.0.2.3/24",
                "IPv6Address": ""
            },
            "ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f": {
                "Name": "upbeat_mirzakhani",
                "EndpointID": "74a458ee6f17aa71fbd6d4fa8d3f1c27ecf5abcb0d4ca90952e034777655e1f3",
                "MacAddress": "02:42:0a:00:02:02",
                "IPv4Address": "10.0.2.2/24",
                "IPv6Address": ""
            }
        },
```

```
        "Options": {
            "parent": "eth0.10"
        },
        "Labels": {}
    }
]
```

Рис 2.26-2.27. Результати введення команд

Контейнери підключені до мережі, то ж спробую зпінгувати контейнер в мережі та ззовні:

docker exec -it bae6e282d540 ping 10.0.2.4

Ззовні контейнер не пінгується, то ж спробую в мережі:

docker exec -it bda58b5e0f7b ping 10.0.2.4

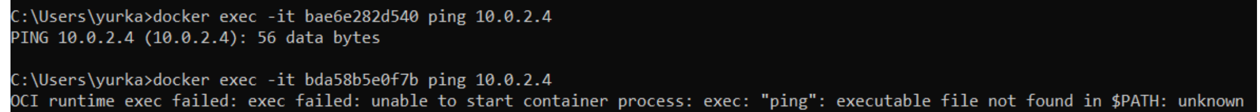

Рис 2.28. Результат введених команд.

При спробі зпінгувати контейнер всередині мережі виникає помилка. Це пояснюється тим, що інтерфейс мережі macvlan працює лише в Linux-пристроях, у Windows та Mac тестування мережі macvlan не є можливим, тож ця частина буде описана теоретично. Для того, щоб можливо було тестувати мережу macvlan на Linux-пристроях, потрібно додатково створити інтерфейс macvlan на хості за допомогою команди ip[8]:

$ sudo ip link add mynet-net link enp0s3 type macvlan mode bridge

Потім треба призначити унікальну IP-адресу інтерфейсу.

Треба перевірити, чи цей IP зарезервований на маршрутизаторі.

$ sudo ip addr add 192.168.2.50/32 dev mynet-net

Треба підключити macvlan інтерфейс.

$ sudo ip link set mynet-net up

Останнім кроком буде вказівка хосту Docker використовувати цей інтерфейс для зв'язку з контейнерами.

Для цього треба додати маршрут до мережі macvlan.

$ sudo ip route add 192.168.2.0/24 dev mycool-net

Тепер, із встановленим маршрутом, хост та контейнери можуть взаємодіяти один з одним.

Можна перевірити маршрути за допомогою ip route.

$ ip route.

Спробуємо під'єднати контейнер до мережі хост:

docker network connect 22df8c8588a26db3d45fe72395158ad113dfe30a3dc888850112acdb8d955d40 ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f

```
C:\Users\yurka>docker network connect 22df8c8588a26db3d45fe72395158ad113dfe30a3dc888850112acdb8d955d40 ee0f23994558bf09e391f7f0e1ea732b824d5c7b54c841b02a0f57c688a2e45f
Error response from daemon: container cannot be disconnected from host network or connected to host network
```

Рис 2.29. Результат введення команди

Помилка з'являється через особливість мережі host, яка не дозволяє вносити в мережу чи виносити з мережі існуючі контейнери. Тому створимо в мережі host нові контейнери:

docker run -it --name=test1 --net host ntop/ntopng

```
Командная строка
C:\Users\yurka>docker run -it --name=test1 --net host ntop/ntopng
Starting redis-server: redis-server.
10/Dec/2022 22:55:09 [Redis.cpp:151] Successfully connected to redis 127.0.0.1:6379@0
10/Dec/2022 22:55:09 [Redis.cpp:151] Successfully connected to redis 127.0.0.1:6379@0
10/Dec/2022 22:55:09 [NtopPro.cpp:311] [LICENSE] No license file found /etc/ntopng.license: reading license from redis
10/Dec/2022 22:55:09 [NtopPro.cpp:483] [LICENSE] Unable to validate license [Empty license file]
10/Dec/2022 22:55:09 [NtopPro.cpp:552] WARNING: [LICENSE] Invalid license [Empty license file]
10/Dec/2022 22:55:09 [NtopPro.cpp:569] WARNING: [LICENSE] ntopng will now run in Enterprise L edition for 10 minutes
10/Dec/2022 22:55:09 [NtopPro.cpp:571] WARNING: [LICENSE] before returning to community mode
10/Dec/2022 22:55:09 [NtopPro.cpp:573] WARNING: [LICENSE] You can buy a permanent license at http://shop.ntop.org
10/Dec/2022 22:55:09 [NtopPro.cpp:575] WARNING: [LICENSE] or run ntopng in community mode starting
10/Dec/2022 22:55:09 [NtopPro.cpp:576] WARNING: [LICENSE] ntopng --community
10/Dec/2022 22:55:10 [Prefs.cpp:1985] ERROR: Too many interfaces (8): discarded br-20453e5227d9
10/Dec/2022 22:55:10 [Prefs.cpp:1986] ERROR: Hint: reset redis (redis-cli flushall) and then start ntopng again
10/Dec/2022 22:55:10 [Prefs.cpp:1985] ERROR: Too many interfaces (8): discarded veth04fd29e
10/Dec/2022 22:55:10 [Prefs.cpp:1986] ERROR: Hint: reset redis (redis-cli flushall) and then start ntopng again
10/Dec/2022 22:55:10 [NetworkInterface.cpp:3278] Cleanup interface dummy
10/Dec/2022 22:55:10 [NetworkInterface.cpp:3278] Cleanup interface lo
10/Dec/2022 22:55:10 [main.cpp:240] Unable to open interface lo [97]: PF_RING not loaded. Falling back to pcap.
10/Dec/2022 22:55:10 [PcapInterface.cpp:96] Reading packets from  [id: 1]
10/Dec/2022 22:55:10 [Ntop.cpp:2589] Registered interface lo [id: 1]
10/Dec/2022 22:55:11 [NetworkInterface.cpp:3278] Cleanup interface services1
10/Dec/2022 22:55:11 [main.cpp:240] Unable to open interface services1 [97]: PF_RING not loaded. Falling back to pcap.
10/Dec/2022 22:55:11 [PcapInterface.cpp:96] Reading packets from  [id: 2]
10/Dec/2022 22:55:11 [Ntop.cpp:2589] Registered interface services1 [id: 2]
10/Dec/2022 22:55:11 [NetworkInterface.cpp:3278] Cleanup interface eth0
10/Dec/2022 22:55:11 [main.cpp:240] Unable to open interface eth0 [97]: PF_RING not loaded. Falling back to pcap.
10/Dec/2022 22:55:11 [PcapInterface.cpp:96] Reading packets from  [id: 3]
10/Dec/2022 22:55:11 [Ntop.cpp:2589] Registered interface eth0 [id: 3]
10/Dec/2022 22:55:11 [NetworkInterface.cpp:3278] Cleanup interface docker0
10/Dec/2022 22:55:11 [main.cpp:240] Unable to open interface docker0 [97]: PF_RING not loaded. Falling back to pcap.
10/Dec/2022 22:55:12 [PcapInterface.cpp:96] Reading packets from  [id: 4]
10/Dec/2022 22:55:12 [Ntop.cpp:2589] Registered interface docker0 [id: 4]
10/Dec/2022 22:55:12 [NetworkInterface.cpp:3278] Cleanup interface docker_gwbridge
10/Dec/2022 22:55:12 [main.cpp:240] Unable to open interface docker_gwbridge [97]: PF_RING not loaded. Falling back to pcap.
10/Dec/2022 22:55:12 [PcapInterface.cpp:96] Reading packets from  [id: 5]
10/Dec/2022 22:55:12 [Utils.cpp:2569] ERROR: Cannot get hw addr for docker_gwbridg
10/Dec/2022 22:55:12 [Ntop.cpp:2589] Registered interface docker_gwbridge [id: 5]
10/Dec/2022 22:55:12 [NetworkInterface.cpp:3278] Cleanup interface veth046fff5
10/Dec/2022 22:55:12 [main.cpp:240] Unable to open interface veth046fff5 [97]: PF_RING not loaded. Falling back to pcap.
10/Dec/2022 22:55:12 [PcapInterface.cpp:96] Reading packets from  [id: 6]
10/Dec/2022 22:55:12 [Ntop.cpp:2589] Registered interface veth046fff5 [id: 6]
10/Dec/2022 22:55:12 [NetworkInterface.cpp:3278] Cleanup interface dm-f4b2da7ee988
10/Dec/2022 22:55:12 [main.cpp:240] Unable to open interface dm-f4b2da7ee988 [97]: PF_RING not loaded. Falling back to pcap.
10/Dec/2022 22:55:12 [PcapInterface.cpp:96] Reading packets from  [id: 7]
10/Dec/2022 22:55:12 [Utils.cpp:2569] ERROR: Cannot get hw addr for dm-f4b2da7ee98
10/Dec/2022 22:55:12 [Ntop.cpp:2589] Registered interface dm-f4b2da7ee988 [id: 7]
10/Dec/2022 22:55:12 [NetworkInterface.cpp:3278] Cleanup interface veth9946a4c
10/Dec/2022 22:55:12 [main.cpp:240] Unable to open interface veth9946a4c [97]: PF_RING not loaded. Falling back to pcap.
```

Рис 2.30. Результат виконання команди

```
10/Dec/2022 22:55:12 [PcapInterface.cpp:96] Reading packets from  [id: 7]
10/Dec/2022 22:55:12 [Utils.cpp:2569] ERROR: Cannot get hw addr for dm-f4b2da7ee98
10/Dec/2022 22:55:12 [Ntop.cpp:2589] Registered interface dm-f4b2da7ee988 [id: 7]
10/Dec/2022 22:55:12 [NetworkInterface.cpp:3278] Cleanup interface veth9946a4c
10/Dec/2022 22:55:12 [main.cpp:240] Unable to open interface veth9946a4c [97]: PF_RING not loaded. Falling back to pcap.
10/Dec/2022 22:55:12 [PcapInterface.cpp:96] Reading packets from  [id: 8]
10/Dec/2022 22:55:12 [Ntop.cpp:2589] Registered interface veth9946a4c [id: 8]
10/Dec/2022 22:55:12 [main.cpp:337] PID stored in file /var/run/ntopng.pid
10/Dec/2022 22:55:12 [Geolocation.cpp:149] Loaded database dbip-city-lite.mmdb [/usr/share/ntopng/httpdocs/geoip//dbip-city-lite.mmdb][ip_version: 6]
10/Dec/2022 22:55:12 [Geolocation.cpp:149] Loaded database dbip-asn-lite.mmdb [/usr/share/ntopng/httpdocs/geoip//dbip-asn-lite.mmdb][ip_version: 6]
10/Dec/2022 22:55:12 [Geolocation.cpp:95] Using geolocation provided by DB-IP (https://db-ip.com)
10/Dec/2022 22:55:13 [HTTPserver.cpp:1408] Found TLS certificate /usr/share/ntopng/httpdocs/ssl/ntopng-cert.pem
10/Dec/2022 22:55:13 [HTTPserver.cpp:1683] Web server dirs [/usr/share/ntopng/httpdocs][/usr/share/ntopng/scripts]
10/Dec/2022 22:55:13 [HTTPserver.cpp:1686] HTTP server listening on 3000
10/Dec/2022 22:55:13 [Utils.cpp:3672] WARNING: Capabilities cap_set_proc error: Operation not permitted
10/Dec/2022 22:55:13 [Utils.cpp:738] WARNING: Unable to retain privileges for privileged file writing
10/Dec/2022 22:55:13 [Utils.cpp:784] User changed to ntopng
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface lo [id: 1]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface lo [id: 1]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface services1 [id: 2]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface services1 [id: 2]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface eth0 [id: 3]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface eth0 [id: 3]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface docker0 [id: 4]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface docker0 [id: 4]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface docker_gwbridge [id: 5]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface docker_gwbridge [id: 5]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface veth046fff5 [id: 6]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface veth046fff5 [id: 6]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface dm-f4b2da7ee988 [id: 7]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface dm-f4b2da7ee988 [id: 7]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface veth9946a4c [id: 8]...
10/Dec/2022 22:55:13 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface veth9946a4c [id: 8]...
10/Dec/2022 22:55:13 [main.cpp:407] Working directory: /var/lib/ntopng
10/Dec/2022 22:55:13 [main.cpp:409] Scripts/HTML pages directory: /usr/share/ntopng
10/Dec/2022 22:55:13 [Ntop.cpp:466] Welcome to ntopng x86_64 v.5.4.221130 (5.4-stable:150715ac4c8dfdac2978c2cd31c5ee081870116c:20221130) - (C) 1998-22 ntop.org
10/Dec/2022 22:55:13 [Ntop.cpp:476] Built on Ubuntu 22.04.1 LTS
10/Dec/2022 22:55:13 [NtopPro.cpp:776] [LICENSE] System Id:     L12030603820DAA8A--OL
10/Dec/2022 22:55:13 [NtopPro.cpp:777] [LICENSE] Edition:       Enterprise L (Bundle)
10/Dec/2022 22:55:13 [NtopPro.cpp:778] [LICENSE] License Type:  Time-Limited [Empty license file] License
10/Dec/2022 22:55:13 [NtopPro.cpp:798] [LICENSE] Validity:      Until Sat Dec 10 23:05:09 2022
10/Dec/2022 22:55:13 [Ntop.cpp:879] Adding 127.0.0.1/32 as IPv4 interface address for lo
10/Dec/2022 22:55:13 [Ntop.cpp:888] Adding 127.0.0.0/8 as IPv4 local network for lo
10/Dec/2022 22:55:13 [Ntop.cpp:879] Adding 192.168.65.4/32 as IPv4 interface address for services1
10/Dec/2022 22:55:13 [Ntop.cpp:888] Adding 192.168.65.4/32 as IPv4 local network for services1
10/Dec/2022 22:55:13 [Ntop.cpp:879] Adding 192.168.65.3/32 as IPv4 interface address for eth0
10/Dec/2022 22:55:13 [Ntop.cpp:888] Adding 192.168.65.0/25 as IPv4 local network for eth0
```

Рис 2.31. Результат виконання команди

```
10/Dec/2022 22:55:13 [Ntop.cpp:879] Adding 172.17.0.1/32 as IPv4 interface address for docker0
10/Dec/2022 22:55:13 [Ntop.cpp:888] Adding 172.17.0.0/16 as IPv4 local network for docker0
10/Dec/2022 22:55:13 [Ntop.cpp:879] Adding 172.18.0.1/32 as IPv4 interface address for docker_gwbridge
10/Dec/2022 22:55:13 [Ntop.cpp:888] Adding 172.18.0.0/16 as IPv4 local network for docker_gwbridge
10/Dec/2022 22:55:13 [Ntop.cpp:910] Adding ::1/128 as IPv6 interface address for lo
10/Dec/2022 22:55:13 [Ntop.cpp:920] Adding ::1/128 as IPv6 local network for lo
10/Dec/2022 22:55:13 [Ntop.cpp:910] Adding fe80::4081:fdff:fe47:c3e3/128 as IPv6 interface address for services1
10/Dec/2022 22:55:13 [Ntop.cpp:920] Adding fe80::4081:fdff:fe47:c3e3/64 as IPv6 local network for services1
10/Dec/2022 22:55:13 [Ntop.cpp:910] Adding fe80::50:ff:fe00:1/128 as IPv6 interface address for eth0
10/Dec/2022 22:55:13 [Ntop.cpp:920] Adding fe80::50:ff:fe00:1/64 as IPv6 local network for eth0
10/Dec/2022 22:55:13 [Ntop.cpp:910] Adding fe80::42:8dff:fee9:ecf7/128 as IPv6 interface address for docker0
10/Dec/2022 22:55:13 [Ntop.cpp:920] Adding fe80::42:8dff:fee9:ecf7/64 as IPv6 local network for docker0
10/Dec/2022 22:55:13 [Ntop.cpp:910] Adding fe80::42:9fff:feb6:c31a/128 as IPv6 interface address for docker_gwbridge
10/Dec/2022 22:55:13 [Ntop.cpp:920] Adding fe80::42:9fff:feb6:c31a/64 as IPv6 local network for docker_gwbridge
10/Dec/2022 22:55:13 [Ntop.cpp:910] Adding fe80::e0db:6dff:fea0:5726/128 as IPv6 interface address for veth046fff5
10/Dec/2022 22:55:13 [Ntop.cpp:920] Adding fe80::e0db:6dff:fea0:5726/64 as IPv6 local network for veth046fff5
10/Dec/2022 22:55:13 [Ntop.cpp:910] Adding fe80::507e:6bff:fefa:90d5/128 as IPv6 interface address for dm-f4b2da7ee988
10/Dec/2022 22:55:13 [Ntop.cpp:920] Adding fe80::507e:6bff:fefa:90d5/64 as IPv6 local network for dm-f4b2da7ee988
10/Dec/2022 22:55:13 [Ntop.cpp:910] Adding fe80::b04a:eff:fe82:653/128 as IPv6 interface address for veth9946a4c
10/Dec/2022 22:55:13 [Ntop.cpp:920] Adding fe80::b04a:eff:fe82:653/64 as IPv6 local network for veth9946a4c
10/Dec/2022 22:55:15 [PeriodicActivities.cpp:109] Started periodic activities loop...
10/Dec/2022 22:55:15 [startup.lua:35] Processing startup.lua: please hold on...
10/Dec/2022 22:55:15 [startup.lua:120] [lists_utils.lua:811] Refreshing category lists...
10/Dec/2022 22:55:16 [startup.lua:120] [lists_utils.lua:439] Updating list 'Abuse.ch URLhaus' [https://urlhaus.abuse.ch/downloads/hostfile/]... OK
10/Dec/2022 22:55:17 [startup.lua:120] [lists_utils.lua:439] Updating list 'Emerging Threats' [https://rules.emergingthreats.net/fwrules/emerging-Block-IPs.txt]... OK
10/Dec/2022 22:55:18 [startup.lua:120] [lists_utils.lua:439] Updating list 'Feodo' [https://feodotracker.abuse.ch/downloads/ipblocklist_recommended.txt]... OK
10/Dec/2022 22:55:18 [startup.lua:120] [lists_utils.lua:439] Updating list 'Feodo Tracker Botnet C2 IP Blocklist' [https://feodotracker.abuse.ch/downloads/ipblocklist.txt]... OK
10/Dec/2022 22:55:19 [startup.lua:120] [lists_utils.lua:439] Updating list 'NoCoin Filter List' [https://raw.githubusercontent.com/hoshsadiq/adblock-nocoin-list/master/hosts.txt]... OK
10/Dec/2022 22:55:19 [startup.lua:120] [lists_utils.lua:439] Updating list 'SSLBL Botnet C2 IP Blacklist' [https://sslbl.abuse.ch/blacklist/sslipblacklist.txt]... OK
10/Dec/2022 22:55:20 [startup.lua:120] [lists_utils.lua:439] Updating list 'ThreatFox' [https://threatfox.abuse.ch/downloads/hostfile/]... OK
10/Dec/2022 22:55:21 [startup.lua:120] [lists_utils.lua:439] Updating list 'URLhaus' [https://urlhaus.abuse.ch/downloads/hostfile/]... OK
10/Dec/2022 22:55:21 [startup.lua:120] [lists_utils.lua:439] Updating list 'dshield 7 days' [https://raw.githubusercontent.com/firehol/blocklist-ipsets/master/dshield_7d.netset]... OK
10/Dec/2022 22:55:23 [startup.lua:120] [lists_utils.lua:739] Category Lists (17717 hosts, 2703 IPs, 0 JA3) loaded in 2 sec
10/Dec/2022 22:55:23 [startup.lua:124] Initializing device polices...
10/Dec/2022 22:55:23 [startup.lua:140] Initializing alerts...
10/Dec/2022 22:55:23 [startup.lua:149] Initializing timeseries...
10/Dec/2022 22:55:23 [startup.lua:212] Fetching latest ntop blog posts...
10/Dec/2022 22:55:24 [startup.lua:216] Completed startup.lua
10/Dec/2022 22:55:24 [PeriodicActivities.cpp:167] Found 10 activities
10/Dec/2022 22:55:24 [NetworkInterface.cpp:3245] Started packet polling on interface lo [id: 1]...
10/Dec/2022 22:55:24 [NetworkInterface.cpp:3245] Started packet polling on interface services1 [id: 2]...
10/Dec/2022 22:55:24 [NetworkInterface.cpp:3245] Started packet polling on interface eth0 [id: 3]...
10/Dec/2022 22:55:24 [NetworkInterface.cpp:3245] Started packet polling on interface docker0 [id: 4]...
10/Dec/2022 22:55:24 [NetworkInterface.cpp:3245] Started packet polling on interface docker_gwbridge [id: 5]...
10/Dec/2022 22:55:24 [NetworkInterface.cpp:3245] Started packet polling on interface veth046fff5 [id: 6]...
10/Dec/2022 22:55:24 [NetworkInterface.cpp:3245] Started packet polling on interface dm-f4b2da7ee988 [id: 7]...
10/Dec/2022 22:55:24 [NetworkInterface.cpp:3245] Started packet polling on interface veth9946a4c [id: 8]...
```

Рис 2.32. Результат виконання команди

та

docker run -it --name=test2 --net host busybox

```
C:\Users\yurka>docker run -it --name=test2 --net host busybox
/ # ip
BusyBox v1.34.1 (2022-11-17 19:59:04 UTC) multi-call binary.
```

Рис 2.33. Результат виконання команди

Перевіримо мережу host командою

docker network inspect 22df8c8588a2

```
C:\Users\yurka>docker network inspect 22df8c8588a2
[
    {
        "Name": "host",
        "Id": "22df8c8588a26db3d45fe72395158ad113dfe30a3dc888850112acdb8d955d40",
        "Created": "2022-12-04T19:03:16.2936171Z",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": []
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "56b9ba7f1e9d98e154cf44aa4e158d7751aad1a9cc0e215bf6a37ccf1b9d2422": {
                "Name": "test1",
                "EndpointID": "561d859d751d8e4722eef849fe8ccf9e973632fce4851c6ccf7517023a06bb85",
                "MacAddress": "",
                "IPv4Address": "",
                "IPv6Address": ""
            },
            "b4ce41d06a042fab65b9fe9808b5fafece17924d450f82f1cb4bbcf717c80e45": {
                "Name": "test2",
                "EndpointID": "bf4d7a97bc8924948c2a30cb43e01c610d37336a1abbd12031aef812a6fd1c55",
                "MacAddress": "",
                "IPv4Address": "",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]

C:\Users\yurka>
```

Рис. 2.34. Результат виконання команди

Як можна помітити, мережа host не видає mac- та ip-адреси своїм контейнерам, також не дозволяє при створенні нового контейнеру всередині мережі вводити ip вручну. Але при створені першого контейнеру він дав сигнал,

що працює в адресі хосту 127.0.0.1 та в порту 6379. Ось це і будемо пінгувати другим контейнером всередині мережі:

docker exec -it b4ce41d06a042fab65b9fe9808b5fafece17924d450f82f1cb4bbcf717c80e45 ping 127.0.0.1:6379

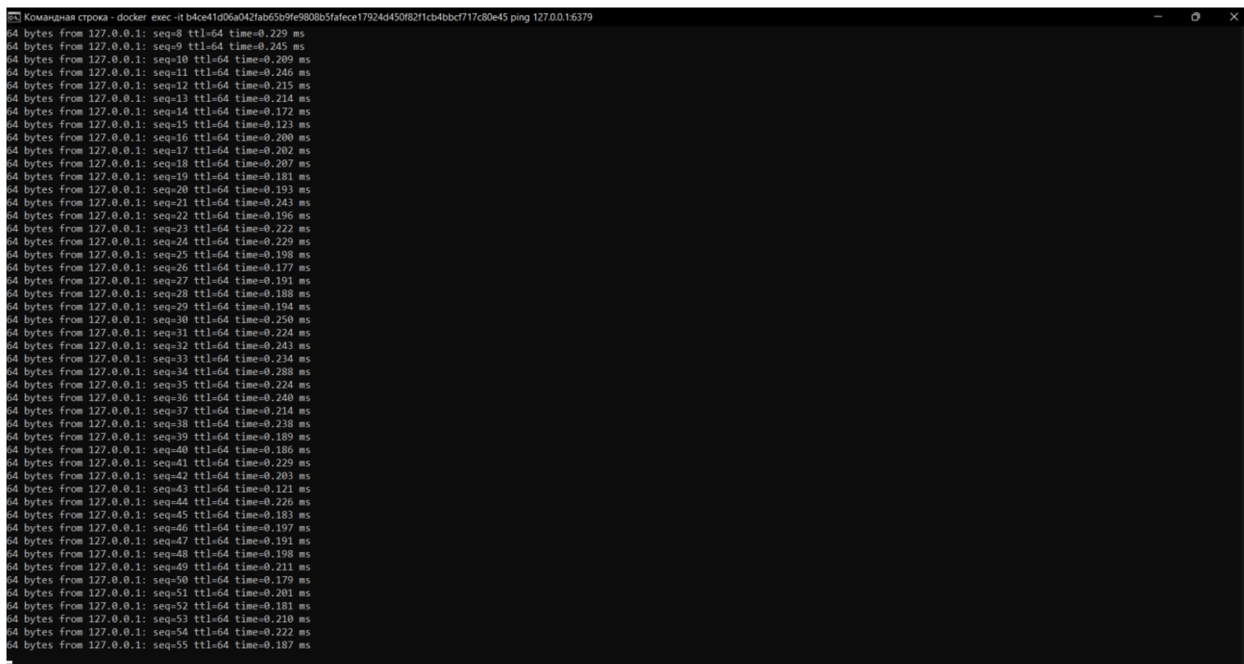

Рис. 2.35. Результат виконання команди

Бачимо, що середній час відгуку контейнера в мережі складає близько 0.200 мс. Спробуємо згенерувати трафік на контейнер:

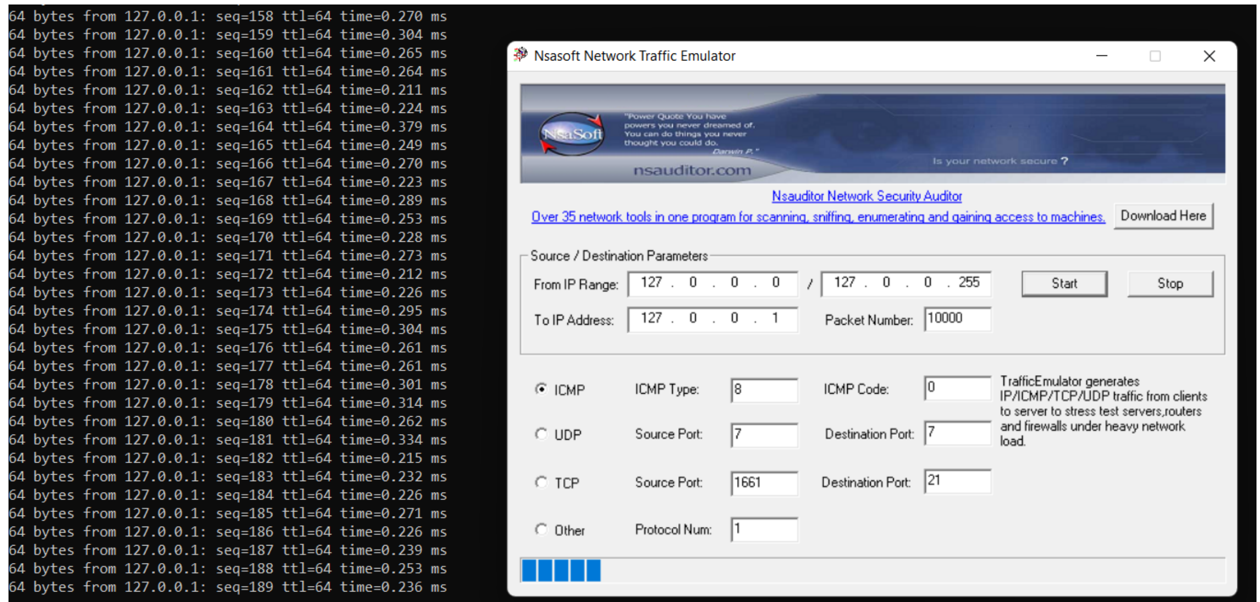

Рис. 2.36. Робота мережі під генерацією трафіку.

В мережі host, як і bridge, підвищується час відгуку контейнерів. Але під час тривалого тестування під генерацією трафіку час відгуку залишається незмінним.

```
Командная строка
64 bytes from 127.0.0.1: seq=358 ttl=64 time=0.303 ms
64 bytes from 127.0.0.1: seq=359 ttl=64 time=0.299 ms
64 bytes from 127.0.0.1: seq=360 ttl=64 time=0.271 ms
64 bytes from 127.0.0.1: seq=361 ttl=64 time=0.200 ms
64 bytes from 127.0.0.1: seq=362 ttl=64 time=0.308 ms
64 bytes from 127.0.0.1: seq=363 ttl=64 time=0.300 ms
64 bytes from 127.0.0.1: seq=364 ttl=64 time=0.291 ms
64 bytes from 127.0.0.1: seq=365 ttl=64 time=0.280 ms
64 bytes from 127.0.0.1: seq=366 ttl=64 time=0.275 ms
64 bytes from 127.0.0.1: seq=367 ttl=64 time=0.221 ms
64 bytes from 127.0.0.1: seq=368 ttl=64 time=0.305 ms
64 bytes from 127.0.0.1: seq=369 ttl=64 time=0.304 ms
64 bytes from 127.0.0.1: seq=370 ttl=64 time=0.288 ms
64 bytes from 127.0.0.1: seq=371 ttl=64 time=0.263 ms
64 bytes from 127.0.0.1: seq=372 ttl=64 time=0.266 ms
64 bytes from 127.0.0.1: seq=373 ttl=64 time=0.262 ms
64 bytes from 127.0.0.1: seq=374 ttl=64 time=0.135 ms
64 bytes from 127.0.0.1: seq=375 ttl=64 time=0.302 ms
64 bytes from 127.0.0.1: seq=376 ttl=64 time=0.316 ms
64 bytes from 127.0.0.1: seq=377 ttl=64 time=0.329 ms
64 bytes from 127.0.0.1: seq=378 ttl=64 time=0.244 ms
64 bytes from 127.0.0.1: seq=379 ttl=64 time=0.241 ms
64 bytes from 127.0.0.1: seq=380 ttl=64 time=0.259 ms
64 bytes from 127.0.0.1: seq=381 ttl=64 time=0.258 ms
64 bytes from 127.0.0.1: seq=382 ttl=64 time=0.233 ms
64 bytes from 127.0.0.1: seq=383 ttl=64 time=0.214 ms
64 bytes from 127.0.0.1: seq=384 ttl=64 time=0.288 ms
64 bytes from 127.0.0.1: seq=385 ttl=64 time=0.229 ms
64 bytes from 127.0.0.1: seq=386 ttl=64 time=0.259 ms
64 bytes from 127.0.0.1: seq=387 ttl=64 time=0.238 ms
64 bytes from 127.0.0.1: seq=388 ttl=64 time=0.302 ms
64 bytes from 127.0.0.1: seq=389 ttl=64 time=0.210 ms
64 bytes from 127.0.0.1: seq=390 ttl=64 time=0.281 ms
64 bytes from 127.0.0.1: seq=391 ttl=64 time=0.297 ms
64 bytes from 127.0.0.1: seq=392 ttl=64 time=0.280 ms
64 bytes from 127.0.0.1: seq=393 ttl=64 time=0.269 ms
64 bytes from 127.0.0.1: seq=394 ttl=64 time=0.255 ms
64 bytes from 127.0.0.1: seq=395 ttl=64 time=0.223 ms
64 bytes from 127.0.0.1: seq=396 ttl=64 time=0.245 ms
64 bytes from 127.0.0.1: seq=397 ttl=64 time=0.285 ms
64 bytes from 127.0.0.1: seq=398 ttl=64 time=0.285 ms
64 bytes from 127.0.0.1: seq=399 ttl=64 time=0.296 ms
64 bytes from 127.0.0.1: seq=400 ttl=64 time=0.236 ms
64 bytes from 127.0.0.1: seq=401 ttl=64 time=0.255 ms
64 bytes from 127.0.0.1: seq=402 ttl=64 time=0.248 ms
64 bytes from 127.0.0.1: seq=403 ttl=64 time=0.260 ms
64 bytes from 127.0.0.1: seq=404 ttl=64 time=0.287 ms
64 bytes from 127.0.0.1: seq=405 ttl=64 time=0.223 ms
64 bytes from 127.0.0.1: seq=406 ttl=64 time=0.306 ms
```

Рис. 2.37. Після 300 запитів під навантаженням час відгуку стабілізувався на середній відмітці близько 0.260 мс.

Для того, щоб вирішити проблему з macvlan, створю мережу ipvlan. Вона за властивостями є подібною на macvlan, але не потребує MAC-адреси для зв'язку в мережу та може працювати в L3. Створю мережу ipvlan командою:

docker network create -d ipvlan --subnet=10.0.6.0/24 --gateway=10.0.6.1 -o parent=eth0 network_name6

```
C:\Users\yurka>docker network create -d ipvlan --subnet=10.0.6.0/24 --gateway=10.0.6.1 -o parent=eth0 network_name6
c2bbea3126fea3093939148b858cc4938f004f689c92d68c2d07dcc26c296acb
```

Рис 2.38. Результат виконання команди

Створю два контейнери всередині мережі:

docker run -it --name=test8 --net network_name6 ntop/ntopng

```
Командная строка - docker  run -it --name=test8 --net network_name6 ntop/ntopng

C:\Users\yurka>docker run -it --name=test8 --net network_name6 ntop/ntopng
Starting redis-server: redis-server.
11/Dec/2022 15:29:13 [Redis.cpp:151] Successfully connected to redis 127.0.0.1:6379@0
11/Dec/2022 15:29:13 [Redis.cpp:151] Successfully connected to redis 127.0.0.1:6379@0
11/Dec/2022 15:29:13 [NtopPro.cpp:311] [LICENSE] No license file found /etc/ntopng.license: reading license from redis
11/Dec/2022 15:29:13 [NtopPro.cpp:483] [LICENSE] Unable to validate license [Empty license file]
11/Dec/2022 15:29:13 [NtopPro.cpp:552] WARNING: [LICENSE] Invalid license [Empty license file]
11/Dec/2022 15:29:13 [NtopPro.cpp:569] WARNING: [LICENSE] ntopng will now run in Enterprise L edition for 10 minutes
11/Dec/2022 15:29:13 [NtopPro.cpp:571] WARNING: [LICENSE] before returning to community mode
11/Dec/2022 15:29:13 [NtopPro.cpp:573] WARNING: [LICENSE] You can buy a permanent license at http://shop.ntop.org
11/Dec/2022 15:29:13 [NtopPro.cpp:575] WARNING: [LICENSE] or run ntopng in community mode starting
11/Dec/2022 15:29:13 [NtopPro.cpp:576] WARNING: [LICENSE] ntopng --community
11/Dec/2022 15:29:33 [boot.lua:24] WARNING: No connectivity detected, ntopng will run in offline mode
11/Dec/2022 15:29:33 [NetworkInterface.cpp:3278] Cleanup interface dummy
11/Dec/2022 15:29:33 [NetworkInterface.cpp:3278] Cleanup interface lo
11/Dec/2022 15:29:33 [main.cpp:240] Unable to open interface lo [97]: PF_RING not loaded. Falling back to pcap.
11/Dec/2022 15:29:33 [PcapInterface.cpp:96] Reading packets from  [id: 1]
11/Dec/2022 15:29:33 [Ntop.cpp:2589] Registered interface lo [id: 1]
11/Dec/2022 15:29:33 [NetworkInterface.cpp:3278] Cleanup interface eth0
11/Dec/2022 15:29:35 [main.cpp:240] Unable to open interface eth0 [97]: PF_RING not loaded. Falling back to pcap.
11/Dec/2022 15:29:36 [PcapInterface.cpp:96] Reading packets from  [id: 2]
11/Dec/2022 15:29:36 [Ntop.cpp:2589] Registered interface eth0 [id: 2]
11/Dec/2022 15:29:36 [main.cpp:337] PID stored in file /var/run/ntopng.pid
11/Dec/2022 15:29:36 [Geolocation.cpp:149] Loaded database dbip-city-lite.mmdb [/usr/share/ntopng/httpdocs/geoip//dbip-city-lite.mmdb][ip_version: 6]
11/Dec/2022 15:29:36 [Geolocation.cpp:149] Loaded database dbip-asn-lite.mmdb [/usr/share/ntopng/httpdocs/geoip//dbip-asn-lite.mmdb][ip_version: 6]
11/Dec/2022 15:29:36 [Geolocation.cpp:95] Using geolocation provided by DB-IP (https://db-ip.com)
11/Dec/2022 15:29:36 [HTTPserver.cpp:1408] Found TLS certificate /usr/share/ntopng/httpdocs/ssl/ntopng-cert.pem
11/Dec/2022 15:29:36 [HTTPserver.cpp:1683] Web server dirs [/usr/share/ntopng/httpdocs][/usr/share/ntopng/scripts]
11/Dec/2022 15:29:36 [HTTPserver.cpp:1686] HTTP server listening on 3000
11/Dec/2022 15:29:36 [Utils.cpp:3672] WARNING: Capabilities cap_set_proc error: Operation not permitted
11/Dec/2022 15:29:36 [Utils.cpp:738] WARNING: Unable to retain privileges for privileged file writing
11/Dec/2022 15:29:36 [Utils.cpp:784] User changed to ntopng
11/Dec/2022 15:29:36 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface lo [id: 1]...
11/Dec/2022 15:29:36 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface lo [id: 1]...
11/Dec/2022 15:29:36 [NetworkInterface.cpp:3055] Started flow user script hooks loop on interface eth0 [id: 2]...
11/Dec/2022 15:29:36 [NetworkInterface.cpp:3092] Started host user script hooks loop on interface eth0 [id: 2]...
11/Dec/2022 15:29:36 [main.cpp:407] Working directory: /var/lib/ntopng
11/Dec/2022 15:29:36 [main.cpp:409] Scripts/HTML pages directory: /usr/share/ntopng
11/Dec/2022 15:29:36 [Ntop.cpp:466] Welcome to ntopng x86_64 v.5.4.221130 (5.4-stable:150715ac4c8dfdac2978c2cd31c5ee081870116c:20221130) - (C) 1998-22 ntop.org
11/Dec/2022 15:29:36 [Ntop.cpp:476] Built on Ubuntu 22.04.1 LTS
11/Dec/2022 15:29:36 [NtopPro.cpp:776] [LICENSE] System Id:     L12030603820DAA8A--OL
11/Dec/2022 15:29:36 [NtopPro.cpp:777] [LICENSE] Edition:       Enterprise L (Bundle)
11/Dec/2022 15:29:36 [NtopPro.cpp:778] [LICENSE] License Type:  Time-Limited [Empty license file] License
11/Dec/2022 15:29:36 [NtopPro.cpp:798] [LICENSE] Validity:      Until Sun Dec 11 15:39:13 2022
11/Dec/2022 15:29:36 [Ntop.cpp:879] Adding 127.0.0.1/32 as IPv4 interface address for lo
11/Dec/2022 15:29:36 [Ntop.cpp:888] Adding 127.0.0.0/8 as IPv4 local network for lo
11/Dec/2022 15:29:36 [Ntop.cpp:879] Adding 10.0.6.2/32 as IPv4 interface address for eth0
11/Dec/2022 15:29:36 [Ntop.cpp:888] Adding 10.0.6.0/24 as IPv4 local network for eth0
```

Рис 2.39. Результат виконання команди

та

docker run -it --name=test9 --net network_name6 busybox

```
C:\Users\yurka>docker run -it --name=test9 --net network_name6 busybox
```

Рис 2.40. Результат виконання команди

Передивимось характеристики новоствореної мережі та контейнерів всередині мережі:

docker network inspect network_name6

Рис 2.41. Характеристики мережі ipvlan

Спробуємо пропінгувати контейнер всередині мережі іншим способом – через інтерфейс busybox, який знаходиться в цій мережі.

ping 10.0.6.2

```
/ # ping 10.0.6.2
PING 10.0.6.2 (10.0.6.2): 56 data bytes
64 bytes from 10.0.6.2: seq=0 ttl=64 time=0.108 ms
64 bytes from 10.0.6.2: seq=1 ttl=64 time=0.035 ms
64 bytes from 10.0.6.2: seq=2 ttl=64 time=0.058 ms
64 bytes from 10.0.6.2: seq=3 ttl=64 time=0.038 ms
64 bytes from 10.0.6.2: seq=4 ttl=64 time=0.041 ms
64 bytes from 10.0.6.2: seq=5 ttl=64 time=0.038 ms
64 bytes from 10.0.6.2: seq=6 ttl=64 time=0.059 ms
64 bytes from 10.0.6.2: seq=7 ttl=64 time=0.035 ms
64 bytes from 10.0.6.2: seq=8 ttl=64 time=0.036 ms
64 bytes from 10.0.6.2: seq=9 ttl=64 time=0.037 ms
64 bytes from 10.0.6.2: seq=10 ttl=64 time=0.036 ms
64 bytes from 10.0.6.2: seq=11 ttl=64 time=0.037 ms
64 bytes from 10.0.6.2: seq=12 ttl=64 time=0.053 ms
64 bytes from 10.0.6.2: seq=13 ttl=64 time=0.043 ms
64 bytes from 10.0.6.2: seq=14 ttl=64 time=0.079 ms
64 bytes from 10.0.6.2: seq=15 ttl=64 time=0.067 ms
64 bytes from 10.0.6.2: seq=16 ttl=64 time=0.042 ms
64 bytes from 10.0.6.2: seq=17 ttl=64 time=0.045 ms
64 bytes from 10.0.6.2: seq=18 ttl=64 time=0.041 ms
64 bytes from 10.0.6.2: seq=19 ttl=64 time=0.053 ms
64 bytes from 10.0.6.2: seq=20 ttl=64 time=0.038 ms
64 bytes from 10.0.6.2: seq=21 ttl=64 time=0.036 ms
64 bytes from 10.0.6.2: seq=22 ttl=64 time=0.082 ms
64 bytes from 10.0.6.2: seq=23 ttl=64 time=0.040 ms
64 bytes from 10.0.6.2: seq=24 ttl=64 time=0.040 ms
64 bytes from 10.0.6.2: seq=25 ttl=64 time=0.038 ms
64 bytes from 10.0.6.2: seq=26 ttl=64 time=0.040 ms
64 bytes from 10.0.6.2: seq=27 ttl=64 time=0.038 ms
64 bytes from 10.0.6.2: seq=28 ttl=64 time=0.042 ms
64 bytes from 10.0.6.2: seq=29 ttl=64 time=0.036 ms
64 bytes from 10.0.6.2: seq=30 ttl=64 time=0.035 ms
64 bytes from 10.0.6.2: seq=31 ttl=64 time=0.120 ms
64 bytes from 10.0.6.2: seq=32 ttl=64 time=0.037 ms
64 bytes from 10.0.6.2: seq=33 ttl=64 time=0.069 ms
64 bytes from 10.0.6.2: seq=34 ttl=64 time=0.042 ms
64 bytes from 10.0.6.2: seq=35 ttl=64 time=0.046 ms
64 bytes from 10.0.6.2: seq=36 ttl=64 time=0.043 ms
64 bytes from 10.0.6.2: seq=37 ttl=64 time=0.056 ms
64 bytes from 10.0.6.2: seq=38 ttl=64 time=0.104 ms
64 bytes from 10.0.6.2: seq=39 ttl=64 time=0.056 ms
64 bytes from 10.0.6.2: seq=40 ttl=64 time=0.032 ms
64 bytes from 10.0.6.2: seq=41 ttl=64 time=0.038 ms
64 bytes from 10.0.6.2: seq=42 ttl=64 time=0.040 ms
64 bytes from 10.0.6.2: seq=43 ttl=64 time=0.041 ms
64 bytes from 10.0.6.2: seq=44 ttl=64 time=0.040 ms
64 bytes from 10.0.6.2: seq=45 ttl=64 time=0.046 ms
```

Рис 2.42. Результати роботи мережі

Як бачимо, в мережі ipvlan контейнери пінгуються та з ними можна працювати в середовищі Docker Desktop. Спробуємо тепер згенерувати трафік для того щоб навантажити мережу:

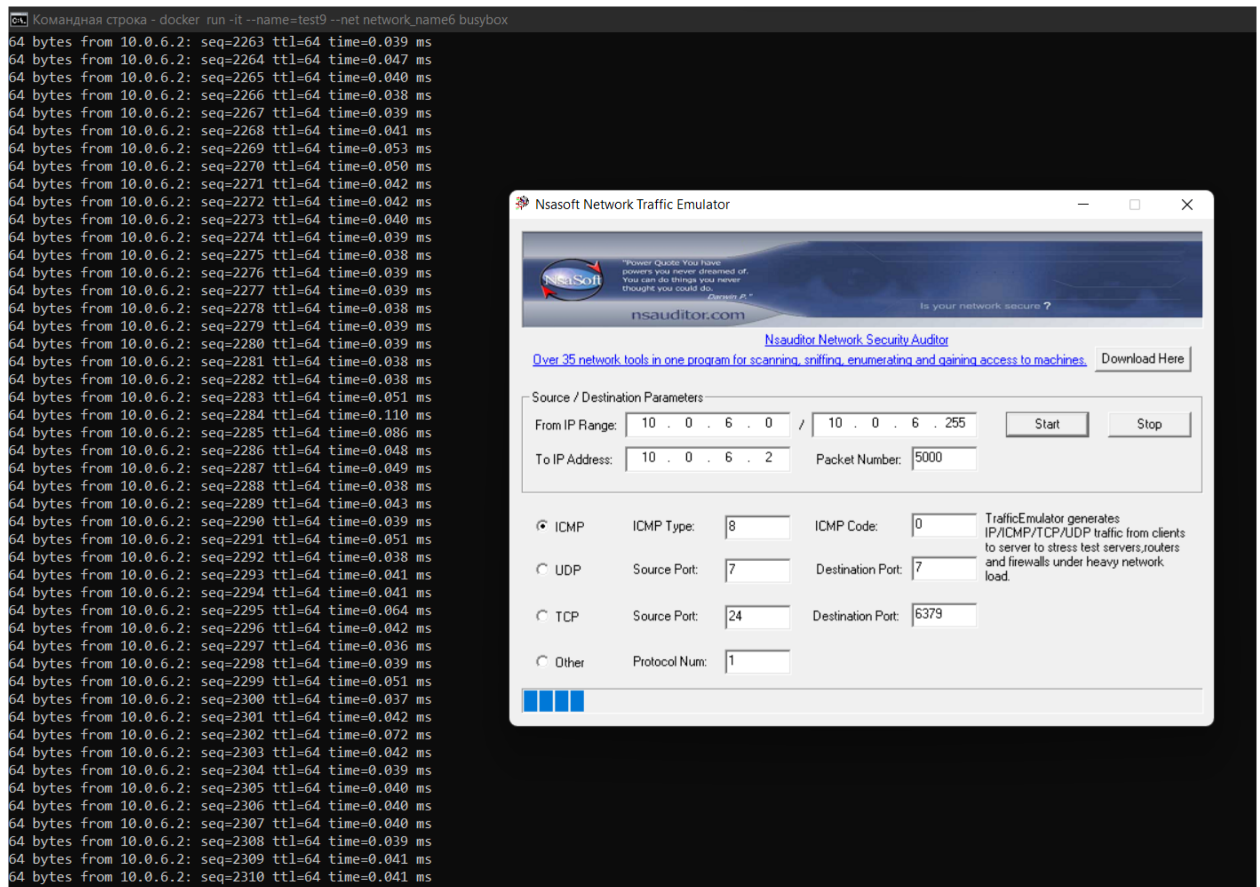

Рис 2.43. Результати відгуку мережі під навантаженням

Як можемо помітити, вхідне навантаження майже ніяк не відображається на часі відгуку контейнера на запит, просадки виникають лише зрідка. Це пояснюється тим, що ipvlan це доволі нова мережева технологія, яка не потребує інтерфейсу мосту та пересилає пакети напряму в віртуальний інтерфейс, тому швидкодія цієї мережі краща за інші.

Висновки

Було проведенно дослідження мереж Docker у середовищі Docker Desktop. Для цього були завантажені образи, з образів зібрані контейнери, були сконфігуровані мережі для дослідження. Після проведених налаштувань та тестів, найкращим типом мережі для використання у середовищі Windows Docker Desktop виявився тип мережі ipvlan.

РОЗДІЛ 3.

РОЗРОБКА СТАРТАП-ПРОЄКТУ

3.1. Інформаційна карта проєкту.

Таблиця 3.1.

Інформаційна карта проєкту

1. Назва проєкту	Адміністрування мереж та контейнерів для потреб розробників
2. Автори проєкту	Казарін Юрій Вікторович
3. Коротка анотація	Мета даного стартапу полягає в організації мереж Docker для користувачів, які не мають власних девайсів для органазації мереж під свої сервіси, але котрі бажають працювати зі своїми сервісами, щоб вони були ізольовані, безпечні та легко конфігурованими.
4. Термін реалізації проєкту	7 місяців

Продовження Таблиці 3.1

5. Необхідні ресурси	Фінансові ресурси
	1. Зарплати співробітників за місяць: • розробник – 20 000 грн; • спеціаліст по маркетингу – 17 500 грн; • адміністратор – 15 000 грн. 2. Оренда та обслуговування приміщення – 140 000 грн. 3. Оформлення торгової марки – 10 000 грн 4. Оформлення авторського плану – 5 000 грн. Всього: 522 500 грн
	Матеріальні ресурси
	1. Ноутбук Acer Nitro 5 AN515-55-56 (2 шт.) – 60 000 грн. 2. Маршрутизатор TP-LINK Archer A64 – 1 399 грн. Всього: 61 399 грн.
	Інтелектуальні ресурси
	Наукові статті, підручники та інше навчально-методичне забезпечення.
	Всього: 583 899 грн

Продовження Таблиці 3.1

6. Опис проблеми, яку вирішує проєкт	Сервіс зі створення та адміністрування Docker-сервісами дозволить інтернет-адміністраторам та бізнесу не возитись самостійно з освоєнням Docker-технології для запуску власних сервісів, а дозволить перекласти роботу на спеціалістів фірми.
7. Головні цілі та завдання проєкту	Головна ціль стартап проєкту – запустити сервіс з обслуговування клієнтських Docker-архітектур. Завдання проєкту: 1. Створення достатніх потужностей для обслуговування великої кількості клієнтів; 2. Створення можливості створювати і обслуговувати будь-які архітектури клієнтів. 3. Створення сучасного, зручного інтерфейсу. 4. Набрати перших 500 активних клієнтів.
8. Очікувані результати	Сервіс надасть клієнтам можливість створити будь-які архітектури для їхніх потреб.

3.2. Формування команди стартапу.

Таблиця 3.2.

Роль кожного спеціаліста

Спеціальність	Роль
Розробник	Генератор ідеї, виконавець
Маркетолог	Дипломат
Адміністратор	Організатор

Таблиця 3.3.

Поставлені завдання та час на їх виконання

	Завдання	Час
1	Аналіз методів реалізації системи	1
2	Розробка Технічного завдання	1,25
3	Розподіл ролей проєкту	0,25
4	Розробка системи	5
5	Тестування системи	2
6	Запровадження системи	0,5
7	Пошук інвесторів	2
8	Маркетингова компанія	2
9	Аналітика	1

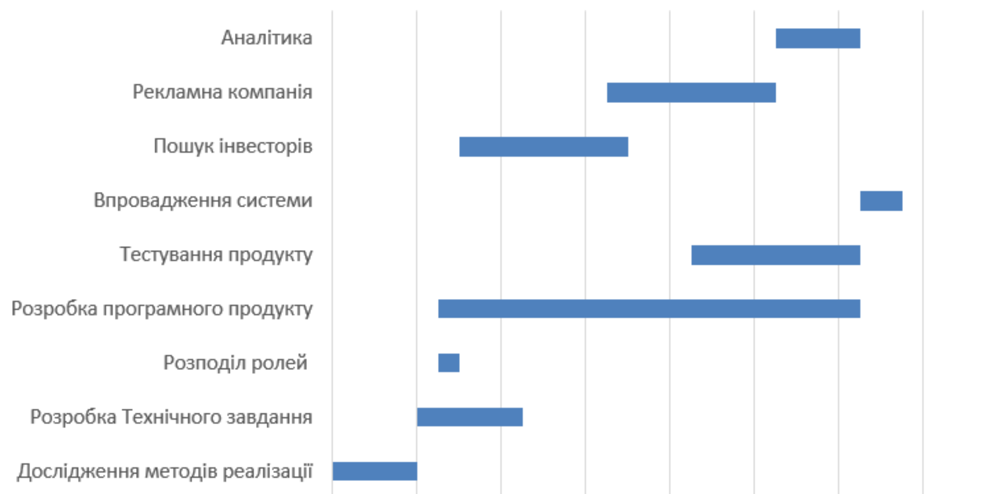

Рис. 4.1. Графік розподілення часу

Таблиця 3.4.

Визначення важливості факторів щодо їх вкладу у створення та реалізацію інноваційного проєкту

Фактор	Вага (важливість)
Ідея	5
Підготовка бізнес плану	7
Компетентність	7
Співпраця та ризики	7
Обов'язки	7

Таблиця 3.5.

Оцінювання особистого внеску кожного партнера у створення та реалізацію стартапу

Фактор	Розробник	Адміністратор	Маркетолог	Разом
Ідея	20	5	0	
Підготовка бізнес плану	10	0	50	

Таблиця 3.5.

Оцінювання особистого внеску кожного партнера у створення та реалізацію стартапу (продовження)

Компетентність	30	25	25	
Залученість і ризики	40	30	10	
Обов'язки	40	35	65	
Разом	140	95	150	385
Процент	36,4	24,7	38,9	100,0

3.3. Розроблення ринкової стратегії проєкту.

Таблиця 3.6.

Вибір цільових груп потенційних споживачів

№ п/п	Профіль цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Попит в межах цільової групи (сегменту)	Інтенсивність конкурен ції в сегменті	Простота входу у сегмент
1.	Інтернет-сфера	Висока, бо адміністратори інтернет-сайтів та сервісів потребують якісного забезпечення.	50%	Невелика, бо технологія відносно нова і затребувана.	Середня.

Продовження Таблиці 3.6

2	Бізнес сфера	Висока, бо цей метод спрощує ведення певних аспектів бізнесу.	50%		Середня.

Таблиця 3.7.

Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції.	Базова стратегія розвитку
1.	Створення сучасного сервісу для створення і адміністрування Docker-мережі та контейнерів з певним наповненням.	Стратегія концентрованого маркетингу	Якісний сервіс, безпека користуванням.	Диференціація.

Таблиця 3.8.

Визначення стратегії конкурентної поведінки

№ п/п	Чи є проєкт «першопро-хідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні риси товару конкурента?	Стратегія конкурентної поведінки
1.	Так, новизна технології.	Так, бо ми можемо надавати якісне адміністрування та обслуговування.	Ні.	Повністю захопити сегмент користувачів за відсутності конкурентів.

Таблиця 3.9.

Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами
1.	Безпека	Надійність, стійкість системи	Використання Docker-технології
2.	Дистанційність	Повне адміністрування системи нашим адміністратором	Можливість замовнику не впливати

3.4. Аналіз ринкових можливостей

Таблиця 3.10.

Попередня характеристика потенційного маркету стартап-проєкту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	2
2	Загальний обсяг продажу реклами, грн/ум.од	50грн*500реклам. =25000 грн в місяць
3	Динаміка ринку	Зростає
4	Наявність обмежень для входу	Відсутні
5	Специфічні вимоги до сертифікації	Можливість створення, підключення та використання будь-якої мережі, безперебійна робота та адміністрування, можливість створити та під'єднати будь-який контейнер з будь-яким додатком або сервісом.
6	Норма рентабельності в галузі, %	70%

Таблиця 3.11.

Характеристика потенційних клієнтів стартап-проєкту

№ п/п	Потреба, яка формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
	Обслуговування та адміністрування Docker-мереж для сервісів	Інтернет-адміністратори та бізнес	Відсутні	Легкий доступ Широкий функціонал Підтримка та покращення продукту

Таблиця 3.12.

Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Конкуренти	Поява конкурентів з більшим штатом та капіталом	Запропонувати більш досконалу підтримку, розширити можливості
2.	Слабка рекламна кампанія	Слабке розповсюдження реклами та слабке зацікавлення цільової аудиторії	Вдосконалити власну рекламну та інтерактивну кампанію
3.	Неякісне обслуговування	Неякісно проведене обслуговування внаслідок відсутності досвіду управління компаніями у засновників	Консультуватися з інвесторами щодо можливостей управління та прийняття рішень

Таблиця 3.13.

Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Нові типи різноманітних питань до знайдених іменованих сутностей	Зростання потужності системи	Витрата часу та грошей на вдосконалення розробки
2.	Відсутність конкурентів на вітчизняному ринку	Вільний ринок	Можливість швидкого розвитку
3.	Консультація з потенційними клієнтами	Можна дізнатися який сервіс користувачі хочуть бачити	Створення сервісу пристосованого до користувача

Таблиця 3.14.

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Чиста	Відсутність конкурентів на вітчизняному ринку	Захищеність від конкуренції
Внутрішньогалузева	Конкуренція спостерігається в сфері інтернет-адміністрування	Впровадження якісного та безпечного сервісу.

Продовження Таблиці 3.14

Товарно-родова	Конкуренція між сервісами різного виду	Зосередження на вдосконалені специфічного сервісу
Цінова	Конкуренція проводиться за рахунок вдосконалення якості продукту та зменшення цін	Зменшення цін
Немарочна	Роль торгової марки незначна	Заохочення клієнтів якістю товару, а не брендом.

Таблиця 3.15.

Порівняльний аналіз сильних та слабких сторін

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-замінників у порівнянні зі стартапом						
			–3	–2	–1	0	+1	+2	+3
1.	Немає аналогів	20							+
2.	Ціна	15						+	
3.	Потреби споживачів	20							+

Таблиця 3.16.

SWOT - аналіз стартап-проєкту

Сильні сторони: даний сервіс надає можливість клієнту отримати налаштовану Docker-архітектуру під постійним адмініструванням.	Слабкі сторони: компанія є «новачком» на ринку, тому не має репутації
Можливості: • Створення власної архітектури • Під будь-яку задачу • Відсутність аналогів	Загрози: Хоч і може надавати широкий спектр сервісу, але існує для вузького кола людей.

Для альтернативного ринкового впровадження протягом 3-7 місяців скупляти рекламу на популярних сайтах для приваблення нових клієнтів. Також в певних соціальних мережах будуть створенні аккаунти компанії з описом сервісу, відповідями на часті питання та іншим контентом для приваблювання нових користувачів.

Таблиця 3.17.

Альтернативи ринкового впровадження стартап-проєкту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Спеціалізовані бізнес- так інтернет-сайти	30%	3-6 місяців

3.5. Реалізація стартап-проєкту

Таблиця 3.18.

Графік реалізації

№ з/п	Етапи реалізації	Період реалізації проєкту						
		0-й рік				1-й рік	2-й рік	3-й рік
		1-й кв.	2-й кв.	3-й кв.	4-й кв.			
1.	Проведення досліджень та інших робіт	+						
2.	Розробка ТЗ та ТЕО	+						
3.	Проєктування сервісу	+	+	+	+			
4.	Створення стартапу			+	+			
5.	Юридична робота з оформленням, отримання документів				+	+		
6.	Оренда робочих приміщень, офісу і т.д.		+	+				
7.	Придбання обладнання	+						

Продовження Таблиці 3.18.

8.	Першочергові маркетингові дослідження	+	+	+				
9.	Впровадження системи				+			
10.	Організація матеріальних ресурсів	+						
11.	Рекламна кампанія				+			
12.	Підтримка продукції					+	+	+

Таблиця 3.19.

Планова потреба у виробничих площах

№ з/п	Тип приміщення (будівлі, ділянки, споруди)	Кількість одиниць	Площа, кв. м	Вимоги до приміщення (будівлі, ділянки, споруди)	Умови надання	Вартість, тис. грн.
1.	Офісне приміщення	1	60	Освітлення, температурний режим відповідно до норм ДСТУ, інтернет	Оренда	15
Разом		—	—	—	—	90

Таблиця 3.20.

Планова потреба у виробничому обладнанні та устаткуванні

№ з/п	Вид обладнання (устаткування, пристрою)	Тип (модель)	Виробник обладнання (устаткування, пристрою)	Терміни постачання	Вартість, тис. грн.
1.	Ноутбук	Acer Nitro 5 AN-515-55-56	Acer	3 місяці	30 000 грн
2.	Ноутбук	Acer Nitro 5 AN-515-55-56	Acer	3 місяці	30 000 грн
Разом		—	—	—	60 000 грн

Таблиця 3.21.

Планова вартість нематеріальних активів

№ з/п	Вид активів	Активи, що можуть бути віднесені до даного виду	Вартість, тис. грн.
1.	Права користування майном	Право на оренду приміщень	15

Таблиця 3.22.

Плановий обсяг виробництва продукції стартап-проєкту

Вид продукції	Одиниця виміру	Обсяги виробництва за період		
		1-й рік	2-й рік	3-й рік
Послуга або сервіс	Клієнт	500	5 000	50 000

Таблиця 3.23.

Планова потреба та витрати на персонал

№ з/п	Категорія персоналу	Чисель-ність	Заробітна плата, тис грн. на місяць	Відрахування на соціальні заходи, тис грн. на місяць	Витрати на оплату праці за період, тис. грн.		
					1-й рік	2-й рік	3-й рік
1.	Генератор ідеї	1	20	1	240	480	720
2.	Реалізатор	1	15	1	180	360	540
4.	Дипломат	1	17.5	1	210	420	630
Разом		3	52.5	3	630	1260	1890

Таблиця 3.24.

Загальні початкові витрати проєкту

№ з/п	Стаття витрат	Обсяги витрат в 0-й рік, тис. грн.
1.	Проведення науково-дослідницьких робіт	-
2.	Розробка ТЗ і ТЕО	-
3.	Робоче проєктування та тестування	522
4.	Витрати на оренду приміщень	90
5.	Витрати на придбання обладнання	61.5
6.	Витрати на впровадження системи	-
7.	Витрати на придбання нематеріальних активів	15
8	Витрати на оплату інтернету	5
9.	Витрати на перед виробничі маркетингові дослідження і створення збутової мережі	-

Продовження Таблиці 3.24.

10.	Витрати, пов'язані з діяльністю персоналу	-
Разом		683.9

Таблиця 3.25.

Планові загальногосподарські витрати

№ з/п	Стаття витрат	Витрати за період, тис. грн.		
		1-й рік	2-й рік	3-й рік
1.	Витрати на оренду приміщення	180	180	180
2.	Витрати на обладнання	62.5	-	-
3.	Витрати на придбання нематеріальних активів	15	-	-
4.	Витрати на персонал	630	630	630
5.	Витрати на зв'язок	9	10	11
6.	Витрати на просування та рекламу	25	30	40
7.	Оплата юридичних послуг	10	15	20
8.	Податкові платежі (земельний, комунальний податки, інші)	10	15	20
Разом		935.5	880	901

Висновок:

В цьому розділі був розроблений стартап-проєкт та проведений аналіз можливого сервісу для для надання послуг адміністрування з маркетингової сторони. Уся необхідна інформація була надана в повній мері.

Було виявлено, що окрім інтернет-адміністраторів, користуватись сервісом зможуть і бізнес-ланки. Так як сервіс має на меті надійність, якісне обслуговування та широкий вибір послуг, то він буде затребуваний.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

Як було сказано в роботі, Docker – це дійсно нова технологія, яка стрімко розвиваються, все більше людей відмовляються від віртуалізації на користь контейнеризації. Дійсно, переваги цієї технології є доволі вагомими для того, щоб проходити повз неї.

Дана робота присвячена дослідженню мереж Docker, їх побудові, аналізу, порівнянню, тестуванню та пошуку оптимальності. Аналіз проблеми показав, що такі порівняння дійсно затребувані, бо все ще людям важко розібратись, яку мережу їм вибирати під яку задачу, а з появою нових рішень та можливостей взагалі не намагаються їх вивчати.

В ході роботи було досліджені мережі Docker та в ході тестування знайдена оптимальна мережа. Цією мережею виявилась ipvlan. Вона показала найменший час відгуку без навантаження та майже непохитність під генерацією трафіку. Це стало можливо завдяки новітнім фічам, які реалізовані в специфікації мережі. Інші протестовані мережі (bridge та host) показали приблизно однаковий гірший результат. Але він не критичний та для своїх задач цілком годяться. Мережу host можна використати для того, щоб контейнер та docker-хост не були ізольовані між собою, але були ізольовані від іншого середовища, а мережу bridge для того, щоб автономні контейнери могли взаємодіяти між собою в різних мережах.

Також був складен стартап-проєкт. В якості предмету стартапу було визначено сервіс, який надає послуги зі створення певного docker-сервісу, його адміністрування та обслуговування. Спеціалісти можуть його створити, забезпечити захист та безперебійну роботу для того, щоб сервіси та додатки кожного клієнта працювали без нарікань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://habr.com/ru/post/258443/
2. https://habr.com/ru/company/flant/blog/327338/
3. https://habr.com/ru/company/flant/blog/331188/
4. https://habr.com/ru/post/253877/
5. https://vps.ua/blog/docker-and-linux-containers/
6. https://habr.com/ru/post/310460/
7. https://habr.com/ru/company/ruvds/blog/439978/
8.https://itsecforu.ru/2022/02/01/%F0%9F%90%B3-%D0%BA%D0%B0%D0%BA-%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D1%82%D1%8C-%D0%B8-%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D1%8C-%D1%81%D0%B5%D1%82%D1%8C-macvlan-%D0%B2-docke/
9. https://itisgood.ru/2019/10/29/objasnenie-koncepcii-setej-v-docker/
10. http://hicu.be/macvlan-vs-ipvlan
11. https://forum.ubuntu.ru/index.php?topic=311386.0
12. https://linux-notes.org/rabota-s-setju-networking-v-docker/
13. https://rohanzi.gitlab.io/balberin-clouds/project/networks/