Вінницький національний технічний університет
Факультет інфокомунікацій, радіоелектроніки та наносистем
Кафедра телекомунікаційних систем та телебачення
Освітньо-кваліфікаційний рівень магістр
Галузь знань 17– Електроніка та телекомунікації
(шифр і назва)
Спеціальність 172 – Телекомунікації та радіотехніка
(шифр і назва)
Освітня програма Телекомунікаційні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедри ТКСТБ
к.т.н., професор Г.Г. Бортник

“___” _____2019 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гринчуку Владиславу Вікторовичу .
(прізвище, ім’я, по батькові)

1. Тема роботи Підвищення інформаційної безпеки Docker мереж
_____
керівник роботи Войцеховська Олена Валеріївна, канд. техн. наук, доцент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “02” 10 2019 року № 254
2. Строк подання студентом роботи 02 грудня 2019 року

3. Вихідні дані до роботи _Хмарна мережа на основі технології контейнеризацій Docker, Операційна система Linux/Ubuntu_____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Архітектурні особливості Docker мереж , Оцінка захищеності Docker мереж, Моделювання ефективності функціонування мережі, Покращення інформаційної безпеки Docker мережі, Економічна частина, Охорона праці та безпека в надзвичайних ситуаціях

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
Код програми для покращення інформаційної безпеки Docker мереж, Схема для порівняння технологій контейнеризації та віртуалізації, Архітектура технології Docker, Рівнева структура Docker образу, Структура шарів Docker контейнера, Граф станів вузла елементу Docker мережі

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Технічна частина	Войцеховска О.В., доцент каф. ТКСТБ		
Економічна частина	Ратушняк О. Г. доцент каф. ЕПВМ		
Охорона праці та безпека в надзвичайних ситуаціях	Березюк О.В. ктн, доцент		

7. Дата видачі завдання вересня 2018 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка технічного завдання	06.09.2019р.	
2.	Техніко-економічне обґрунтування розробки	13.09.2019р.	
3.	Аналіз архітектури Docker мереж	04.10.2019р.	
4.	Моделювання ефективності функціонування мережі	25.10.2019р.	
5.	Оцінка захищеності Docker мереж	08.11.2019р.	
6.	Аналіз економічної ефективності розробки	15.11.2019р.	
7.	Аналіз безпеки життєдіяльності, цивільний захист	22.11.2019р.	
8.	Оформлення пояснювальної записки та графічної частини	29.11.2019р.	
9.	Нормоконтроль МКР	02.12.2019р.	
10.	Попередній захист МКР, рецензування МКР	06.12. 2019р.	
11.	Захист МКР ДЕК	18.12. 2019р.	

Студент _____ Гринчук В.В.
(підпис)

Керівник роботи _____ Войцеховська О.В.
(підпис)

ВСТУП

Актуальність. Чисельність населення нашої планети поступово зростає і вже перевищила 7 мільярдів людей. Зростають і інформаційні потреби населення, а одночасно активно розвиваються технології «Інтернету речей» (IoT-пристроїв). За прогнозами фахівців щомісячний обсяг переданої інформації до 2020 року буде вимірюватися в зеттабайт (1 · +1021 байт). Чисельний зростання кількості абонентів і все більш об'ємні запити до інформаційного сервісу постійно вимагає зростання швидкості передачі інформації в інфотелекоммунікаційному просторі.

Сучасний стан інформаційних технологій дозволяє створювати додатки використовуючи і поєднуючи різні технології та бібліотеки. Часом їх стає настільки багато, що з'являються проблеми зв'язані з їх розповсюдженням на доставкою до замовника чи покупця.

Перша з них — це передача продукту клієнту. Припустимо у вас є серверний проект, який вже завершений і тепер його необхідно передати користувачеві. Для цього можна приготувати багато різних файлів, скриптів або ж написати інструкцію по установці. Після цього витрачається багато часу на вирішення проблем клієнта на зразок: «у мене нічого не працює», «ваш скрипт впав на середині - що тепер робити», «я переплутав порядок кроків в інструкції і тепер не можу йти далі» і т. п

Друга — тиражування. Якщо потрібно розгорнути для одного, або, навіть, декількох клієнтів не одну, не дві, а 50 — 100 машин. Затрачати час на їх налаштування та вирішувати проблеми окремо не є дуже вдалою практикою.

Третя — перевикористання. Наприклад, маємо 3 додатки які використовують однаковий стек технологій java-tomcat-nginx-postgre. Для того, щоб запустити новий додаток на новому сервері, потрібно з нуля встановити на налаштувати увесь цей стек.

Зазвичай ці проблеми вирішуються скриптом. Для цього у файл пишуться всі необхідні команди для встановлення ПЗ, які виконуються одна за одною. Це
дещо вирішує проблеми, але на заздалегідь налаштованих серверах щось може піти не так або банально система застаріла та не буде підтримувати певні рядки скрипта, що вимагатиме додаткового втручання зі сторони розробника.

Хмарні сервіси також певною мірою вирішують проблеми, можна створити з нього образ і розповсюдити по іншим серверам. Але і тут є vendorlock-in, що не дозволить запуск на інших сервісах окрім обраного, що може не сподобатись деяким клієнтам. Також вони не надто швидкі.

Навіть, найсучасніші сервіси набагато відстають у продуктивності від стаціонарних.

Віртуальні машини — одне із найкращих рішень, але вони теж мають деякі недоліки. Розмір який має образ віртуальної машини є досить великим, що не дуже подобається користувачеві. Також цей образ потрібно скачувати кожен раз, коли робляться зміни у проекті з початку. До цього він складне управління спільним використанням серверних ресурсів - не всі віртуальні машини взагалі підтримують спільне використання пам'яті або CPU.

Останнє і найсучасніше вирішення цих проблем є контейнеризація в даному випадку — докер контейнеризація. Подібно віртуальній машині Docket запускає свої процеси у власній, заздалегідь налаштованої операційній системі.

Але при цьому всі процеси докера працюють на фізичному хост сервері ділячи
всі процесори і всю доступну пам'ять з усіма іншими процесами, запущеними в серверній системі. Підхід, який використовується докер знаходиться посередині між запуском всього на фізичному сервері та повної віртуалізації, пропонованої віртуальними машинами. Цей підхід називається контейнеризацією.

Аналіз останніх досліджень. Дослідженням ММД приділено велику увагу в роботах як вітчизняних, так і зарубіжних вчених усього світу, таких як Бистров Р.П., Соколов А. В., Чеканов Р.Н., Яковлєв О.І., Калмиков Ю.П, Титов З .В., Андрєєв Г.А., Паршин В. В., Третьяков М. Ю., Кошелев М. А., Павельєв В.А., Хаміни Д.В., Зражевський А.Ю., Красюк В.Н ., Anderson CR, Rappaport TS, Alejos A., Sanchez M.G., Frenzel L., Pi Z., Khan F., Cuinas I., Doann C.H., Xu H. і багатьох інших. Моделювання радіоканалів присвячені роботи Кловського Д.Д., Самойлова А.Г., Галкіна А.П., Шінакова Ю.С., Маркова В.В., Іванова А.П., Єрмолаєва В.Т., Erceg V., Schumacher L., Watterson SS, Jroshek JR, Dtnsema V.D., Bello P., Son V.V. та інших[1-8].

Мета та постановка задачі. Метою магістерської дипломної роботи є покращення інформаційної безпеки Docker мереж

Для досягнення поставленої мети потрібно вирішити наступні задачі:

- виконати аналіз і визначити основні загрози безпеки в Docker мережах;
- розробити математичну модель надійності Docker мережі;
- запропонувати практичні рекомендації щодо конфігурування налаштувань інформаційної безпеки Docker мережі;
- розробити програмне забезпечення для перевірки правильності конфігурації Docker мережі.

Предметом дослідження інформаційна безпека мереж, створених на архітектурі Docker.

Об'єктом дослідження особливості конфігурації Docker мереж.

Методи дослідження. При вирішенні поставлених завдань використовувалися методи, засновані на основних рекомендація щодо інформаційної безпеки систем, теорії побудови мереж, методів теорії ймовірностей і математичної статистики.

Наукова новизна магістерської дипломної роботи полягає в наступному:

1. У повному обсязі описано основні види вразливостей, присутніх у Docker мережах.

2. Запропоновано моделі для визначення надійності Docker мереж.

Апробація результатів роботи. Основні ідеї роботи доповідались і обговорювались на І Міжнародній науково-технічній конференції "Сучасні проблеми інфокомунікацій, радіоелектроніки та наносистем" і на науковій конференції ВНТУ у 2019 році.

Структура і обсяг роботи. Магістерська дипломна робота складається зі анотації, вступу, шістьох розділів, висновків, списку використаних джерел із 40 найменувань і шістьох додатків. Обсяг роботи 117 сторінки, рисунків, 15 таблиць, 7 додатків.

АРХІТЕКТУРНІ ОСОБЛИВОСТІ DOCKER МЕРЕЖ

1.1 Порівняння класичних віртуальний машин з контейнерами

При проектуванні хмарної інфраструктури, технологія віртуалізації набула широкого розповсюдження завдяки ряду переваг пов'язаних з можливістю запускати на одному сервері багато різних операційних систем, повністю ізольованих одна від одної та широким набором програмних продуктів для конфігурування [1]. Однак, наряду з класичними віртуальними машинами, широку популярність отримала технологія контейнеризації, яка дозволяє створювати менш ресурсозатратну та більш економічну інфраструктуру та володіє кращим потенціалом для розширення.

Концепція віртуалізації є досить простою. Установка віртуальної машини дозволяє запускати декілька операційних систем одночасно на одному фізичному сервері. Спеціальна програма, під назвою гіпервізор, ізолює віртуальні машини одна від одної і динамічно виділяє ресурси сервера кожній операційній системі. Віртуальні машини можна запускати використовуючи вже готовий образ, що значно полегшує створення інфраструктури. Також можна створити базовий образ віртуальної машини і на його основі підготувати потрібне середовище. Таким чином, віртуалізація є абстракцією апаратного рівня, тобто кожна віртуальна машина симулює реальний сервер. (рисунок 1.1.а).

Гіпервізор – частина програмного забезпечення, яке або працює безпосередньо на апаратному забезпеченні, або на серверній операційні системі. Він дозволяє декільком віртуальним машинам працювати на одному сервері. Ізоляція на основі гіпервізора може одночасно використовувати ресурси пам’яті, оскільки більшість віртуальних машин не використовують усю виділену пам’ять. Таким чином, користувачі можуть мати більше обчислювальних ресурсів, ніж фізично доступно в хмарних середовищах.

Розрізняють два типи гіпервізора:

- тип 1 – виконується безпосередньо на апаратному забезпеченні. У такому випадку гіпервізор виконує дві основні функції: виділяє ресурси процесорного часу та пам’яті на фізичному сервері та працює як керуючий шар, який дозволяє маніпулювати віртуальними машинами.

- тип 2 – працює всередині операційної системи у вигляді звичайного програмного забезпечення. Вітруалізація на основі програмного гіпервізора є найбільш поширеним типом віртуалізації.

Віртуалізація задовольняє потребу в розділенні ресурсів системи та повного відділення екземпляра платформи від основної системи. Дає можливість проводити точне налаштування системи (виділяти оперативну

пам’ять, кількість процесорних ядер, виділених на віртуальну машину і кількість дискового простору.

Перевагою такого методу є повна ізоляція усіх процесів які виконуються на віртуальній машині. Недоліком є обмежена кількість віртуальних машин які можна запустити на сервері, оскільки запуск повноцінної віртуальної машини вимагає використання досить великої кількості ресурсів. Таким чином запускати багато віртуальних машин на одному сервері надто ресурсозатратно, причому більша частина виділених ресурсів використовується для забезпечення роботи системи, а не для виконання поставленої задачі.

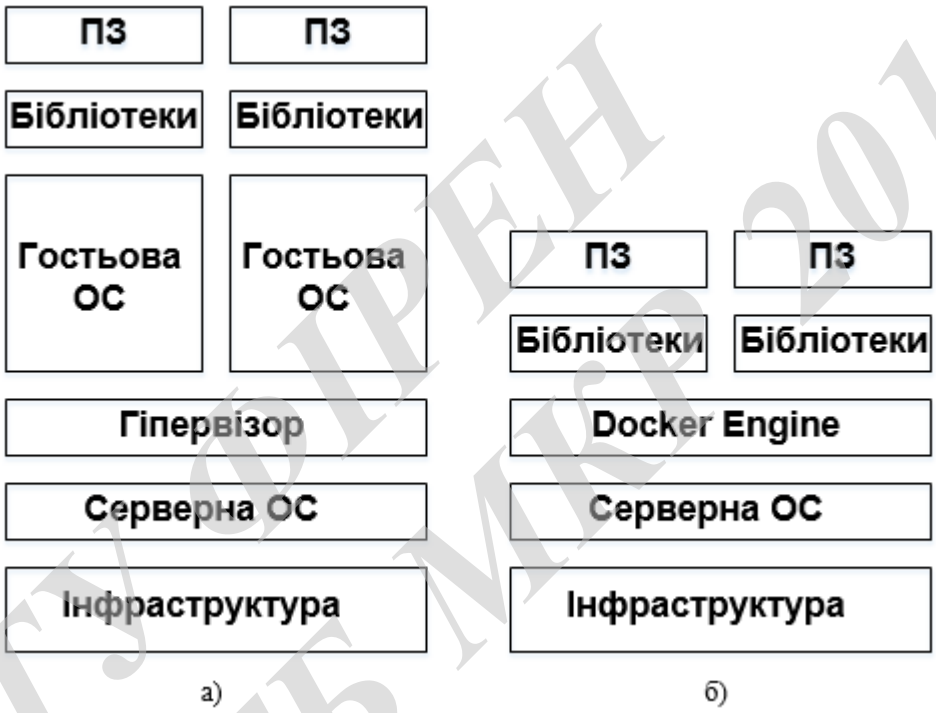


Рисунок 1 – Порівняння технологій контейнеризації та віртуалізації

Віртуалізація на рівні операційної системи або контейнеризація – дозволяє використовувати декілька віртуальних середовищ, не відходячи від основної операційної системи. Вона використовує функціональність ядра для ізоляції груп процесів одна від одної та системи в цілому. Ізольовані середовища називають контейнерами. Вони мають одне і те ж ядро операційної системи, при чому гіпервізор не вимагається.

Термін «контейнеризація» походить від морських контейнерів, оскільки він має таки же принцип: відправляти або зберігати усі види «вантажів». Контейнерна технологія не нова, контейнери були вперше представлені в 2004 році, однак найбільш широкого застосування набули в останні роки. Нові

досягнення зробили контейнеризацію більш популярною і простою у використанні.

Технологія контейнеризації є абстракцією програмного рівня, тобто кожний контейнер симулює окремо запущену програму. Хоча контейнери ізольовані один від одного, вони використовують спільну операційну систему.

Надаючи засоби для створення і використання контейнерів, операційна система дає можливість програмному забезпеченню працювати як на окремому сервері, при цьому спільно використовуючи велику кількість базових ресурсів. Економія від спільного використання ресурсів у поєднанні з ізоляцією призводить до того, що контейнери вимагають значно менших витрат ніж віртуальні машини. Це досягається за рахунок ізоляції на рівні операційної системи замість віртуалізації ядра апаратного забезпечення та операційної системи. Для створення ізольованого середовища між операційною системою та вищими рівнями абстракції використовуються функціональні можливості ядра Linux та Linux-подібних операційних систем:

- namespaces (pid, uts, net, mnt, ips, user) – абстракція над ресурсами в операційній системі, яка дозволяє ізолювати процеси. Linux namespaces розділяє ресурси таким чином, що кожен набір процесів бачить лише виділений йому набір ресурсів.

- cgroups – це функція ядра Linux, яка обмежує, вираховує та ізолює використання ресурсів (процесорний час, оперативна пам'ять, режим введення/виведення диска, мережа тощо) для набору процесів процесів.

Використання однієї операційної системи значно економить місце на жорсткому диску та в оперативній пам’яті, оскільки віртуалізація вимагає наявності окремих копій операційних систем для кожної віртуальної машини.

Оскільки для запуску контейнера не потрібно завантажувати цілу операційну систему, контейнери займають менше місця в пам’яті та можуть запускатись в лічені секунди, тим самим динамічно підлаштовуючись під навантаження системи.

Так як контейнери містять в собі усі необхідні для запуску програми компоненти (бібліотеки, додаткове ПЗ) та не прив’язані до конкретної операційної системи, їх можна легко переносити на інші системи без ризику виникнення проблеми з несумісністю програмного забезпечення. В той час як при перенесенні програми з однієї віртуальної машини на іншу вона може не працювати через конфлікт у версіях програмного забезпечення.

Основні відмінності віртуальних машин від контейнерів представлені в таблиці 1.1

Таблиця 1 – Відмінності між віртуальними машинами та контейнерами [2].

Віртуальна машина	Контейнер
Віртуалізація апаратного рівня	Віртуалізація програмного рівня
Займає багато місця на жорсткому диску (декілька Гігабайт)	Займає набагато менше місця на жорсткому диску (декілька Мегабайт)
Довгий запуск (декілька хвилин)	Швидкий запуск (декілька секунд)
Повністю ізольована	Ізоляція на рівні процесу
Володіє кращими показниками безпеки	Гірші показники безпеки в основному через меншу ізольованість
Вимагає більше ресурсів для роботи	Використовує значно менше ресурсів

Таким чином, контейнери мають багато переваг перед віртуальними машинами:

- легкість, отримана за рахунок відсутності гіпервізора;
- простота розгортання;
- швидкодія – оскільки програмне забезпечення запускається безпосередньо на ядрі, запуск та початкова конфігурація відбувається майже миттєво;
- ізоляція контейнерів один від одного та від сервера;
- зменшення затрат за рахунок оптимізації ресурсів;
- інкапсуляція та портативність – код програмного забезпечення із всіма залежностями може бути переміщений всередину контейнера і може бути запущений на будь якому сервері зі встановленим Linux та Docker;
- соціальний обмін – зберігання та спільне використання Docker образів.
- масштабованість.

Отже, використання контейнерів у порівнянні з віртуальними машинами є більш зручним та дозволяє використовувати менше ресурсів, що, відповідно, зменшує затрати на створення обчислювальної інфораструктури.

Контейнеризація є більш кращим рішенням порівняно з віртуалізацією у тих випадках, коли важливим показником є продуктивність. Більшість компаній використовують як контейнери так і віртуальні машини, наприклад, для запуску контейнерів у віртуальних машинах.

1.2 Docker

Серед сучасних реалізацій контейнерної інфраструктури найбільшого розповсюдження здобула платформа Docker. Основними її перевагами над конкурентними рішеннями (LXC, LDX та rtk) є зручна реалізація управління контейнерами та баланс між компактністю та доступним функціоналом.

Docker – це проект з відкритими початковим кодом, який дозволяє автоматизувати запуск програм всередині контейнерів, що є простим механізмом ізоляції для поділу одного сервера на декілька віртуальних середовищ.

Початково проект Docker був випущений компанією dotCloud під ліцензією Apache 2.0 як продукт з відкритим початковим кодом в березні 2013 року. Через 2 місяці було запущено публічний реєстр Docker. У другій половині того ж року Google, Yandex та Baidu інтегрували Docker у свої хмарні сервіси. Ідея Docker добре виражає цілі, які були поставлені при його створенні: «Створити «кнопку», яка дозволить створювати і розгортати будь яку програмну інфраструктуру на будь якому сервері в будь якому місці».

Docker почався з користувацької програми для запуску Linux-контейнерів. Починаючи з версії Docker 0.9 почали підтримуватись два драйвера: драйвер LXC, в якому використовувався раніше створений liblxc та власний драйвер, який використовував бібліотеку libcontainer. Даний драйвер обробляє управління контейнерами, використовуючи можливості ядра, такі як простір імен та cgroups.

Але Docker це більше ніж просто бібліотека віртуалізації, дана технологія абстрагує різницю між дистрибутивами операційних систем і створює стандартизоване середовище для розробки програм. Розробник програмного забезпечення може створити стандартну програму, яка є портативною та може працювати скрізь, де встановлений Docker Engine. Це значно економить час, оскільки більше немає необхідності підтримувати багато різних платформ і дистрибутивів операційної системи. Тай факт, що кожна програма працює у власному контейнерному середовищі, вирішує багато проблем, таких як повна деінсталяція або коли для двох програм вимагається дві різні версії одних і тих же залежностей.

Частина програмного забезпечення знаходиться в файловій системі в контейнерах Docker та містить всі залежності необхідні для запуску програмного забезпечення: код програми, середовище виконання, системні інструменти, системні бібліотеки, які встановлені на сервері. Це гарантує, що незалежно від середовища програмне забезпечення завжди буде проацювати однаково.

Особливості контейнерів Docker наступні:

- легкість: контейнери Docker мають невелику вагу та спільно використовують одне і те саме ядро операційної системи та ефективно використовують оперативну пам’ять, що дозволяє програмному забезпеченню запускатись практично миттєво. Завантаження образів набагато ефективніша завдяки мінімальному використанні диску, причому всі образи структуровані з багаторівневих файлових систем.

- відкритість: контейнери Docker побудовані на відкритих стандартах і можуть працювати поверх будь-яких операційних систем.

- безпека: додавання рівня захисту програмного забезпечення присутнє в кожному контейнері для ізоляції від інших контейнерів та хмарної інфраструктури.

- можливість запуску великої кількості програм на одному і тому ж обладнанні.

Docker забезпечує конкурентоспроможну, економічно ефективну альтернативу віртуальним машинам на основі гіпервізора. За останні роки екосистеми Docker набула значних змін. База користувачів продовжує збільшуватись, великі компанії публічно оголошують про те, що вони активно використовують Docker у якості своєї основної платформи.

1.3 Архітектура Docker

У Docker використовується клієнт-серверна архітектура. Основною її складовою є Docker Engine, який необхідно встановити на всі системи, які взаємодіють з Docker. Docker Engine включає в себе такі основні компоненти: Docker deamon, REST API, Docker client.

Користувач використовує Docker client для взаємодії з Docker deamon. Docker client передає інформацію на Docker daemon, який керує роботами з запуску, підтримки та розповсюдження контейнерів. Docker client та Docker daemon можуть бути запущені на одній і ті ж самій системі, також можна приєднати Docker client до віддаленого Docker daemon. Docker client та Docker daemon пов’язані через REST API використовуючи UNIX сокети або мережевий інтерфейс.

Архітектура платформи Docker схематично зображена на рисунку 2[4].

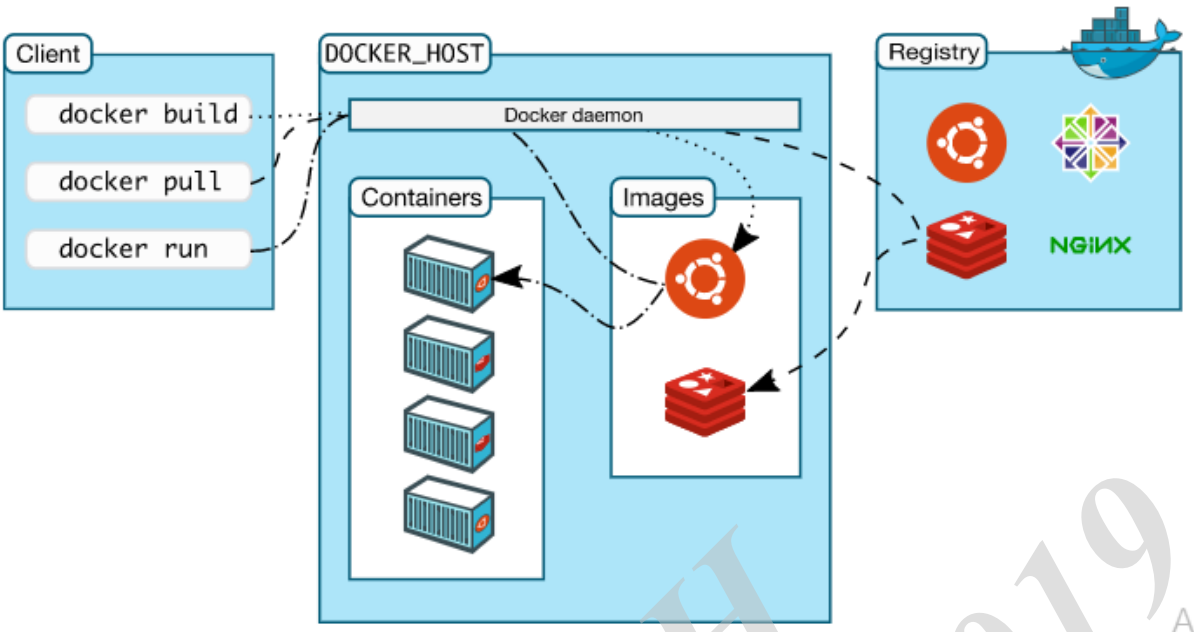


Рисунок 2 – Архітектура Docker

Docker client може бути встановлений на Windows, MacOS або сервери які працюють на Linux. Він використовується безпосередньо для взаємодії платформи з користувачем. При введенні команди (наприклад Docker run) Docker client передає її на Docker daemon, де введена команда виконується системою. Важливою перевагою даної платформи є можливість одного Docker client керувати декількаами Docker daemon.

REST API є проміжним рівнем для взаємодії із сервером Docker і для керування усіма його функціями. API можна відкрити або через локальний сокет на тому ж сервері, або через порт, відкритий через мережу.

Docker daemon це процес, який обробляє запити від Docker API та керує створенням образів, контейнерів, конфігурування мережі та доступом контейнерів до серверної операційної системи. Декілька Docker daemon можуть також комунікувати між собою для спільного керування сервісами.

Docker host це сервер на якому запущений Docker daemon. Docker client може бути запущений на іншому сервері та налаштований для керування віддаленим Docker daemon.

Docker registry це реєстр де зберігаються образи для створення контейнерів. Образи можуть бути створені користувачами або розробниками певного програмного забезпечення. Виділяють приватні та публічні реєстри.

Найбільш популярним публічним реєстром є Docker Hub. Docker Hub це хмарне сховище, в якому користувачі та партнери Docker створюють, тестують, зберігають та поширюють образи контейнерів.

Docker виконує свої функції завдяки поєднанню різних складових:

- складова збірки для створення контейнерів (Docker образ, або read-only шаблон);
- складова розповсюдження и зберігання образів (Docker реєстр);
- складова роботи для функціонування створеного програмного забезпечення (Docker контейнери)

1.4 Головні компоненти Docker

Для запуску програми в Docker необхідне розуміння 4 основних компонентів даної системи:

- Docker compose
- Dockerfile
- Образ (image)
- Контейнер

Docker образ (image) це read-only шаблон, який використовується для створення контейнера. Кожен Docker образ це файл, який складається з декількох рівнів, які використовуються для виконання коду в контейнері. При створенні образу, но кожен рівень можуть додаватись системні бібліотеки, утиліти та інші необхідні файли. Завдяки рівневій структурі, користувач може використовувати раніше створені рівні для побудови нових образів. Рівнева структура типового Docker контейнера представлена на рисунку 3.

Рисунок 3 – Рівнева структура Docker образу

Кожен з рівнів має криптографічний хеш, який його ідентифікує. Шари доступні тільки для зчитування, тобто всі зміни, які виконуються в

запущеному контейнері, будуть записувати в новий рівень контейнера поверх інших, з випадковим ідентифікатором. Коли контейнер буде видалено, цей шар також буде видалено, але шари під ним будуть існувати.

Рівнева структура це одна з причин малої ресурсозатратності. При зміні образу, наприклад, оновлення компонентів, створюється новий рівень. Тобто немає необхідності створювати образ заново, додається лише рівень.

Docker використовує рівневу структуру щоб спростити процес переміщення контейнерів в різних северних середовищах. AUFS (розширена багаторівнева файлова система уніфікації) є основним засобом, завдяки якому Docker забезпечує економію пам'яті і більш швидке розгортання контейнерів. Завдяки AUFS контейнери можуть спільно використовувати одні і ті самі частини файлової системи, які доступні тільки для читання. При запуску контейнера Docker використовує механізм під назвою union mount – файлові системи не монтуються в різних місцях, а поверх одна одної, тому вміст каталога може складатись із файлів каталогів із різних файлових систем.

Такий розподіл файлових систем и образів забезпечує дві основні переваги. По-перше, це забезпечує високу ступінь економії пам'яті. Якщо в двох контейнерах працює одна і та сама операційна система і вони маються спільні бібліотеки та залежності, більшість їх файлових систем будуть представлені на диску тільки один раз і не будуть дублюватись. По-друге, при завантаженні і рознгортанні контейнера, якщо хост уже має попередні рівні файлової системи, від яких залежить конкретний контейнер, йому потрібно завантажити тільки інкрементні зміни.

В основі більшості Docker образів лежить базовий образ, але користувач може також створити образ повністю з нуля. Кожен образ має один верхній рівень, в який програма може записувати дані.

При створенні нового контейнеру, верхній змінюваний рівень створюється автоматично. Цей рівень включає в себе всі змінювані об'єкти та називається контейнерним. Даний рівень містить усі новостворені файли, модифікації до існуючих файлів та недавно видалені файли. Зміни, які були внесені на контейнерному рівні, залишаються лише на даному рівні. Багато контейнерів можуть розділяти один спільний базовий образ, маючи свій власний контейнерний рівень.

Docker контейнер являє собою копію Docker образу з додатковим верхнім змінюваним рівнем. Контейнер містить всі необхідні компоненти для роботи програми. Контейнер можна створити, запустити, зупинити, перенести чи видалити. Кожний контейнер є ізольованою та безпечною платформою для програми.

Контейнери повністю ізолюють наступні елменти:

- файлову систему;
- IP адресу, мережеві інтерфейси;
- внутрішні процеси;
- простір імен;
- бібліотеки операційної системи;
- бінарні файли програм;
- залежності;
- файли конфігурацій програм.

Можливо запустити велику кількість контейнерів, заснованих на одному і тому ж образі. Кожен контейнер має унікальний ідентифікатор, який надається автоматично або призначається користувачем.

Контейнер володіє доступом тільки до свого екземпляра багаторівневої файлової системи. Якщо контейнеру необхідно відредагувати файл, який є частиною одного із нижніх рівнів тільки для зчитування, то цей файл буде скопійовано на рівень читання-запису верхнього рівня (рисунок 4). Контейнер може редагувати файл, але файл не зберігається в базовому образі. Це забезпечує ізоляцію для кожного контейнера.

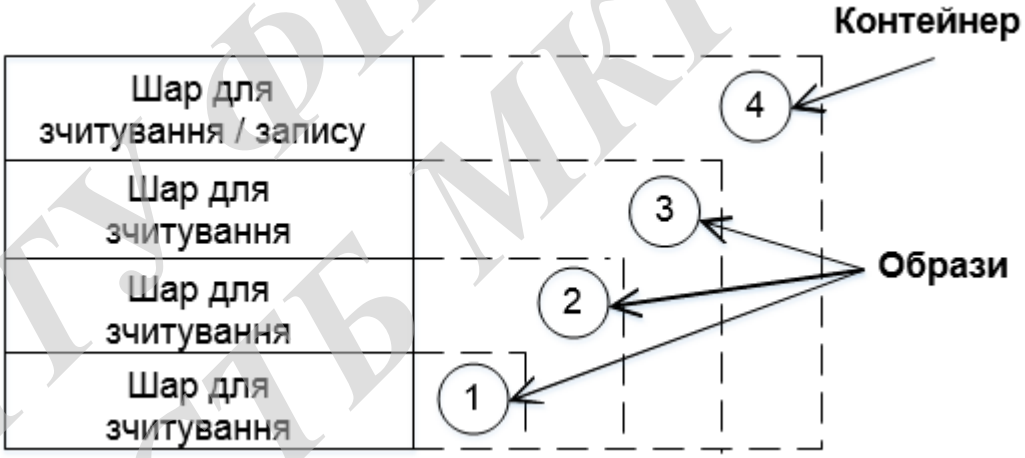


Рисунок 4 – Шари контейнера

Docker контейнери є основним компонентом в Docker мережі і містять усі компоненти, необхідні для виконання одного конкретного завдання.

Docker може створювати образи автоматично, читаючи інструкції з Dockerfile. Dockerfile автоматизує процес створення образів. Dockerfile це текстовий документ який містить у собі всі необхідні команди які вводить користувач для конфігурування образу. Наприклад, при необхідності створити образ з декількома встановленими додатками, замість того, щоб встановлювати і налаштовувати їх вручну для кожного контейнера, можна

прописати усі необхідні дії в Dockerfile. Піч час створення образу Docker виконає вказані в Dockerfile зміни.

Для того, щоб зберегти аналогію з мовами програмування, Dockerfile є програмним кодом який компілюється в запущений образ програмою Docker. Після того скомпільований образ може бути додатково сконфігурований і запущено через інтерфейс Docker, який використовує базовий Docker-daemon для взаємодії з операційною системою.

В Dockerfile дозволені такі ключові функції:

- додавання програмного забезпечення;
- створення змінного середовища;
- визначення TCP портів, які будуть відображатися поза контейнером;
- запуск команди всередині контейнера під час збірки;
- опціональна установка точки входу для контейнера (точка входу це певний процес запуску для контейнера);
- додавання файлу або директорії.

За певними виключеннями, майже кожна команда в Dockerfile створить новий шар в багаторівневій файловій системі образу. Docker Engine відповідає за розбір і інтерпретацію цих інструкцій та за подальшу побудову кінцевого Docker образу на їх основі [8].

Docker compose це утиліта, яка допомагає керувати всіма контейнерами, які складають одну мережу. Docker compose аналізує файл docker-compose.yml та створює з нього багатоконтейнерне середовище. Сам файл docker-compose.yml являє собою конфігурацію програми, яка складається з одного або декількох сервісів. Docker compose переводить вказані дані мереж і томів в команди типу docker run, docker volume create чи docker network create та виконує їх. В даних файлах описуються основні параметри контейнерів, які запускаються. Таким чином, значно спрощується процес запуск складних програм, які вимагають декілька контейнерів, визначаються потрібні розділи серверної файлової системи, що доступні контейнерам, та порти, які доступні контейнерам. За допомогою даного інструменту можна задавати структуру складної мікросервісної програми, обчислювальні ресурси та залежності.

Наприклад, при необхідності створення додатку з 2 контейнерів потрібно зібрати образ для кожного з них, виконати правильне налаштування мережі, конфігурувати для кожного з контейнерів доступ до хостової ОС при необхідності. У випадку падіння одного з контейнерів, його доведеться запускати знову, повторюючи усі операції. У складних мережах декількома десятками контейнерів це завдання стає досить нетривіальним. За допомогою Docker compose можна створити файл, який, крім образів, містить у собі усі необхідні залежності. У випадку, коли один з контейнерів завершить свою

роботу через певну помилку, буде створено новий контейнер з такою ж конфігурацією.

1.5 Принцип роботи Docker

Як було згадано раніше, образ – це read-only шаблон який використовується для створення контейнера. Усі образи складаються з певного набору рівнів. Спеціальна файлова система використовується щоб поєднати всі ці рівні в один, дозволяючи файлам і директоріям з різних систем когерентно накладатись, створюючи єдину файлову систему.

Використання рівнів це одна із головних причин малої ваги образу. При оновленні програмного забезпечення всередині контейнеру створюється новий рівень, всі інші рівні образу залишаються незмінними, в той час як при використанні віртуальних машин часто доводиться створювати усю систему заново.

В основі кожного образу знаходиться базовий образ. Базові образи можна використовувати для створення нових образів. Базові образи зазвичай завантажуються з реєстру.

Реєстр – місце зберігання Docker образів. Образ можна опублікувати у публічному реєстрі Docker Hub. Docker клієнт може завантажувати вже опубліковані образи на сервер та створювати з них контейнери.

Docker Hub – це сервіс, який надає інструменти для управління і адміністрування образів Docker. Репозиторій Docker Hub містить у собі велику кількість програмного забезпечення і бібліотек, готових до запуску в будь-якому середовищі Docker.

Docker Hub обробляє аутентифікацію і авторизацію користувачів і включає три види репозиторіїв:

- офіційні репозиторії, які містять у собі образи від вендорів Docker;
- приватні репозиторії, де можуть зберігатись непублічні образи;
- публічні репозиторії для обміну публічними образами.

Приватні репозиторії виключаються з результатів пошуку і доступні тільки певним людям, які володіють правами на завантаження даних образів.

Публічний репозиторій доступний для пошуку і образ з нього може бути завантажений ким завгодно.

Docker Hub містить у собі тисячі образів, створених товариством, з багатьма формами і властивостями, які можуть стати основою для інших розробок.

Для створення нових образів користуються інструкціями. Інтсрукціями вважають наступні дії:

- запуск команди
- додавання файлу або директорії
- створення змінного оточення
- вказання команд для запуску всередині контейнера.

Усі інструкції зберігаються у Dockerfile. Docker зчитує дані інструкції, виконує та віддає готовий образ.

1.6 Принцип роботи контейнера

Контейнер містить файли та метадані користувача. Кожен контейнер створюється з образу. Образ доступний лише для зчитування. При запуску контейнера створюється верхній рівень, який використовується для запису даних. Контейнер запускається за допомогою Docker deamon, який отримує команди від Docker клієнта або програмного інтерфейсу.

Для запуску контейнера необхідні такі атрибути:

- образ для запуску контейнера;
- команда яка буде запущена всередині контейнера.

Після введення команди відбуваються наступні дії:

- завантаження базового образу. Наявність образу перевіряється спочатку на сервері, якщо образ відсутній, він завантажується з реєстру, вказаного за замовчування;
- ініціалізація файлової системи. Створюється контейнер і монтується read-only образ;
- створення мережі. Ініціалізується мережевий інтерфейс, який забезпечує комунікацію контейнера з серверо;
- запускається процес (контейнер) та усі програми у ньому.

1.7 Технології, які використовуються в Docker

Docker використовує ряд можливостей ядра Linux для реалізації ізольованого запуску контейнерів.

Ключовою технологією для ізольованості контейнерів є Linux namespaces. Усі Linux процеси формують єдину деревоподібну ієрархію. Зазвичай привілейовані процеси у такому дереві можуть створювати або вбивати інші процеси. Linux namespaces дають можливіть створювати окремі процеси з власними «піддеревами». Процеси в таких «піддеревах» не мають доступу до інших процесів.

Linux namespaces виділяє ресурси таким чином, що кожен підпроцес бачить їх як свої. Наприклад, при розгляді PID namespace, видно що усі процеси ієрархічно виходять від процесу PID 1 (init). Процес, який створює новий простір імен, все ще залишається у батьківському просторі імен, але робить його дочірнім коренем нового дерева процесів.

Але це лише означає, що процеси в новому просторі імен не можуть бачити батьківський процес, але простір імен батьківського процесу може бачити дочірній простір імен. Процеси в новому просторі імен тепер мають 2 PID: один для нового простору імен та один для глобального простору імен.

Список деяких просторів імен, які використовує docker:

- pid: для ізоляції процесу;
- net: для управління мережевими інтерфейсами;
- ipc: для управління ICP ресурсами. (ICP: InterProccess Communication);
- mnt: для управління точками монтування;
- utc: для ізолювання ядра і контролю генерації версій (UTC: Unix timesharing system).

Використовуючи namespaces контейнери можуть монтувати частину файлової системи сервера таким чином, щоб вона була доступна всередині контейнера.

Контрольні групи (cgroups) – це функція ядра Linux, яка обмежує, ізолює та вимірює використання ресурсів групи процесів. Можна встановити квоти ресурсів на пам'ять, процесор, мережу та запис/зачитування. Контрольні групи також забезпечують додаткову ізоляцію простору імен для того, щоб повністю ізолювати контейнер або програму від операційного середовища, включаючи дерева процесів, мережу, ідентифікатори користувачів та змонтовані файли операційної системи.

Хоча Linux досить добре обробляє та розподіляє доступні ресурси між процесами, іноді потрібно краще контролювати ресурси. Наприклад, необхідно виділити або гарантувати певну кількість ресурсів групі процесів. Дану функцію реалізують контрольні групи.

Однією з цілей створення контрольних груп було забезпечення єдиного інтерфейсу для багатьох завдань, починаючи від контролю одного процесу закінчуючи повною віртуалізацією системного рівня.

Контрольні групи гарантують що кожен контейнер має достатню кількість ресурсів для правильної роботи та запобігають вичерпанню цих ресурсів, обмежуючи використанням ресурсів контейнером для того, щоб одни контейнер не мав можливості вичерпати систему, використавши всі її ресурси.

Контрольні групи забезпечують:

- Обмеження ресурсів. Групи можуть бути встановлені таким чином, щоб не перевищувати встановлений ліміт пам’яті, куди також входить і ліміт файлової системи.

- Пріорітизація. Деяким групам можна дозволити більше використання системних ресурсів.

- Облік. Керування використанням ресурсів групою. Дана функція можу бути використана для виділення групі визначеної кількості ресурсів.

- Контроль заморожених груп чи процесів, визначення чи є вони правильно функціонуючими та їх перезавантаження.

Контрольною групою є сукупність процесів, які пов’язані одними критеріями і прив’язані до набору параметрів. Групи можуть бути ієрархічними, тобто кожна група отримує ресурси з «батьківської» групи. Ядро забезпечує доступ до багатьох підсистем за допомогою інтерфейсу контрольних груп [4].

Контрольні групи можна використовувати декількома способами:

- Шляхом доступу до віртуальної системи контрольних груп вручну.
- Створюючи і керуючи групами автоматично використовуючи утиліти cgcreate, cgexec, та cgclassify.
- Використовуючи «rules engine daemon», який автоматично переносить процеси певного користувача, груп користувачів або команд до контрольних груп залежно від конфігурації.
- Непрямо, завдяки програмному забезпеченню, наприклад Docker.

По мірі розвитку Linux виникла необхідність ізолювати процеси від файлової системи хоста, і для цієї цілі була створена команда chroot. Більшість програм для контейнеризації на сьогоднішній день розроблено на основі chroot. Chroot – скорочення від change root – стара функція в Linux, яка замінює видимий каталог для процеса. Chroot використовується як для виклику утиліти, так і для системного виклику і дозволяє вказати нову кореневу директорію. Процес і його потомки таким чином не можуть отримати доступ до файлів вище нового кореневого каталогу.

1.8 Типи Docker мереж

Одна з причин великої популярності Docker контейнерів це можливість з’єднати їх разом або навіть з’єднати контейнер мережею з серверами, які не використовують Docker. Сервіси, які знаходяться всередині контейнера, можуть не знати, що вони знаходяться всередині Docker мережі. Також

зовнішні сервіси можуть обмінюватись інформацією з контейнером так само як і з звичайним серверним додатком.

Конфігурація Docker мереж здійснюється за допомогою спеціальних драйверів. Деякі з них існують як конфігурація за замовчуванням і надають основні мережеві функції:

- bridge: мережевий драйвер за замовчуванням. Якщо при налаштуванні не вказаний інший мережевий драйвер, це тип мережі, яка використовується. Мережі bridge зазвичай використовуються, коли програми працюють в автономних контейнерах, які потребують зв'язку.

- host: використовується для автономних контейнерів. Даний тип видаляє мережеву ізоляцію між контейнером та сервером Docker та використовує мережу сервера безпосередньо.

- overlay: створює внутрішню приватну мережу, яка охоплює всі вузли, що входять у кластері. Таким чином, мережі Overlay полегшують зв’язок між службою та окремим контейнером або між двома окремими контейнерами на різних Docker Daemons.

- macvlan: Мережі Macvlan дозволяють призначити MAC-адресу контейнеру, завдяки чому він відображається як фізичний пристрій у мережі. Демон Docker спрямовує трафік до контейнерів за їх MAC-адресами. Використання драйвера macvlan іноді є найкращим вибором для роботи з застарілими програмами, які очікують, що вони будуть безпосередньо підключені до фізичної мережі, а не спрямовані через мережевий стек хоста Docker.

- мережеві плагіни: існує можливість встановлювати та використовувати сторонні мережеві плагіни за допомогою Docker. Ці плагіни доступні у Docker Hub або від сторонніх постачальників.

2 ОЦІНКА ЗАХИЩЕНОСТІ DOCKER МЕРЕЖ

До появи технології Docker та технології контейнеризації більшість організацій використовували віртуальні машини або розміщували програми безпосередньо на сервері. З точки зору безпеки, ці технології відносно прості. При проектуванні обчислювальної мережі у таких випадках необхідно зосередитись лише на двох шарах (сервер та безпосередньо програмний продукт) під час моніторингу та контролю за подіями, що стосуються безпеки. Також при використанні таких підходів зазвичай не потрібно приділяти увагу API, мережевих конфігурацій або складних конфігурацій, визначених програмним забезпеченням, оскільки це, як правило, не є основною частиною розгортання віртуальної машини або сервера.

З метою покращення безпеки в Docker з самого початку вбудовано кілька важливих засобів забезпечення безпеки [5]:

- Docker контейнери мінімальні: один або кілька працюючих процесів, тільки необхідне програмне забезпечення. Це знижує ймовірність атак пов’язаних з вразливостями в ПО.

- Docker контейнери виконують специфічне завдання. Заздалегідь відомо, що повинно виконуватися в контейнері, визначено шляхи до розділів файлової системи, відкриті порти, конфігурації процесів, точки монтування і т.д. У таких умовах простіше виявити будь-які пов'язані з безпекою аномалії. Цей принцип організації систем пов’язаний з мікросервісной архітектурою, та дозволяє значно зменшити поверхню атаки.

- Docker контейнери ізольовані як від сервера, так і від інших контейнерів. Такий рівень ізоляції ддосягається завдяки здатності ядра Linux ізолювати ресурси за допомогою cgroups і namespaces. Але у той же час ядро доводиться ділити між хостом і контейнерами, що веде до потенційної можливості виходу з контейнера.

- Docker контейнери відтворювані. Завдяки їх декларативні системі збірки досить легко можна з'ясувати, з чого і як був зроблений контейнер.

В той же час, забезпечення інформаційної безпеки контейнерів Docker є набагато складнішим завданням, в більшості випадків тому, що в типовому середовищі Docker є набагато більше компонентів, які потенційно можуть піддаватись атакам. Ці компоненти включають:

- Безпосередньо контейнери. Типов інфорастуктура з використанням Docker контейнерів зазвича містить декілька Docker образів, кожен з яких розміщує окремі мікросервіси. Кожен із таких образів та контейнери які з них створюються необхідно захищати та контролювати окремо.

- Docker Daemon, конфігурації безпеки якого також повинні бути чітко визначені для того щоб забезпечити безпечну роботу контейнерів, які ним керуються.

- Інфраструктура нижчого рівня, на якій розгортається Docker мережа, причому вона можу бути побудована як і з використанням серверів, так і на віртуальних машинах, що веде до використання різних підходів до безпеки на даному рівні.

- Якщо контейнерна інфраструктура розгортається в хмарних сервісах, то це призводить до створення додаткових потенціальних вразливостей, які, до того ж, залежать від конфігурації конкретної платформи.

- Накладені мережі та API, що полегшують зв'язок між контейнерами можуть також стати потенційним слабким місцемта призвести зміни зловмисником початкових Docker образів.

- Прив'язка томів ОС або інших системи зберігання даних, які існують зовні від контейнерів.

Таким чином, питання безпеки Docker мереж є набагато складнішим завданням через наявність більшої кількості компонентів та більшого числа рівнів абстракції, що вимагає використання різного роду методів та підходів при створенні Docker мережі.

2.1 Основні види вразливостей

Хоча Docker є відносно безпечною платформою, при розробці якої приділяється багато уваги питанням безпеки, існують деякі специфічні частини архітектури на основі Docker, які більш схильні до атак.

- Безпека сервера та ядра системи

Одне з найпростіших і в той же час ключових питань захисту Docker мереж - це безпека сервера, на якому розгортається Docker мережа, зокрема безпека ядра ОС. В уже скомпрометованій системі ізоляція та інші механізми безпеки контейнерів, які можна було б використовувати, втрачають свій сенс. Основна проблема закладена в архітектурі Docker, оскільки Docker спроектований таким чином, що всі запущені контейнери використовують ядро серверної операційної системи.

Класичний приклад атаки спрямованої на вразливість серверної операційної системи - це втеча з Docker контейнера. В офіційній документації така поведінка називається «виходом за межі контейнера» (container breakout)[6] і описує ситуацію, при якій програмі, запущеній всередині контейнера, вдається подолати механізми ізоляції і отримати root доступ або

доступ до важливої інформації, що зберігається на сервері. Типовий приклад даної вразливості - DirtyCow (CVE-2016-5195)[7].

Для реалізації захисту від описаних загроз найбільш дієвим підходом є зменшення кількості привілеїв для контейнера, що видаються йому за умовчанням. Так, якщо Docker контейнер запускається як root користувач, необхідно створити для такого користувача простір імен (user-level namespace) з мінімальними привілеями.

- Зловживання ресурсами контейнерів

Контейнери в середньому набагато численніші, ніж віртуальні машини, їхня мала ресурсозатратність дозволяє розмістити великі кластери навіть на скромному обладнання. Це, безумовно, є перевага, але в той же час це означає, що велика кількість програмних засобів конкурує за ресурси серверної операційної системи. Недоліки програмного забезпечення, помилки при розробці або навмисна атака можуть легко викликати відмову в обслуговуванні, якщо обмеження ресурсів невірно сконфігуровано.

Наприклад, один контейнер або цілий пул контейнерів може зайняти собі весь процесорний час серверної операційної системи і різко зменшити її продуктивність. Аналогічна ситуація може скластися і з мережевими інтерфейсами, коли кількість пакетів, що генеруються перевищує нормальну пропускну здатність мережі.

- Уразливості безпеки, присутні в Docker образі

Docker розповсюджується як open source проект, і Docker образи для створення власних контейнерів, також знаходяться в публічно безкоштовному доступі. Це означає, що зловмисники за допомогою фішингу та інших методів соціальної інженерії можуть маніпулювати діями користувачів, які використовують сховища Docker Image, щоб змусити їх завантажити заздалегідь скомпрометований образ з бекдором або зловмисним програмним забезпеченням.

Наприклад, в квітні 2019 року багато шуму наробила новина про те, як невідомі зловмисники отримали доступ до бази даних найбільшої в світі бібліотеки образів для контейнерів Docker Hub. В результаті були скомпрометовані імена користувачів, хеші паролів, а також маркери для репозиторіїв GitHub і Bitbucket, що використовуються для автоматизованих збірок Docker.

- Дані користувачів та Docker secrets.

Дуже часто програмному забезпеченню потрібна конфіденційна інформація для запуску: хеш-коди для паролів користувачів, сертифікати на стороні сервери, ключі шифрування тощо. Дана ситуація погіршується природою контейнерів: при побудові архітектури відбувається на просте

налаштовуєте сервера, Docker контейнери є динамічними та постійно створюються і руйнуються. Тому виникає необхідність у створенні автоматичного і безпечного процесу, для того щоб обмінюватися цією конфіденційною інформацією.

Оскільки основні загрози безпеці Docker мереж є відомими, можна розробити ряд стандартних рекомендацій для боротьби з такими загрозами.

Для покращення безпеки серверів, на яких розгортаються Docker мережі, перш за все необхідно переконатися в тому, що конфігурації Docker engine та серверної операційної системи захищені (обмежений і автентифікований доступ, зашифрована передача даних тощо).

Серверна операційна системи повинна отримувати регулярні оновлення, необхідно також слідкувати за інформаційними оновленнями безпеки від розробників операційної системи та усього програмного забезпечення, яке на ній встановлено, особливо якщо воно надходить із сторонніх сховищ.

Використання мінімальних, орієнтованих на контейнер серверних операційних систем, таких як CoreOS, Red Hat Atomic, RancherOS тощо, зменшить поверхню атаки і може принести деякі корисні нові функції, такі як запуск системних служб у контейнерах.

Також необхідно застосовувати обов'язковий контроль доступу, щоб запобігти небажаним операціям - на сервері та у контейнерах - на рівні ядра, використовуючи такі інструменти, як Seccomp, AppArmor або SELinux.

Найкращим методом боротьби з виходом за межі контейнера є обмеження доступу і можливих варінтів використання команд всередині контейнера. Команди, які дозволено виконувати всередині Docker контейнера, повинні обмежуватись мінімальним набором, який залежить від конкретногоь програмного забезпечення, що встановлене в контейнері.

CAP_SYS_ADMIN - це особливо небезпечний рівень безпеки, він надає широкий спектр дозволів кореневого рівня: монтаж файлових систем, введення просторів імен ядра, операції ioctl тощо.

Найкращим вирішенням проблема виходу з контейнера є створення ізольованого простору імен користувачівдля того, щоб обмежити максимальні привілеї контейнерів над хостом еквівалентом звичайного користувача. Якщо це можливо, необхідно уникати запуску контейнерів як uid 0. Якщо виникає необхідність запускати контейнер з великою кількістю наданих йому прав, необхідно завжди перевіряти чи надходить цей контейнер з надійного джерела.

Необхідно завжди слідкувати за небезпечними точками монтування серверної файлової системи у контейнер: Docker socket (/var/run/docker.sock), / proc, / dev тощо. Зазвичай ці спеціальні розділи файлової системи необхідні для виконання основних функцій контейнера, необхідно розуміти навіщо і як

обмежити процеси, які можуть отримати доступ до цієї привілейованої інформації. Іноді просто монтування файлової системи з правами лише для читання повинно бути достатньо, не рекомендується надавати доступ до запису, не оцінивши усі можливі ризики. У будь-якому випадку, Docker виконує копіювання при записі для того, щоб зміни, зроблені в одному контейнері, не вплинули на образ, що використовується для інших контейнерів.

Для забезпечення безпеки мережевої взаємодії можна налаштувати правила iptables, реалізовані в Docker. Наприклад, вказати діапазон IP-адрес для джерела пакетів, щоб обмежити трафік на контейнер. Це допоможе запобігти звернення глибоко ізольованого контейнера в зовнішню мережу.

Основним методом боротьби із скомпроментованими Docker образами є перевірка джерела образу та сканування його на наявність зловмисного програмного забезпечення. Щоб уникнути проблем із скомпроментованими образами, необхідно використовувати приватні (private) або строго довірені репозиторії (trusted repositories) на зразок Docker Hub. На відміну від інших репозиторіїв, образи, які там зберігаються, завжди сканує і переглядає спеціальний робот безпеки Docker's Security Scanning Service.

Ще один корисний і доступний широкому колу інструмент, яким варто скористатися, - Docker Content Trust. Це нова функція, доступна з версії Docker Engine 1.8. Вона дозволяє верифікувати власника образу. Таким чином цей сервіс захищає Docker мережу від скомпроментованих контейнерів і підробок, атак повторного відтворення і компрометації ключів.

Необхідно періодично оновлювати та перезбирати Docker образи, щоб отримати найновіші оновлення безпеки. Для таких цілей важливо проводити тестування збірки перед запуском контейнера в основному середовищі, щоб переконатися, що ці оновлення не порушують роботу системи.

Оновлення вже існуючих контейнерів, як правило, вважається поганою практикою. Найкращим підходом, який зарекомендував себе з точки зору безпеки та швидкодії є перезбірка всього контейнера з кожним оновленням. Docker має декларативні, ефективні, легкі для розуміння системи побудови, тому така практика є досить малозатратною в часі та ресурсах.

Необхідно використовувати програмне забезпечення від дистриб'ютора, який гарантує оновлення безпеки, оскільки все, що буде встановлене вручну безпосередньо з дистрибутива серверної операційної системи, доведеться оновлювати самостійно, що може займати деякий час.

Не потрібно створювати важкі контейнери з надишковим програмним забезпеченням. Мінімальні системи рідше потребують оновлення. Менша кількість програмного забезпечення також приводить до меншої кількості атак

та проблем з оновленнями. Якщо контейнери містять в собі багато складних конфігурацій, їх варто розділити на декілька сервісів.

Ні в якому разі не можна використовувати змінні середовища для збереження секретів, це дуже поширена, але дуже небезпечна практика.

Обмеження ресурсів за замовчуванням вимкнено на більшості систем контейнеризації. Однак, їх ручне конфігурування є необхідністю при створенні мереж з великою кількістю контейнерів.

Щоб відкалібрувати ресурси, що виділяються для контейнера (CPU, RAM, SWAP, I / O), потрібно задати порогові значення наступним параметрам:
- встановити ліміт виділення оперативної пам'яті (hard);
- встановити м'який ліміт пам'яті (soft);
- встановити ліміт пам'яті ядра (kernel OS);
- обмежити кількість використовуваних CPU;
- обмежити пропускну здатність для конкретного пристрою.

Також необхідно пам’ятати про можливість використання технології cgroups. Контрольні групи (cgroups) - це штатний інструмент що надається ядром Linux, який дозволяє обмежувати доступ процесів і контейнерів до системних ресурсів.

Всі описані раніше статичні контрзаходи не охоплюють усіх векторів нападу. Інколи може виникнути ситуація коли власна внутрішня програма має вразливість, або зловмисники використовують атаку нульового дня, яку неможливо виявити скануванням.

Моніторинг, виявлення аномальної, підозрілої поведінки контейнера і сповіщення про це - один з ключових інструментів аналізу та своєчасного реагування на інциденти.

Моніторинг не є заміну для будь-яких інших статичних практик безпеки: попередження атак завжди краще для виявлення атак. Його необхідно використовувати лише як додатковий шар спокою.

Конфігурування об’ємних журналів подій від усіх служб та серверів, які правильно зберігаються, легко фільтруються та співвідносяться з будь-якими змінами, можуть суттєво допомогти при аналізі атак та створенні заходів з подальшого їх запобігання.

2.2 Огляд доступних рішень для покращення безпеки Docker мереж

Існує ряд програмних засобів які дозволяють проаналізувати вразливості Docker мереж та виправити деякі недоліки в їх конфігурації.

Базовий рівень безпеки всієї інфраструктури Docker – захист сервера, на якому встановлений Docker і на якому запускаються контейнери. Перевірка

безпеки хоста як такого - досить обширна практика і виходить за рамки лише безпеки Docker, проте в загальному випадку можна використовувати дану утиліту для того, щоб зрозуміти, на якому рівні захисту сервер знаходиться в даний момент.

Lynis – утиліта для базової швидкої перевірки основних параметрів безпеки Unix-based систем, таких як конфігурація користувачів ОС, робота підсистем аудиту, невикористовувані акаунти і файли і навіть пара перевірок для Docker, таких як, наприклад, наявність невикористовуваних контейнерів і т.д.

Найбільшим розділом з точки зору кількості утиліт, які аналізують Docker образи на предмет відомих вразливостей в системних пакетах і залежностях є перевірка Docker образу. Дані програмні пакети аналізують проблеми саме конкретних збірок, які використовуються в образі і те, який набір програмного забезпечення і з якими проблемами в ці збірки потрапляє.

В основному дані інструменти аналізують наявні в образі пакети, отримують їх версії, а потім для конкретної версії пакету, використовуючи загальнодоступні бази вразливостей, складають перелік проблем для конкретного об'єкту сканування образу.

Dockle – даний інструмент не стільки шукає уразливості в системному програмному забезпеченні образу, скільки перевіряє коректність і безпеку конкретного образу як такого, аналізуючи його шари і конфігурацію – які користувачі створені, які інструкції використовуються, які томи підключені, присутність порожнього пароля і т.д. На даний момент кількість перевірок не дуже велике і базується на рекомендаціях CIS Benchmark для образів і декількох власних перевірках.

Clair – являє собою API і бекенд для перевірки шарів заданого образу на ті чи інші вразливості, список яких він бере із загальнодоступних баз.

За замовчуванням для використання Clair додатково встановлюється сервер та Postgres база даних, куди тривалий час заносяться дані про наявні загрози в різних збірках. Потім ці дані використовуються для аналізу образу.

Trivy – знаходить уразливості двох типів - проблеми збірок операційних систем (підтримуються Alpine, RedHat (EL), CentOS, Debian GNU, Ubuntu) і проблеми в залежностях (Gemfile.lock, Pipfile.lock, composer.lock, package-lock.json, yarn. lock, Cargo.lock). На відміну від Clair вміє сканувати як в репозиторії, так і локально, так і на підставі переданого .tar файлу з Docker образом. Програма вміє опціонально показувати тільки ті загрози, для яких можливе виправлення, а також приховувати загрози, додані в локальний білий список. Вивід здійснюється як на екран, так і в json файл. При цьому є можливість фільтрації вразливостей згідно їх критичності.

3 МОДЕЛЮВАННЯ ЕФЕКТИВНОСТІ ФУНКЦІОНУВАННЯ МЕРЕЖІ

Для визначення захищеності Docker мережі необхідно розробити математичну модель стану її елементів, на підставі якої можна здійснити прогнозування її станів. Таким чином, необхідно задатися набором станів, в яких можуть знаходитися елементи досліджуваної системи, і визначити причини, що викликають переходи між ними. Також необхідно визначити ступінь впливу стану елементів, що становлять мережу, на стан мережі в цілому. Також необхідно визначити метод розрахунку, за допомогою якого здійснюється розрахунок характеристик Docker мережі в цілому через характеристики її елементів.

Для вирішення поставленого завдання слід здійснити вибір показника, з яким буде ототожнюватися стан досліджуваної системи. Виходячи з поставленого завдання і особливостей розглянутих загроз ІБ, вибір показника пропонується здійснити серед нормованих показників надійності, пов’язаних з доступністю інформації в мережі. Вихідні дані про стан елементів мережі для побудови математичної моделі і здійснення подальшого прогнозування потоку відмов і відновлень елемента слід почати на підставі спостережень досліджуваної мережі.

Також даної частини дослідження необхідно проаналізувати залежність стану системи від стану її елементів беручи до уваги відмінності топологічні схеми, розміри і склад її елементів.

3.1 Моделювання станів мережі з використання методів теорії надійності

Досліджувана системи являє собою сукупність вершин і ребер зі складними зв’язками. Складна структура мережі може бути представлена у вигляді математичного графа. Вершинами графа, є вузли зв'язку, ребрами – лінії зв'язку. Завданням для побудови математичної моделі системи є забезпечення зв'язності між контейнером, де розміщені інформаційні ресурси, і контейнером, що є споживачем інформації, розміщеної на даних ресурсах, що є завданням теорії графів. Дане завдання має аналітичне рішення. Таким чином, кожен елемент системи являє собою або вузол зв'язку, або лінію зв'язку (ребро).

Шлях графа являє собою послідовність пристроїв, які повинен подолати мережевий трафік від передавача до приймача. Кожен з контейнерів мережі можна представити як пристрій з певними характеристиками, як технічними, так і пов'язаними з надійністю. Для математичного моделювання і прогнозування характеристик надійності досліджуваної системи, її має бути

представлено у вигляді сукупності окремих серверів, з'єднаних між собою. Оскільки Кг є ймовірнісної величиною, то для розрахунку Кг мережі слід використовувати математичний апарат теорії ймовірностей.

Одним з методів розрахунку Кг складної системи є метод її послідовного розкладання на сукупність мереж з лінійної топологією. Кг послідовно з'єднаних пристроїв відповідно до теореми про ймовірності незалежних подій розраховується за такою формулою:

$$К_{г_{посл}} = К_{г_1} \times К_{г_2} \times \ldots \times К_{г_n} \quad (3.1)$$

де $К_{г_1} \ldots К_{г_n}$ – Кг n послідовно з’єднаних елементів.

Кг паралельно з'єднаних елементів розраховується за формулою:

$$К_{г_{пар}} = 1 - (1 - К_{г_1)} \times \left(1 - К_{г_2}\right) \times \ldots \times \left(1 - К_{г_n}\right) \quad (3.2)$$

де $К_{г_1} \ldots К_{г_n}$ – Кг n паралельно з’єднаних елементів.

Таким чином, використовуючи вказані формули, є можливим розрахунок характеристики надійності складної системи, представивши її у вигляді шляху між вершинами графа, що описує розглянуту систему, яка є передавачем і приймачем трафіку.

3.2 Математична модель стану елементу системи

На підставі спостережень за станом серверів, можливо визначити число їх відмов і час знаходження в працездатному і непрацездатному стані. На підставі цих даних можливо створити математичну модель характеристик надійності і використовуватися для прогнозування стану контейнера в довільний момент часу.

Кожен елемент системи являє собою сукупність певних контейнерів. На рисунку представлений граф станів обладнання, де стану 1 відповідає справний стан, а стану 2 відповідає несправний стан, λ - інтенсивність потоку відмов, μ - інтенсивність потоку відновлень.

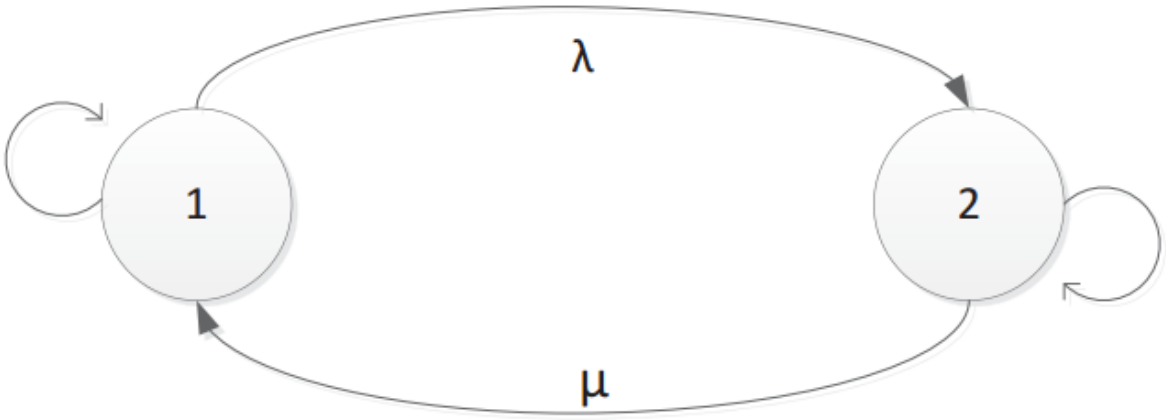


Рисунок 5 – Граф станів одного елементу системи

Коли стан елемента стабілізовано (воно знаходиться або у справному, або у несправному стані), його поведінку можна уподібнити поведінці невідновлюваного об'єкта. При виключенні з розгляду часу відновлення пристрою (оскільки в цей час відмова статися не може) відмови формують потік. Інтенсивність потоку відмов буде розраховуватися за формулою:

$$\lambda(t) = \frac{n(t)}{N_{cp} + \Delta t} \tag{3.3}$$

де $n(t)$ – число елементів, які знаходяться в непрацездатному стані на інтервалі часу Δt,

Δt – інтервал часу,

N_{cp} – середнє число пристроїв, які знаходяться в працездатному стані на інтервалі часу Δt.

Також інтенсивність потоку відмов можна виразити за допомогою ймовірності безвідмовної роботи і частоти відмов:

$$\lambda(t) = \frac{N_0 \times a(t)}{N_{cp}(t)} \tag{3.4}$$

Де N_0 – число контейнерів, які знаходяться в працездатному стані на момент початку спостереження.

$a(t)$ – середньозважена імовірність перебування контейнера в непрацездатному стані.

Середньозважена імовірність знаходження пристрою в непрацездатному стані визначається за формулою:

$$a(\Delta t) = \frac{n(\Delta t) \times t_{\text{в}}}{N_0 \times \Delta t} \tag{3.5}$$

де $n(\Delta t)$ – число контейнерів, які знаходяться в непрацездатному стані на інтервалі часу Δt,

$t_{\text{в}}$ – середньозважений час знаходження контейнера в непрацездатному стані,

N_0 – число контейнерів, які знаходяться в працездатному станів в момент початку спостереження,

Δt – інтервал часу.

Оскільки відмови контейнерів є випадковими подіями і піддаються експоненціального закону розподілу, достовірно визначити кількість контейнерів, що знаходяться в працездатному стані в довільний момент часу не є можливим. Для цього слід використовувати математичний апарат прогнозування числа контейнерів, що знаходяться в працездатному стані, $N(\Delta t)$ в залежності від середньозваженого часу переведення контейнера в непрацездатний стан і довжини часового інтервалу:

$$N(\Delta t) = N_0 \times \left(1 - \frac{n(\Delta t) \times t_{\text{в}}}{\Delta t}\right) \tag{3.6}$$

де $n(\Delta t)$ – число контейнерів, які знаходяться в непрацездатному стані на інтервалі часу Δt,

$t_{\text{в}}$ – середньозважений час знаходження контейнера в непрацездатному стані.

N_0 – число контейнерів, які знаходяться в працездатному стані на момент початку спостереження,

Δt – інтервал часу.

Таким чином, середнє число контейнерів, що знаходяться в працездатному стані на заданому інтервалі часу Δt можна визначити за:

$$N_{ср} = \frac{N_0 + N_{(\Delta t)}}{2} \tag{3.7}$$

де N_0 – число контейнерів, що знаходяться в працездатному стані но момент початку інтервалу спостереження,

$N_{(\Delta t)}$ – число контейнерів, які знаходяться в працездатному стані на момент закінчення інтервалу спостереження.

Оскільки контейнер є легким у порівнянні з класичною віртуальною машиною, процес відновлення починається відразу ж після відмови, тому час перебування контейнера в непрацездатному стані буде тотожно часу його відновлення. Далі в тексті під tв мається на увазі середньозважене час відновлення пристрою. Величина tв визначається на підставі даних спостережень за масивом відповідних контейнерів в даному інтервалі часу.

Для визначення середньозваженого часу відновлення на підставі зібраних статистичних даних проводиться побудова графіка апроксимуючої функції, яка описує відповідний закон розподілу величини спостережуваного часу відновлення в діапазоні X1 –Xi , Де X - час відновлення,. Таким чином, $t_{\text{в}}$ визначається за формулою:

$$t_{\text{в}} = \frac{1}{i}\int_{1}^{i} f(i) \tag{3.8}$$

де i – випадків відновлення пристрою в розглянутому періоді,
$f(i)$ – функція розподілення часу відновлення.

Відмови, що утворюють потік, розподілені по закону Пуассона. Оскільки відмова пристрою може мати місце лише під час його знаходження в справному стані (до або після відновлення), то потік відмов носить експонентний характер. Імовірність безвідмовної роботи в момент часу t в цьому випадку визначається за формулою:

$$P(t) = e^{-\lambda \times t} \tag{3.9}$$

де λ – інтенсивність потоку відмов,
t – момент часу, в який визначається імовірність відмови.

Фізичний зміст миттєвого $K_{\text{г}}$ пристрої готовності є ймовірністю перебування пристрою в працездатному стані в розглянутий момент часу, на підставі (3.9) можна записати (3.10):

$$K_{\text{г}}(t) = e^{-\lambda \times t} \tag{3.10}$$

Середній Кг на інтервалі з t_1 до t_2 визначається за []:(2.11),

$$K_{\text{г}} = \frac{1}{t_2 - t_1}\int_{t1}^{t2} K_{\text{г}}(t)dt \tag{3.11}$$

де t_1 – початок інтервалу часу,
t_2 – кінець інтервалу часу.

Підставляючи (3.10) в (3.11), можна отримати вираз для середнього Кг на інтервалі часу з t_1 до t_2 (3.12):

$$K_{\text{г}} = \frac{1}{t_2 - t_1} \int_{t1}^{t2} e^{-\lambda \times t} dt \qquad (3.12)$$

Залежність (3.12) дозволяє обчислити $K_{\text{г}}$ контейнера на підставі даних про кількість відмов і часу перебування його в непрацездатному стані на обмеженому часовому інтервалі спостереження. Отримані дані можна використовувати для прогнозування значень показників надійності на довільному часовому інтервалі.

Таким чином, використання математичного та методичного апарату теорії надійності обґрунтовано для опису випадкових процесів які викликають перехід контейнерів з працездатного стану в непрацездатний. Однак використання експоненціального закону розподілу для дослідження впливу загроз інформаційної безпеки, спрямованих на порушення доступності інформації, вимагає додаткового обґрунтування, оскільки недолік даних про інциденти (частоті їх виникнення і довжині) не дозволяє зробити однозначний висновок про той чи інший характер розподілу.

3.3 Вплив загрозі інформаційної безпеки на Кг Docker мереж

Для оцінки впливу загроз інформаційної безпеки на характеристики Docker мереж слід розробити математичну модель, що описує зміну станів її складових, щоб за допомогою вищеописаних методів, застосувати отриману математичну модель до Docker мережі в цілому. Для виконання цього завдання слід детально вивчити механізм впливу загроз інформаційної безпеки на Кг системи і визначити стани елемента мережі в яких він може перебувати в умовах виникаючих відмов і загроз інформаційній безпеці.

Оскільки для оцінки стану Docker мереж обраний Кг, то слід розробляти модель впливу загроз інформаційній безпеці, спрямованих на доступність інформації, на Кг елемента досліджуваної системи.

Під загрозою безпеки інформації розуміється сукупність умов і чинників, що створюють потенційну або реально існуючу небезпеку порушення безпеки інформації []. Оскільки по цілі впливу на інформаційну систему загрози інформаційної безпеки діляться на загрози цілісності, конфіденційності та доступності циркулюючої в інформаційній системі інформації []. Таким чином,

загрози інформаційної безпеки класифікуються в залежності від властивостей інформації, на порушення яких вони спрямовані.

Успішно реалізована загроза доступності інформації в інформаційній системі, на відміну від загроз, спрямованих на інші властивості інформації (цілісність і конфіденціальність), призводить до простою атакованих інформаційних служб і, як до крайнього варіанту, простою всієї інформаційної системи. Час простою (знаходження в непрацездатному стані) елементів інформаційної системи можливо оцінити кількісно та включити його в розрахунок Кг кожного розглянутого елемента як одну з складових часу відновлення працездатності елемента, що дозволить надалі оцінити ефективність його функціонування.

Оскільки Кг є нормованої характеристикою, то його мінімальне значення обмежено відповідною нормою. Таким чином, завдання визначення актуальності розглянутих загроз інформаційної безпеки і достатності прийнятих захисних заходів може бути вирішена математично.

3.4 Математична модель Docker мережі як сукупності елементів

Як визначено в розділі 1 цього дослідження, Docker мережа складається з сукупності пов’язаних між собою контейнерів, які беруть участь в обробці інформації. Кожен контейнер містить у собі власну логіку обробки інформації і, таким чином, володіє власними показниками, що характеризують його надійність. Показником надійності, який використовується для побудови математичної моделі обраний Кг. Кг є характеристикою кожного контейнера. Також визначено, що потік інформації обробляється всіма контейнерами мережі, які задіяні в інформаційному обміні, послідовно. Таким чином, для розрахунку Кг інформаційної мережі можна записати (3.1):

$$К_{г^у} = К_{г_1} \times К_{г_2} \times \ldots \times К_{г_n} \qquad (3.13)$$

де $К_{г_1} \ldots К_{г_n}$ – Кг контейнера в Docker мережі.

3.5 Марківска модель надійності для Docker мережі, яка враховує вплив загроз доступності інформації

Процес впливу загроз інформаційної безпеки на інформаційну мережу не є певним за своїми параметрами. Потоки подій, які призводять до виникнення загроз інформації, спрямованих на порушення доступності інформації в мережі, націлених на досліджувані вузли інформаційної мережі,

є в цьому процесі керуючими. Крім загроз інформаційної безпеки, контейнери в Docker мережі схильні до відмов, пов'язаних з помилками в конфігурації програмного забезпечення, розміщеному в них. Потоки розглянутих відмов незалежно від причин їх виникнення характеризується експоненціальним законом розподілу[].

Допускаючи, що потоки подій, які керують станами досліджуваного вузла Docker мережі, мають експонентний характер, для визначення ймовірностей безвідмовної роботи вузлів зв'язку можливо використовувати математичний апарат марківських випадкових процесів, що характеризується безперервним часом і дискретними станами. В рамках даного дослідження цими станами є :

1. Вузол Docker мережі знаходиться в стані готовності.
2. Вузол Docker мережі знаходиться в стані неготовності через реалізацію загрози інформаційної безпеки, спрямованої на порушення доступності інформації.
3. Вузол Docker мережі знаходиться в стані неготовності через технічні відмови обладнання.

Граф описаних станів вузла Docker мережі наведено на рисунку 6.

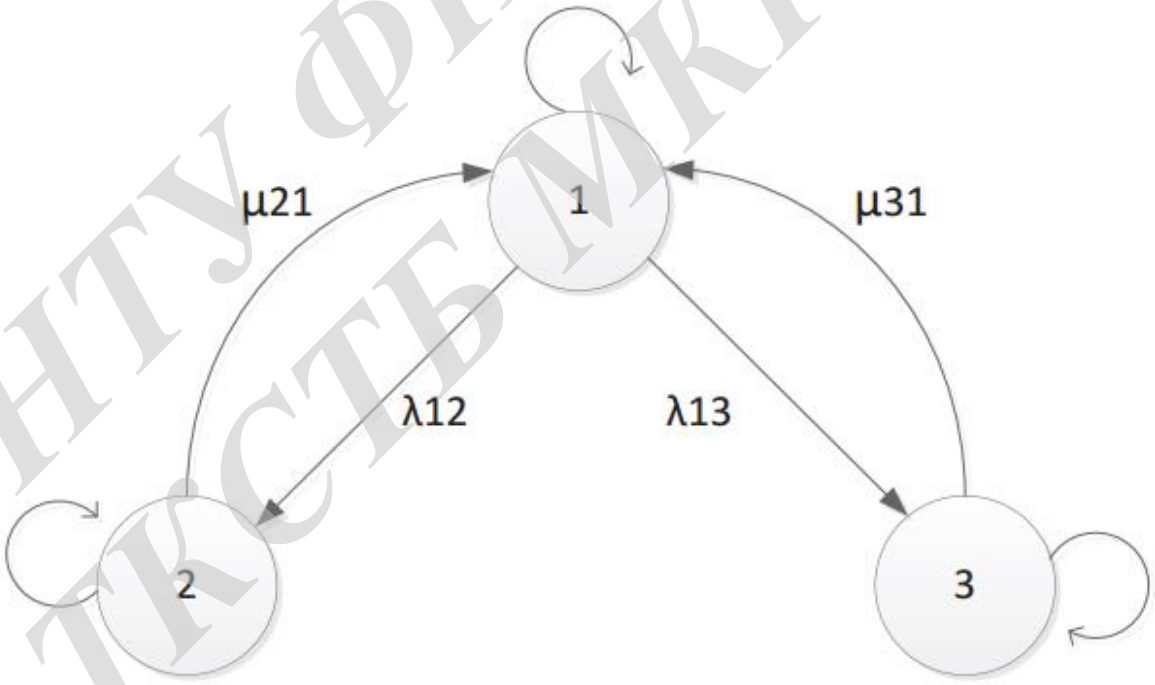


Рисунок 6 – Граф станів вузла Docker мережі

Математична модель процесу, що визначається графом (рісунок), буде описуватися системою рівнянь Колмогорова-Чепмена:

$$\begin{cases} \dfrac{dP_1(t)}{dt} = -(\lambda_{12} + \lambda_{13}) \times P_1(t) + \mu_{21} \times P_2(t) + \mu_{31} \times P_3(t) \\ \dfrac{dP_2(t)}{dt} = \lambda_{12} \times P_1(t) - \mu_{21} \times P_2(t) \\ \dfrac{dP_3(t)}{dt} = \lambda_{13} \times P_1(t) - \mu_{31} \times P_3(t) \end{cases} \quad (3.14)$$

де $P_1(t)$, $P_2(t)$, $P_3(t)$ – вірогідність знаходження вузла зв'язку Docker мережі в першому, другому і третьому станах, якщо від початку процесу пройшов період часу (t);

λ_{12}, λ_{13} – інтенсивність потоків відмов в Docker мережі;

μ_{21}, μ_{31} – інтенсивність потоків відновлення в Docker мережі.

Крім того, оскільки вузол зв'язку в інформаційні мережі весь аналізований період часу знаходиться тільки в одному з трьох описаних вище станів, слід ввести нормувальну умову до системи рівнянь 3.14 (3.15):

$$P_1(t) + P_2(t) + P_3(t) = 1 \quad (3.15)$$

В рамках даного дослідження процеси в системі є сталими, тобто слід прийняти:

$$\frac{dP_i(t)}{dt} = 0 \quad (3.16)$$

де $P_i(t)$ – вірогідність знаходження вузла Docker мережі в одном із станів.

В цьому випадку система диференціальних рівнянь (3.14) з урахування нормувальної умови (3.15) і умови стаціонарності (3.16) перетворюється в систему алгебраїчних рівнянь (3.17):

$$\begin{cases} -(\lambda_{12} + \lambda_{13}) \times P_1(t) + \mu_{21} \times P_2(t) + \mu_{31} \times P_3(t) = 0 \\ \lambda_{12} \times P_1(t) - \mu_{21} \times P_2(t) = 0 \\ \lambda_{13} \times P_1(t) - \mu_{31} \times P_3(t) = 0 \\ P_1(t) + P_2(t) + P_3(t) = 1 \end{cases} \quad (3.17)$$

Вирішуючи систему рівнянь (3.17) отримаємо формули ймовірності знаходження вузла зв'язку в одному з станів (3.18) – (3.20) :

$$P_1 = \frac{\mu_{21} \times \mu_{31}}{\mu_{21} \times \mu_{31} + \lambda_{12} \times \mu_{31} + \lambda_{13} \times \mu_{21}} \quad (3.18)$$

$$P_2 = \frac{\mu_{12} \times \mu_{31}}{\mu_{21} \times \mu_{31} + \lambda_{12} \times \mu_{31} + \lambda_{13} \times \mu_{21}} \quad (3.19)$$

$$P_3 = \frac{\mu_{13} \times \mu_{21}}{\mu_{21} \times \mu_{31} + \lambda_{12} \times \mu_{31} + \lambda_{13} \times \mu_{21}} \quad (3.20)$$

Таким чином, ймовірність безвідмовної роботи Docker мережі, при умові впливу на неї виключно загроз інформаційної безпеки, відповідає ймовірності непотрапляння її в стан 2 і розраховується за формулою (3.21):

$$P_2 = \frac{\mu_{21} \times (\lambda_{13} + \mu_{31})}{\mu_{21} \times \mu_{31} + \lambda_{12} \times \mu_{31} + \lambda_{13} \times \mu_{21}} \quad (3.21)$$

Вираз (3.21) являє собою Кг вузла Docker мережі, що враховує вплив загроз інформаційної безпеки, спрямованих на порушення доступності інформації, з допущенням відсутність технічних відмов. Розрахунки величин λ_{12}, λ_{13}, μ_{21}, μ_{31} проводяться по (3.4) – (3.6).

3.6 Дослідження впливу загроз доступності інформації

Як визначено раніше, не завжди є можливість використовувати для розрахунків статистичні дані про параметри потоків відмов і відновлень працездатності елемента інформаційної мережі. У такому випадку доцільно використовувати математичну модель, описану нижче, що дозволяє здійснити розрахунок значення $К_{нг}$, обумовленого впливом загроз інформаційної безпеки, з використанням результатів експеримента. Таким чином, можливо здійснити розрахунок значення P_2 ($К_{нг}$, обумовленого впливом загроз інформаційної безпеки) в умовах нестачі даних про інциденти інформаційної безпеки (зокрема, для проектування Docker мережі).

Оскільки результат реалізації загроз інформаційної безпеки, спрямованих на порушення доступності інформації, являє собою непрацездатність сервісів, то можливо знайти вплив загроз доступності інформації в $К_г$ вузла зв'язку. Наслідком реалізованої загрози інформаційної безпеки є простій мережевих служб, пов'язаний із зовнішнім впливом, а також

час, необхідний для відновлення працездатності після припинення зовнішнього деструктивного впливу. Таким чином, вплив загроз інформаційної безпеки, спрямованих на порушення доступності інформації, можна привести до часу, протягом якого Docker мережа знаходиться в стані неготовності з причин, викликаних реалізацією загроз інформаційної безпеки.

Для того, щоб загроза інформаційної безпеки викликала порушення доступності повинні відбутись такі пов'язані між собою події:

- загроза інформаційної безпеки повинна виникнути, тобто зловмисники повинні провести ряд технічних заходів, спрямованих на порушення доступності Docker мережі;

- загроза інформаційної безпеки повинна бути реалізована на атакованому вузлі, тобто системи захисту інформації, призначені для протидії і нейтралізації загроз, повинні не виконати своє призначення;

- наслідком реалізації загрози інформаційної безпеки став часовий період, протягом якого вузол Docker контейнер знаходився в стані неготовності.

Таким чином, для вирішення поставленого завдання необхідно визначити коефіцієнт неготовності $К_{нг}$, який відповідає стану 2 моделі вузла зв'язку, зображеної на рисунку. Коефіцієнт неготовність буде складатися з трьох складових [39]:

- $Р_в$, ймовірність виникнення загрози інформаційної безпеки, що характеризує потік виникаючих загроз інформаційної безпеки, метою яких є порушення доступності вузлів;

- $Р_р$, ймовірність реалізації загрози інформаційної безпеки, що характеризує недосконаліть застосовуваних систем захисту інформації;

- $К_{нг_{у(і)}}$, $К_{нг}$, викликаний реалізацією загрози інформаційної безпеки, що характеризує час знаходження контейнера в стані неготовності, викликаної реалізацією загрози інформаціійних безпеки, спрямованої на порушення доступності.

Для того, щоб загроза інформаційної безпеки вплинула на $К_г$ Docker мережі, всі перераховані вище умови повинні виконуватися одночасно. Оскільки всі три показники ($Р_в$, $Р_р$ і $К_{нг_{у(і)}}$) мають ймовірнісну природу і описані вище події повинні відбутися одночасно, то загальна ймовірність настання всіх трьох подій ($К_{нг}$, пов'язаний із загрозою інформаційної безпеки) буде розраховуватися за (3.22) відповідно до теореми про ймовірність залежних подій [140].

$$К_{нг_{у(і)}} = Р_в \times Р_{р|в} \times К_{нг}^{р.у} \qquad (3.22)$$

Виходячи з фізичного змісту, імовірнісні величини $К_г$ і $К_{нг}$ є взаємопов'язаними [36]. Таким чином, можна записати (3.23):

$$К_{г_у} = 1 - К_{нг_у} \quad (3.23)$$

Підставивши (3.22) в (3.23) отримаємо (3.24):

$$К_{г_у} = 1 - Р_в \times Р_{р|в} \times К_{нг}^{Р.У} \quad (3.24)$$

Для визначення $К_{нг}$, що відображає вплив загроз інформаційної безпеки, необхідно визначити час, протягом якого вузол зв'язку буде простоювати внаслідок реалізації загроз інфомаційної безпеки. Реалізовану загрозу слід розглядати як збільшення часу простою мережі. Для вирішення практичного завдання визначення $К_{нг_у}$ слід використовувати результати спостереження за станом мережі і моделювання поведінки системи при реалізації загроз інфомаційної безпеки.

У разі, коли модель загроз передбачає, що вузол зв'язку схильний до більш ніж одної загрози інформаційної безпеки, $К_{нг}$ буде розраховуватися відповідно до (3.1). В результаті розрахунок $К_{нг}$ повинен проводитися по (3.25):

$$К_{нг}^{Р.У} = 1 - К_{г_{у(1)}} \times К_{г_{у(2)}} \times \ldots \times К_{г_{у(n)}} \quad (3.25)$$

де $К_{г_{у(1)}}$ ... $К_{г_{у(n)}}$ – $К_г$, що відображають вплив загроз інформаційної безпеки відповідно до прийнятої моделі загроз.

Величини $К_{г_{у(1)}}$ ... $К_{г_{у(n)}}$ розраховуються за (3.23) – (3.24).

Шляхом аналізу фізичного змісту і можливості впливу на ці величини, визначено наступне:

1. $К_{нг}^{Р.У}$ визначається часом перебування контейнера в стані неготовності. Цей час може визначатися двома способами. У першому випадку, це буде часовий інтервал, який потрібно для активізації систем захисту інформації, призначених для боротьби з відповідними загрозаим інформаційної безпеки або активізації рішення для аутсорсингу захисту від атак типу «відмова в обслуговуванні». У другому випадку, тривалість часу простою буде визначатися тривалістю атаки, цей випадок характеризує ситуацію, коли штатні системи для захисту інфомації не впоралися з реалізованої атакою і не змогли їй протистояти. В деяких випадках [] рекомендується використовувати

аутсорсинг захисту від атак типу «відмова в обслуговуванні» для забезпечення інформаційної безпеки загальнодоступних ресурсів, які встановлюють зв'язок з невизначеним числом хостів (наприклад, веб - серверів). У разі контейнеризації, найбільш перспективним варіантом є фільтрація вхідних запитів і пропуск на Docker контейнер, що атакується, тільки тих запитів, які виробляються з заздалегідь визначених адрес сусідніх контейнерів. Інші запити ігноруються, і обчислювальні потужності контейнера не утилізуються. Такий підхід добре зарекомендував себе в протидії атак типу «Відмова в обслуговуванні» малої і середньої інтенсивності, спрямованих на виснаження обчислювальної потужності Docker мережі. Таким чином, у атакованій системі можливість впливати на час знаходження Docker контейнера в непрацездатному стані фактично відсутня. У разі успішної реалізації атаки, цей час буде визначатися тільки можливостями зловмисника. Значення Кнгр.у буде характеризуватися часом знаходження вузла зв'язку в непрацездатному стані за досліджуваний період часу.

Значення $К_{нг}^{Р.У}$ у відповідності до (3.25) для стаціонарного показника залежить від відношення часу перебування контейнера в непрацездатному стані, викликаного загрозами інформаційної безпеки за певний період до довжини цього періоду [101]:

$$К_г = \frac{Т_р}{Т_п + Т_{нп}} \qquad (3.26)$$

де $Т_п$ – середня тривалість працездатного стану

$Т_{нп}$ – середня тривалість непрацездатного стану

У разі, коли відомий період непрацездатного стану за певний період $\Delta Т$, значення $Т_п$ і $Т_{нп}$ через тривалість періоду спостережень визначається за (3.27):

$$\Delta Т = Т_п + Т_{нп} \qquad (3.27)$$

Підставивши (3.26) і (3.27) в (3.25) отримаємо формулу визначення Кнг на підставі даних про знаходження вузла зв'язку в стані неготовності за певний період часу (3.28):

$$К_{нг}^{Р.У} = \frac{Т_{нп}}{\Delta Т}$$

Пропонований метод розрахунку величин $К_{нг}$ і $К_г$ не забезпечує високої точності обчислення [132], проте його застосування виправдане в ситуації, коли потік відмов не піддається детермінації і узагальнення у вигляді закону його розподілу через мале число, нерегулярність відмов і повну невизначеність у часі знаходження вузла зв'язку в непрацездатному стані.
Таким чином, представляється можливим розрахувати Кг контейнера з урахуванням впливу загроз інформаційної безпеки на підставі відомого часу, що витрачається на відновлення працездатності контейнера після реалізації загроз інформаційної безпеки.

2. $P_в$ загрози інформаційної безпеки, спрямованої проти досліджуваної Docker мережі, залежить тільки від привабливості мети для зловмисників. У разі великої мережі, значення $P_в$ наближається до 1. Таким чином, приймаючи той факт, що досліджувана Docker мережа достовірно буде піддаватися атакам зловмисників, за умови однакової привабливості всіх контейнерів для атак, $P_в$ загрози інформаційної безпеки для досліджуваної мережі буде залежати від загального числа контейнерів мережі, які доступні атакуючим, і загального числа атак за досліджуваний період. Таким чином, можливість впливати на значення $P_в$ з боку захищаються ресурсів також відсутня.

$$P_в = \begin{cases} \frac{a}{n_y}, & a < n \\ 1, & a \geq n \end{cases} \qquad (3.28)$$

де a – число атак, спрямованих на досліджувану Docker мережу у розглянутий період,
n_y – середнє число контейнерів Docker мережі у розглянутий період, доступних для атаки.

Залежність (3.28) дозволяє виконати розрахунок значення $P_м$ за наявними даними про кількість інцидентів інформаційної безпеки, викликаних розглянутими загрозами, і інформації про загальну кількість контейнерів Docker мережі.

У разі нестачі статистичних даних про інциденти інформаційної безпеки, допустимо в розрахунках виходити з негативного припущення про величину $P_м$. припущення, що $P_м$ = 1 відображає те, що досліджуваний контейнер Docker мережі достовірно піддається атакам зловмисників, що відображає найбільш негативний сценарій розвитку подій. Даний сценарій розроблений на підставі відомостей про динаміку атак типу «відмова в обслуговуванні »на публічні мережеві адреси великих компаній [46, 47, 48].

3. $P_р$ загрози інформаційної безпеки, спрямованої проти Docker мережі, що захищається за допомогою систем захисту інформації, встановлених безпосередньо на сервері буде залежати в першу чергу від досконалості захисних механізмів. Таким чином, $P_р$ є єдиним доступним показником для впливу на величину $К_{нг}^{Р.У}$ з боку Docker мережі що атакується.

ПОКРАЩЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ DOCKER МЕРЕЖ

Оскільки існує ряд основних загроз безпеці Docker мереж, виникає необхідність їх усунення та виявлення при конфігуруванні середовища та його подальшій роботі. Ручна конфігурація Docker Engine можлива при невеликій кількості серверів та малому навантаженні на систему, але при розширенні інфрастуктури керування безпекою Docker мереж значно ускладнюється, що веде до необхідності створення автоматичного інструмента для перевірки та конфігурації безпеки мережі в цілому.

Так як класичні брандмауери не здатні виявити описані вище загрози, оскільки не володіють функціоналом для перевірки контейнерів, образів та дозволів, які можуть їм надаватись, найбільш правильним варіантом є створення простого інструменту, який би перевіряв Docker мережу на наявність основних прогалин у конфігурації безпеки.

Програма для перевірки конфігурації безпеки буде розроблена на скриптовій мові bash і буде базуватись на описаних вище основних загрозах безпеці.

Програма потребує привілейованого доступу до системи, тому перш за все необхідно перевірити чи запущена вона під root користувачем:

```
ID=$(id -u)
if [ "x$ID" != "x0" ]; then
  echo "Будь ласка, запустіть програму використовуючи root"
fi
```

Дана частина коду буде виводити попередження, якщо програма запущена без root прав.

При перевірці важливо, щоб Docker був запущений на сервері, що перевіряється наступним кодом:

```
if ! docker ps -q >/dev/null 2>&1; then
  printf "Помилка з'єднання з docker\n"
  exit 1
```

Оскільки з виходом нових версій Docker виходять оновлення безпеки та постійно виправляються знайдені недоліки, необхідно перевіряти чи використовується на сервері найновіша версія програмного забезпечення:

```
docker_version=$(docker version | grep -i -A2 '^server' | grep ' Version:' \
  | awk '{print $NF; exit}' | tr -d '[:alpha:]-,')
docker_current_version="$(date +%y.%m.0 -d @$(( $(date +%s) - 2592000)))"
do_version_check "$docker_current_version" "$docker_version"
if [ $? -eq 11 ]; then
  echo "Ви використовуєте застарілу версію docker"
else
  echo "Ви використовуєте найновішу версію docker"
fi
```

Важливо слідкувати за тим, які користувачі сервера мають право на конфігурування Docker, даний код виводить на екран усіх користувачів, які мають права на використання Docker:

```
docker_users=$(getent group docker)
echo "docker використовується такими користувачами:"
for u in $docker_users; do
  echo "$u \t"
done
  echo "\n"
```

Для забезпечення достовірності Docker образів рекомендовано використовувати функцію Docker Content Trust. Docker Content Trust дозволяє використовувати цифрові підписи для даних, що надсилаються та отримуються з віддалених реєстрів Docker. Ці підписи дозволяють перевірити цілісність і автора конкретних образів на стороні клієнта або під час виконання. Щоб перевірити чи увімкнено Docker Content Trust використовується наступний програмний код:

```
if [ "x$DOCKER_CONTENT_TRUST" = "x1" ]; then
  echo "Docker Content Trust увімкнено\n"
else
  echo "Docker Content Trust вимкнено\n"
fi
```

Для перевірки того, чи працює певний контейнер у мережі використовується команда healthcheck. Дана команда повідомляє Docker, як перевірити працездатність контейнерів. Не вказуючи жодної перевірки, Docker не може дізнатися, чи дійсно контейнер запущений та працює так як вимагається. Перевірка чи зконфігурована дана команда виконується наступним кодом:

```
fail=0
no_health_images=""
for img in $images; do
  if docker inspect --format='{{.Config.Healthcheck}}' "$img" 2>/dev/null
  | grep -e "<nil>" >/dev/null 2>&1; then
    if [ $fail -eq 0 ]; then
      fail=1
    fi
    imgName=$(docker inspect --format='{{.RepoTags}}' "$img" 2>/dev/null)
    if ! [ "$imgName" = '[]' ]; then
  no_health_images="$no_health_images $imgName"
    fi
  fi
done
if [ $fail -eq 0 ]; then
  echo "healthcheck зконфігуровано вірно \n"
else
  echo "наявні проблеми в конфігурації healthcheck \n"
fi
```

Docker надає дві команди для копіювання файлів з хоста на образ Docker під час його створення: COPY та ADD. Інструкції схожа за своєю суттю, але відрізняються своїм функціоналом.

COPY - копіює локальні файли рекурсивно, приймає як аргумент точно вказані шляхи до файлів чи каталогів.

ADD - копіює локальні файли рекурсивно, неявно створює каталог призначення, коли його не існує, і приймає як архіви так і віддалені URL-адреси в якості аргументів, які він розпаковує або завантажує відповідно в каталог призначення.

Якщо віддалені URL-адреси використовуються для завантаження даних безпосередньо у образ, вони можуть спричинити атаки типу "людина посередині", що призведе до зміни вмісту завантажуваного файлу. Більше того, походження та достовірність віддалених URL-адрес потрібно додатково підтверджувати. При використанні COPY джерело файлів, що завантажуються з віддалених URL-адрес, можна оголосити через захищене TLS-з'єднання, що дозволить уберегтись від атак при створенні образу. Наступний код перевіряє чи використовується команда ADD замість команди COPY:

```
fail=0
add_images=""
for img in $images; do
  if docker history --format "{{ .CreatedBy }}" --no-trunc "$img" | \
    sed '$d' | grep -q 'ADD'; then
    if [ $fail -eq 0 ]; then
      fail=1

    fi
    imgName=$(docker inspect --format='{{.RepoTags}}' "$img" 2>/dev/null)
    if ! [ "$imgName" = '[]' ]; then
      add_images="$add_images $imgName"
    fi
  fi
done
if [ $fail -eq 1 ]; then
  resulttestjson "Спробуйте використати команду COPY замість ADD \n"
fi
```

При управлінні Docker мережами віддалено, виникає необхідність верифікації користувача та шифрування введених команд. Для такої цілі найкраще всього підходить протокол TLS (Transport Layer Security), який надає можливості автентифікації і безпечної передачі даних через Інтернет з використанням криптографічних засобів. Для перевірки правильної роботи такого протоколу можна використати наступний код:

```
if [ grep -i 'tcp://' "$CONFIG_FILE" 2>/dev/null 1>&2 ] || \
  [ $(get_docker_cumulative_command_line_args '-H' | grep -vE '(unix|fd)://') >/dev/null 2>&1 ];
  then
  if [ $(get_docker_configuration_file_args '"tlsverify":' | grep 'true') ] || \
      [ $(get_docker_cumulative_command_line_args '--tlsverify' | grep 'tlsverify') >/dev/null 2>&1 ];
      then

    echo "TLS сконфігуровано вірно \n"

  elif [ $(get_docker_configuration_file_args '"tls":' | grep 'true') ] || \
      [ $(get_docker_cumulative_command_line_args '--tls' | grep 'tls$') >/dev/null 2>&1 ];
      then
    echo "Docker daemon працює з TLS через TCP, але не отримав верифікації \n"
  else
    echo "Docker daemon працює через TCP без використання TLS \n"
  fi
else
  echo "Docker daemon не працює через TCP \n"
fi
```

Можливість запускати велику кількість контейнерів на одному фізичному сервері це одна з ключових переваг інфраструктури Docker. Однак з розвитком мікросервісної архітектури та постійними змін в конфігуранції прогамного забезпечення стає важко ефективно керувати контейнерами. У процесі тестування та збірки готових контейнерів дуже часто доводить створювати по декілька схожих образів та контейнерів, що ускладнює процес

керування ними та може призвести до випадкового створення контейнера із застарілого образу або призвести до варіанту, коли адміністратор системи не знає скільки контейнерів взагалі запущено. Таким чином є доцільним додати в програму код, який буде перевіряти чи запущено з певних докер образів якийсь контейнер та чи існують у системі неактивні контейнери, що допоможе у їх видалені та зменшить імовірність запуску контейнерів із застарілих або тестових образів. Код програми для перевірки наявності невикористовуваних образів має такий вигляд:

```
images=$(docker images -q | sort -u | wc -l | awk '{print $1}')
active_images=0

for c in $(docker inspect --format "{{.Image}}" $(docker ps -qa) 2>/dev/null); do
  if docker images --no-trunc -a | grep "$c" > /dev/null ; then
    active_images=$(( active_images += 1 ))
  fi
done

  echo   "В системі присутньо: $images образів \n"

if [ "$active_images" -lt "$((images / 2))" ]; then
  echo   " Лише $active_images з $images використовується \n"
fi
```

Код програми для перевірки наявності неактивних контейнерів:

```
total_containers=$(docker info 2>/dev/null | grep "Containers" | awk '{print $2}')
running_containers=$(docker ps -q | wc -l | awk '{print $1}')
diff="$((total_containers - running_containers))"
if [ "$diff" -gt 25 ]; then
  echo "В системі присутньо $total_containers контейнерів,
  з яких лише $running_containers є активними \n"
else
  echo "В системі присутньо $total_containers контейнерів,
  з них $running_containers активні \n"
fi
```

4. ЕКОНОМІЧНА ЧАСТИНА

4.1 Визначення рівня комерційного потенціалу підвищення інформаційної безпеки Docker мереж

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу підвищення інформаційної безпеки Docker мереж, створеної в результаті науково-технічної діяльності. В результаті оцінювання можна буде зробити висновок щодо напрямів (особливостей) організації подальшого її впровадження з врахуванням встановленого рейтингу.

Для проведення технологічного аудиту залучимо 3-х незалежних експертів. У нашому випадку такими експертами будуть керівник магістерської роботи та провідні викладачі випускової та споріднених кафедр.

Оцінювання комерційного потенціалу підвищення інформаційної безпеки Docker мереж будемо здійснювати за 12-ю критеріями згідно рекомендацій.

Результати оцінювання комерційного потенціалу підвищення інформаційної безпеки Docker мереж заносимо до таблиці 1.1.

Таблиця 1.1. - Результати оцінювання комерційного успіху підвищення інформаційної безпеки Docker мереж

Критерії	Експерти		
	д.т.н., професор Семенов А.О.	д.т.н., професор Осадчук О.В.	к.т.н., доцент Гаврілов Д.В.
	Бали, виставлені експертами		
1	2	2	2
2	3	1	3
3	2	2	3
4	3	1	2
5	3	2	3
6	2	2	2

7	3	2	3
8	2	2	2
9	3	2	1
10	3	3	3
11	2	2	3
12	3	3	2
Сума балів	31	24	29
Середньоарифметична сума балів, СБ	28		

За даними таблиці 1.1 робимо висновок щодо рівня комерційного потенціалу підвищення інформаційної безпеки Docker мереж. При цьому користуємося рекомендаціями, наведеними в таблиці 1.2.

Таблиця 1.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 50	Високий

Таким чином, робимо висновок, щодо рівня комерційного потенціалу нашої підвищення інформаційної безпеки Docker мереж – середній.

1.2 Визначення рівня якості підвищення інформаційної безпеки Docker мереж

Оцінювання рівня якості підвищення інформаційної безпеки Docker мереж проводиться з метою порівняльного аналізу і визначення найбільш ефективного, з технічної точки зору, варіанта інженерного рішення.

Рівень якості – це кількісна характеристика міри придатності певного виду продукції для задоволення конкретного попиту на неї при порівнянні з відповідними базовими показниками за фіксованих умов споживання.

Абсолютний рівень якості підвищення інформаційної безпеки Docker мереж знаходимо обчисленням вибраних для її вимірювання показників, не порівнюючи їх із відповідними показниками аналогічних виробів. Для цього необхідно визначити зміст основних функцій, які повинні реалізовувати розробка, вимоги замовника до неї, а також умови, які характеризують експлуатацію, визначають основні параметри, які будуть використані для розрахунку коефіцієнта технічного рівня виробу. Система параметрів, прийнята до розрахунків, повинна достатньо повно характеризувати споживчі властивості інноваційного товару (його призначення, надійність, економічне використання ресурсів, стандартизація тощо).

Далі визначаємо величину параметрів якості в балах та встановлюємо граничні його значення (кращі, гірші, середні). Всі ці дані для кожного параметра заносимо в табл. 1.3.

Таблиця 1.3 – Основні параметри підвищення інформаційної безпеки Docker мереж

Параметри	Абсолютне значення параметра			Коефіцієнт вагомості параметра
	Краще +5…+4	Середнє +3	Гірше +1…+2	
Точність вимірювання			2	0,1
Кількість вимірювальних каналів			2	0,1
Діапазон вимірювання		3		0,7
Відносна похибка		3		0,1

Із врахуванням коефіцієнтів вагомості відповідних параметрів можна визначити абсолютний рівень якості інноваційного рішення за формулою:

$$К_{я.а.} = \sum_{i=1}^{n} Рні \cdot аі, \quad (1.1)$$

де Рні – числове значення і-го параметра інноваційного рішення, n – кількість параметрів інноваційного рішення, що прийняті для оцінювання, аі – коефіцієнт вагомості відповідного параметра (сума коефіцієнтів вагомості всіх параметрів повинна дорівнювати 1).

Отже, абсолютний рівень якості підвищення інформаційної безпеки Docker мереж становитиме – 2,8 бали.

Одночасно визначаємо відносний рівень якості підвищення інформаційної безпеки Docker мереж, що виробляється (проектується), порівнюючи її показники з абсолютними показниками якості найліпших вітчизняних та зарубіжних аналогів (товарів-конкурентів) (табл. 1.4).

Таблиця 1.4 – Основні параметри підвищення інформаційної безпеки Docker мереж та товару-конкурента

Параметри	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (конкурент)	Новий		
Точність вимірювання	2	1	0,5	0,1
Кількість вимірювальних каналів	2	1	0,5	0,1
Діапазон вимірювання	10	20	2	0,7
Відносна похибка	3	2	1,5	0,1

Відносний рівень якості підвищення інформаційної безпеки Docker мереж визначаємо за формулою:

$$К_{я.в.} = \sum_{i=1}^{n} qi \cdot ai, \quad (1.2)$$

За розрахунками відносний рівень якості підвищення інформаційної безпеки Docker мереж становитиме – 1,7. Це означає, що наша розробка краща за якістю на 70% від товару-аналога.

1.3 Визначення конкурентоспроможності підвищення інформаційної безпеки Docker мереж

У найширшому розумінні конкурентоспроможність товару – це можливість його успішного продажу на певному ринку і в певний проміжок часу. Водночас конкурентоспроможною можна вважати лише однорідну продукцію з технічними параметрами і техніко-економічними показниками, що ідентичні аналогічним показникам уже проданого товару. Для того, щоб

високоякісний товар був одночасно і конкурентоспроможним, він має відповідати критеріям оцінювання споживачів конкретного ринку в конкретний час.

Дані для розрахунку загального показника конкурентоспроможності розробки необхідно занести до таблиці 1.5.

Таблиця 1.5 – Нормативні, технічні та економічні параметри підвищення інформаційної безпеки Docker мереж і товару-конкурента

Параметри	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (конкурент)	Новий		
Точність вимірювання	2	1	0,5	0,1
Кількість вимірювальних каналів	2	1	0,5	0,1
Діапазон вимірювання	10	20	2	0,7
Відносна похибка	3	2	1,5	0,1
Ціна за продукт, тис. грн.	1000	850	1,06	-

Загальний показник конкурентоспроможності розробки (К) з урахуванням вищезазначених груп показників визначаємо за формулою:

$$К = \frac{Іт.п.}{Іе.п.} = \frac{1,7}{0,85} = 1,94, \quad (1.3)$$

де Іт.п. – індекс технічних параметрів (відносний рівень якості інноваційного рішення); Іе.п. – індекс економічних параметрів.

$$Іе.п. = \frac{РНеі}{РБеі} = \frac{850}{1000} = 0,85, \quad (1.4)$$

де РНеі, РБеі – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

Згідно розрахунків загальний показник конкурентоспроможності – 1,94 . Це означає, що наша підвищення інформаційної безпеки Docker мереж більш конкурентна на 94% від товару-аналога.

2. Прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи

2.1 Розрахунок витрат, що стосуються виконавців підвищення інформаційної безпеки Docker мереж

Основна заробітна плата кожного із розробників (дослідників) Зо, якщо вони працюють в наукових установах бюджетної сфери:

$$Зо = \frac{М}{Тр} \cdot t, \qquad (2.1)$$

де М – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.

У 2019 році величини окладів (разом з встановленими доплатами і надбавками) рекомендується брати в межах (5000…10000) грн. за місяць; Тр – число робочих днів в місяці; приблизно Тр = (21…23) дні; t – число робочих днів роботи розробника (дослідника).

Зроблені розрахунки зводимо до таблиці 2.1.

Таблиця 2.1 – Заробітна плата розробників

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	10000	455	5	2275
Інженер-програміст	7500	341	5	1705
Консультанти	5000	227	5	1135
Всього:				5115

Основна заробітна плата робітників Зр, якщо вони беруть участь у виконанні даного етапу роботи і виконують роботи за робочими професіями у випадку, коли вони працюють в наукових установах бюджетної сфери, розраховується за формулою:

$$Зр = \sum_{i=1}^{n} t_i \cdot Сі, \quad (2.2)$$

де ti – норма часу (трудомісткість) на виконання конкретної роботи, годин; n – число робіт по видах та розрядах; Сі – погодинна тарифна ставка робітника відповідного розряду, який виконує дану роботу. Сі визначається за формулою:

$$Сі = \frac{Мм \cdot Кі}{Тр \cdot Тзм}, \quad (2.3)$$

де Мм – розмір мінімальної заробітної плати за місяць, грн.; в 2019 році мінімальна заробітна плата становить – 4173 грн., Кі – тарифний коефіцієнт робітника відповідного розряду, Тр – число робочих днів в місяці; приблизно Тр = 21…23 дні; Тзм – тривалість зміни, зазвичай Тзм = 8 годин.

Величина чинних тарифних коефіцієнтів робітників відповідних розрядів для бюджетної сфери наведена в таблиці:

Розряд	1	2	3	4	5	6	7	8
Кі	1,00	1,09	1,18	1,27	1,36	1,45	1,54	1,64

Таблиця 2.2 – Заробітна плата робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Пайка підложки до корпусу	0,26	4	30,1	1,27	7,8
Пайка кристалу на підложку	0,36	4	30,1	1,27	10,8
Приварка виводів	0,4	4	30,1	1,27	12
Герметизація	1,0	2	25,8	1,09	25,8
Контроль	0,32	4	30,1	1,27	9,6
Вимірювання параметрів	1,1	2	25,8	1,09	28,4
Маркування	0,16	2	25,8	1,09	4,1
Всього					98,5

Додаткова заробітна плата Зд всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$Зд = 0{,}1 \cdot (Зр + Зо) = 0{,}1 \cdot (5115 + 98{,}5) = 521{,}4 \text{ грн.} \quad (2.4)$$

Нарахування на заробітну плату Нзп розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою: де Зо – основна заробітна плата розробників, грн.; Зр – основна заробітна плата робітників, грн.; Зд – додаткова заробітна плата всіх розробників та робітників, грн.; β – ставка єдиного внеску на загальнообов'язкове державне

соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

$$Нзп = 0{,}22 \cdot (Зр + Зо + Зд) = 0{,}22 \cdot (5115 + 98{,}5 + 521{,}4) =$$
$$= 1262грн. \quad (2.5)$$

Амортизація обладнання, комп'ютерів та приміщень А, які використовувались під час (чи для) виконання даного етапу роботи.

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

У спрощеному вигляді амортизаційні відрахування А в цілому бути розраховані за формулою:

$$А = \frac{Ц \cdot На}{100} \cdot \frac{Т}{12},$$

де Ц – загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн.; На – річна норма амортизаційних відрахувань. Для нашого випадку можна прийняти, що На = (10...25)%; Т – термін, використання обладнання, приміщень тощо, місяці.

Таблиця 2.3 - Амортизаційні відрахування

Найменування	Ціна, грн.	Норма амортизації, %	Термін використання, м.	Сума амортизації
Комп'ютер	7900	20	1	132
Лабораторний стенд	22000	20	1	367
Генератор сигналів	5500	20	1	92
Цифровий осцилограф	4900	20	1	82
Набір інструментів (відкрутка, пасатижі ...)	100	10	1	0,8
Паяльник	80	10	1	0,7

Всього	674,5

Витрати на матеріали М, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$М = \sum_1^n Нi \cdot Цi \cdot Кi, грн$$

де Нi – кількість матеріалу і-го виду, шт.; Цi – ціна матеріалу і-го виду, грн.; Кi – коефіцієнт транспортних витрат, Кi = (1,1…1,15); n – кількість видів матеріалів.

Таблиця 2.5 - Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість, грн.
Припой, кг	52,3	0,08	4,2
Герметик, кг	41,2	0,09	3,8
Спирт метиловий, кг	50,5	0,08	4
Фарба Ф-1,кг	65,2	0,09	5,9
Проволока, кг	38,9	0,06	2,3
Ізолятор, кг	47,1	0,05	2,4
Всього, з урахуванням коефіцієнта транспортних витрат	24,8		

Витрати на комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$К = \sum_1^n Нi \cdot Цi \cdot Кi, грн$$

де Нi – кількість комплектуючих і-го виду, шт.; Цi – ціна комплектуючих і-го виду, грн.; Кi – коефіцієнт транспортних витрат, Кi = (1,1…1,15); n – кількість видів комплектуючих.

Таблиця 2.5 - Комплектуючі, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість, грн.
Полікорова гібридна плата	18,3	1	18,3
Пермикач двопозиційний	14,8	1	14,8
НВЧ діод	15,4	1	15,4
Котушка індуктивності з феритовим осердям	4,65	1	4,65
Резистор	0,75	2	1,5
Конденсатор – 3,3 нФ	2,05	1	2,05
Конденсатор – 47 нФ	2,2	1	2,2
Роз'єми	8,1	2	16,2
Коаксіальний кабель з вхідним опором 50 Ом	8,4	1	8,4
Всього, з урахуванням коефіцієнта транспортних витрат	315		

Витрати на силову електроенергію Ве, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

$$Ве = В \cdot П \cdot Ф \cdot Кп, грн$$

В – вартість 1 кВт-год. електроенергії, в 2019 р. В ≈ 8,45 грн./кВт; П – установлена потужність обладнання, кВт; Ф – фактична кількість годин роботи обладнання, годин, Кп – коефіцієнт використання потужності; Кп < 1.

Потужність обладнання складає – 0,5 кВт.

Кількість годин роботи складає – 700 годин.

Коефіцієнт викор. потужності -0,9.

Ве=2662 грн.

Інші витрати Він охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо.

Інші витрати Ів можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$Ів = 1 \cdot (Зо + Зр) = 1 \cdot (5115 + 98{,}5) = 521{,}4 \text{ грн.} \quad (2.6)$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини (розділу, етапу) роботи – В.

$$В = 11195 \text{ грн.}$$

2.2 Розрахунок загальних витрат на підвищення інформаційної безпеки Docker мереж

Загальна вартість всієї наукової роботи визначається за Взаг формулою:

$$Взаг = \frac{Ів}{\alpha} = \frac{521{,}4}{0{,}6} = 869 \text{ грн,} \quad (2.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях.

2.3 Прогнозування витрат на виконання та впровадження підвищення інформаційної безпеки Docker мереж

Прогнозування загальних витрат ЗВ на виконання та впровадження підвищення інформаційної безпеки Docker мереж здійснюється за формулою:

$$ЗВ = \frac{Взаг}{\beta} = \frac{869}{0,5} = 1738 \text{ грн}, \quad (2.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Так, якщо розробка знаходиться: на стадії науково-дослідних робіт, то β ≈ 0,1; на стадії технічного проектування, то β ≈ 0,2; на стадії розробки конструкторської документації, то β ≈ 0,3; на стадії розробки технологій, то β ≈ 0,4; на стадії розробки дослідного зразка, то β ≈ 0,5; на стадії розробки промислового зразка, β ≈ 0,7; на стадії впровадження, то β ≈ 0,9.

3. Прогнозування комерційних ефектів від реалізації підвищення інформаційної безпеки Docker мереж

З метою прогнозування комерційних ефектів від реалізації підвищення інформаційної безпеки Docker мереж складемо таблицю вихідних показників, за рахунок яких і відбуватиметься отримання комерційного ефекту.

Таблиця 3.1 – Вихідні дані для прогнозування комерційного ефекту від реалізації підвищення інформаційної безпеки Docker мереж

Рік реалізації розробки	1	2	3
Кількість од. реалізації, шт.	200	500	700

Величина зростання ціни реалізації підвищення інформаційної безпеки Docker мереж, грн. – 150 грн.

Кількість продукції, що випускалась до впровадження підвищення інформаційної безпеки Docker мереж – 350 шт.

Збільшення чистого прибутку підприємства Пі для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta\text{Пі} = \sum_1^n (\Delta\text{Ц0} \cdot N + \text{Ц0} \cdot \Delta N)\text{і} \cdot \rho \cdot \gamma \cdot (1 - \frac{\upsilon}{100}) \qquad (3.1)$$

де ΔПо – покращення основного оціночного показника від впровадження результатів розробки у даному році. Зазвичай таким показником може бути ціна одиниці нової розробки; N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки; ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки; Цо – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки; n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки; λ– коефіцієнт, який враховує сплату податку на додану вартість. У 2018 р. ставка податку на додану вартість дорівнює 20%, а коефіцієнт – 0,8333. З 2014 року ставка податку на додану вартість встановлена на рівні 17%, а коефіцієнт – 0,8547; ρ– коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати – 0,2…0,3; υ– ставка податку на прибуток. У 2018 році – 21%, у 2013 році – 19%, а з 2014 року – 16%.

Збільшення чистого прибутку підприємства Пі протягом першого року складе:

ΔП1=17296 грн.

Збільшення чистого прибутку підприємства Пі протягом другого року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

ΔП2=82030 грн.

Збільшення чистого прибутку підприємства протягом третього року (відносно базового року, тобто року до впровадження результатів наукової розробки) складе:

ΔП3=111560 грн.

4. Розрахунок ефективності вкладених інвестицій та період їх окупності

4.1 Визначення абсолютної ефективності вкладених інвестицій у підвищення інформаційної безпеки Docker мереж

Для цього користуються формулою:

$$\text{Еабс} = (\text{ПП} - PV), \quad (4.1)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн.; PV – теперішня вартість інвестицій PV = ЗВ, грн.

У свою чергу, приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$\text{ПП} = \sum_1^{\text{т}} \frac{\Delta\text{Пі}}{(1+\tau)^t} \quad (4.2)$$

де ΔПі– збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн.; т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки; τ– ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1; t – період часу (в роках) від моменту отримання чистого прибутку до точки „0”.

$$\text{ПП} = 167334 \text{ грн.},$$

$$\text{Еабс} = 167334 - 1738 = 165596 \text{грн.}$$

Оскільки Еабс > 0, то результат від проведення наукових досліджень та їх впровадження принесе прибуток, але це також ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні підвищення інформаційної безпеки Docker мереж.

4.2 Розрахунок відносної ефективності вкладених коштів в НДДКР підвищення інформаційної безпеки Docker мереж

Для цього користуються формулою:

$$\text{Ев} = \sqrt[\text{Т}]{1 + \frac{\text{Еабс}}{PV}} - 1 \qquad (4.3)$$

де Еабс – абсолютна ефективність вкладених інвестицій, грн..; PV – теперішня вартість інвестицій PV = ЗВ, грн.; Тж – життєвий цикл наукової розробки, роки.

$$\text{Ев} = 3{,}6$$

Далі, розрахована величина Ев порівнюється з мінімальною (бар'єрною) ставкою дисконтування, що дорівнює:

$$\tau = d + f, \qquad (4.4)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2018 році в Україні d = (0,14...0,2); f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1), але може бути і значно більше.

$$\text{Ев} = 3{,}6 \geq \tau = 0{,}2 + 0{,}1 = 0{,}3.$$

Оскільки величина Ев > τмін, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

4.3 Розрахунок терміну окупності коштів, вкладених в підвищення інформаційної безпеки Docker мереж

Термін окупності вкладених у реалізацію наукового проекту інвестицій Ток можна розрахувати за формулою:

$$\text{Ток} = \frac{1}{\text{Ев}} = \frac{1}{3{,}6} = 0{,}3 \text{ роки}. \qquad (4.5)$$

Оскільки Ток < 3…5-ти років, то фінансування підвищення інформаційної безпеки Docker мереж є доцільним.

ВИСНОВКИ

В першому розділі було виконане порівняння віртуалізації та контейнеризації. В результаті було виявлено, що контейнери займають менше місця на системному диску, але не досконалі в плані безпеки. В свій час віртуальні машини краще підходять для ізольованих систем.

Якщо необхідно запускати максимум додатків на мінімальній кількості серверів, в такому випадку краще скористатися контейнерами. Але пам'ятайте про необхідність додатково подбати про безпеку. Якщо ж потрібно виконувати безліч додатків і / або підтримувати різні ОС - краще підійдуть віртуальні машини. І якщо питання безпеки для стоїть на першому місці, то теж краще залишитися при VM.

Вважаю, більшість з нас будуть використовувати контейнери спільно з віртуальними машинами як в хмарах, так і у власних серверах. Адже можливості економії, що відкриваються нам контейнерами, на масштабі дуже великі, щоб їх ігнорувати. А у віртуальних машин, як і раніше є чимало переваг.

У другому розділі було виконане порівняннь сучасних контейнерних систем, та виявленно, що найпопулярніша контейнерна технологія — Docker. За останні роки її використання значно збільшилось і стало “мейнстрімом” у світі розробки та розгортання додатків. LXD не являється прямим конкурентом через різний підхід та мету цих технологій. А от платформа rkt може конкурувати з Docker, але вона відносно нова та не популярна. Та з часом вона можливо стане досить стабільною та інноваційною, щоб стати на озброєння у розробників та замінити Docker.

При дедальному розгляді Docker у третьому розділі було виявлено, що користувалький інтерфейс містить досить багато команд, а основний файл налаштування dockerfile має багато інструкцій, та їх налаштування вимагає багато зусиль, щодо тонкостей роботи кожної інструкції чи прапорця для команд терміналу. Функціонал досить обширний та дозволяє зручно налаштовувати, відлагоджувати та використовувати всю систему.

В четвертому розділі були продемонстровані кроки зі встановлення, налаштування та запуску контейнерів в системі Docker та роботи з мікрофреймворком Flask. Зокрема:

- Встановлення Flask на Ubuntu
- Модифікація базового Flask додатку
- Встановлення та перший запуск Docker на Ubuntu
- Налаштування Docker для запуску контейнерів Flask додатку.
  - Файл .dockerignore
  - Файл dockerfile

- Створення образу
- Налаштування мережі
- Запуск контейнера з різними налаштуваннями мережі

Матеріал четвертого розділу є досить детальною інструкцією з використання можливостей системи, зі всіма кроками, командами та їх тонкощами, який може бути досить корисним при вивчені Docker. Вобудована таблиця класифікації ризиків в даній роботі. Виявлено, що методи управління термінами дозволяють краще зрозуміти часові рамки виконання проекту, його важливі задачі, які можуть зупинити процес при неправильному розподілені часу та розрахувати ризики та заздалегідь запобігти їх появі.

Список використаних джерел

1. Офіційний сайт Docker. – Режим доступа: https://www.docker.com/. Дата доступа: 12.08.2019.
2. Хабр-Хабр. – Режим доступу: http://habrahabr.ru. - Дата доступа: 10.01.2017.
3. Client-Server Programming and Applications. — Department of Computer Sciences, Purdue University, West Lafayette, IN 47907: Prentice Hall, 1993.
4. Офіційний сайт документації Docker. – Режим доступа: https://docs.docker.com/ - Дата доступа: 08.07.2019.
5. Офіційний Git репозиторій Docker. – Режим доступа: https://github.com/docker/docker/ - Дата доступа: 13.09.2019.
6. Вікіпедія. – Режим доступа: https://uk.wikipedia.org - Дата доступа: 15.10.2019.
7. Записки программиста. Режим доступа: http://eax.me/docker/ - Дата доступа: 12.09.2019.
8. DOU. Режим доступа: https://dou.ua - Дата доступа: 07.09.2019.
9. Хакер ру. Режим доступа: https://xakep.ru - Дата доступа: 05.10.2019.
10. Docker: Creating Structured Containers. - Pethuru Raj et al, 2016.
11. The Docker Book. - James Turnbull, 2016.
12. Docker Cookbook. Solutions and Examples for Building Distributed Applications. - O'Reilly Media, 2015.
13. XGU ru. Режим доступа: http://xgu.ru/ - Дата доступа: 13.08.2019.
14. Ubuntu LXD. Режим доступа: https://www.ubuntu.com/cloud/lxd - Дата доступа: 3.09.2017.
15. Авторские статьи об OpenSource. Режим доступа: http://vasilisc.com/ - Дата доступа: 9.09.2019.
16. Flask. Режим доступа: http://flask.pocoo.org/ - Дата доступа: 5.09.2019.
17. Самойлов А.Г., Самойлов С.А. Адаптивное программирование в цифровых системах телеметрии / Проектирование и технология электронных средств, 2015, No3. - С.3-6.
18.Жоую Пи, Фарук Хан Введение в широкополосные системы связи миллиметрового диапазона / Электроника: наука, технологии, бизнес. - 2012, No3.- С.86-94.
19. Рекомендация МСЭ-R P.2040 Влияние строительных материалов и структур на распространение радиоволн на частотах выше приблизительно 100 МГц.
20. Жоую Пи, Фарук Хан Введение в широкополосные системы связи миллиметрового диапазона / Первая миля. - No6, 2011. - С.10-19.

21. Двухлучевая модель. [Електронний ресурс] / Режим доступа: http://systemseti.com/CCPO/405.html

22. Десятилучевая модель. [Електронний ресурс] / Режим доступа: http://systemseti.com/CCPO/406.html

23. Модель Окамуры. [Електронний ресурс] / Режим доступа: http://systemseti.com/CCPO/408.html

24. Модель Хата. [Електронний ресурс] / Режим доступа: http://systemseti.com/CCPO/409.html

25. Модель Уолфица-Бертони. http://systemseti.com/CCPO/410.html

26. Малютин Н.Д., Лощилов А.Г., Федоров В.Н., Зыков Д.Д. Широкополосные дискретные не дисперсионные фазовращатели на основе эффекта кратного изменения фазовой скорости в многосвязных полосковых структурах с существенно неуравновешенной электромагнитной связью при сохранении согласования в широкой полосе частот / Доклады ТУСУРа, No 4 (34), декабрь 2014. - С. 22-30.

27. Шаров Г.А. Волноводные устройства сантиметровых и миллиметровых волн. - М.: Радиотехника. - 2016. - 638 стр. Рекомендации ITU-R PN.837-7 Характеристики осадков для моделирования распространения. – 2017, No 6.

28. Kurakova T., Valdburger M. How ITU can help develop future networks / ITU News. - 2013, No 1. – p. 38-41.

29. Куракова Т.П. Использование миллиметрового диапазона волн для мобильной связи поколения 5G / Проектирование и технология электронных средств. - 2016, No4. – С. 3-7.

30. Куракова Т.П., Сарьян В.К. Моделирование радиоканалов миллиметрового диапазона частот / Труды НИИР. - 2017, No1. – С.33-39.

31. Куракова Т.П., Сарьян В.К. Методика моделирования радиоканалов миллиметрового диапазона // Материалы XII Международной научнотехнической конференции "Перспективные технологии в средствах передачи информации"-ПТСПИ-2017 в 2-х томах. - г. Суздаль, 5-7 июля 2017.- Т.2.- С.87-90.

32. Куракова Т.П. Доплеровские сдвиги частоты в высокоскоростных каналах мобильной связи // Материалы XII Международной научнотехнической конференции "Перспективные технологии в средствах передачи информации"-ПТСПИ-2017 в 2-х томах. - г. Суздаль, 5-7 июля 2017.- Т.1.- С.47-48.

33. Куракова Т.П., Михайлов С.Н. Моделирование каналов сотовой связи нового поколения // Сб. статей по материалам 1-ой ВНПК

"Инфотелекоммуникации и космические технологии: состояние, проблемы и пути решения". – Курск. - В 2-х частях. – 2017, ч.1. – С.61-66.

34. Куракова Т.П. О моделировании радиоканала миллиметрового диапазона// XXXVI Всероссийская научно-техническая конференция "Проблемы эффективности и безопасности функционирования сложных технических и информационных систем", 28-29 июня 2017 г. - Т.5. - Филиал военной академии РВСН, г. Серпухов. – С.177-180.

35. Куракова Т.П., Самойлов А.Г. О моделировании каналов связи поколения 5G // Международная научно-техническая и научно-методическая конференция «Современные технологии в науке и образовании» СТНО-2017. -Рязань.- Сб. тр., Том 3. - С.102-106.

36. Куракова Т.П., Самойлов А.Г., Самойлов С.А. Разработка имитатора радиоканалов мобильной связи поколения 5G // Четвертая международная конференция Инжиниринг & Телекоммуникации - En&T - 2017, 29-30 ноября, 2017, Москва. – С.

37. Куракова Т.П., Самойлов А.Г., Самойлов С.А., Сарьян В.К. Имитация многолучевых радиоканалов // Международная научно-техническая конференция «Фундаментальные проблемы радиоэлектронного приборостроения» (Intermatic - 2017). - 20 - 24 ноября 2017, Москва. - С.

38. Kurakova T. Overview of the Internet of things // Conference INTHITEN (IN-ternet of THings and ITs ENablers), 3 – 4 June, 2013, St. Petersburg. - p.82-

39. Kurakova T. 5G in simple words / http://www.itu.int/en/ITU-T/studygroups/2017-2020/13/Pages/CB-Future Networks.aspx . - October 2017.

40. Слюсар В.И. Системы MIMO: принципы построения и обработка сигналов. // Электроника: наука, технология, бизнес. – 2005. – № 8. – С. 52 – 58.

46. ДСН 3.3.6.039-99. Державні санітарні норми виробничої та загальної вібрацій.

47. ГОСТ 12.2.032-78. ССБТ. Рабочее место при выполнении работ сидя. Общие эргономические требования.

48. Березюк О. В. Охорона праці. Підсумкова державна атестація спеціалістів, магістрів в галузях електроніки, радіотехніки, радіоелектронних апаратів та зв’язку : навчальний посібник / О. В. Березюк, М. С. Лемешев. – Вінниця : ВНТУ, 2017. – 104 с.

49. Правила улаштування електроустановок. 2-е вид., перероб. і доп. – Х: "Форт", 2009. – 736 с.

50. ДБН В.2.5-27-2006. Захисні заходи електробезпеки в електроустановках будинків і споруд.

Додаток А

(обов'язковий)

ВНТУ

ЗАТВЕРДЖУЮ

Зав.кафедри ТКСТБ ВНТУ,

канд. техн. наук, професор

Г.Г.Бортник

. "__" _____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

ПІДВИЩЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ DOCKER МЕРЕЖ

08-34.МКР.003.00.000 ТЗ

Керівник роботи

к.т.н., доц. кафедри ТКСТБ ВНТУ

Войцеховська О.В.

Виконавець: ст. гр. ТТК-18м

Гринчук В.В.

Вінниця-2019

1 ПІДСТАВА ДЛЯ ВИКОНАННЯ РОБОТИ

Робота проводиться на підставі наказу ректора по Вінницькому національному технічному університету від “02” 10 2019 року № 254 та індивідуального завдання на магістерську кваліфікаційну роботу.

Дата початку роботи: 02.09.2019 р.

Дата закінчення: 09.12.2019 р.

2 МЕТА І ПРИЗНАЧЕННЯ МКР

Метою даної магістерської кваліфікаційної роботи є підвищення інформаційної безпеки Docker мережі шляхом створення програмного забезпечення для перевірки конфігурації серверного програмного забезпечення.

Задачами магістерської кваліфікаційної роботи є:

- розробка технічного завдання;
- аналіз інфраструктури та архітектури технології Docker мережі;
- моделювання ефективності Docker мережі з технологією контейнерізації;
- створення програмного забезпечення для перевірки конфігурації серверного програмного забезпечення.

Об’єкт дослідження є процеси передачі даних в контейнерних мережах.

Предмет дослідження є методи підвищення інформаційної безпеки контейнерних мереж.

Основними завданнями роботи є:

- техніко-економічне обґрунтування доцільності даної розробки;
-
- аналіз інфраструктури та архітектури технології Docker мережі;
- моделювання ефективності Docker мережі з технологією контейнерізації;

- створення програмного забезпечення для перевірки конфігурації серверного програмного забезпечення.

- аналіз економічної ефективності проведеної розробки;

- дослідження питань безпеки життєдіяльності.

Магістерська кваліфікаційна робота створює науково-технічну основу для подальшого розвитку і вдосконалення технології контейнеризації та покращення інформаційної безпеки хмарної інфраструктури.

3 ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ МКР

Робота базується на результатах звіту з переддипломної практики “Підвищення інформаційної безпеки Docker мережі”, який виконувався у ВНТУ 2019/2020 н.р. Під час підготовки магістерської кваліфікаційної роботи будуть використані матеріали цього звіту.

Список використаних джерел розробки:

3.1 Офіційний сайт Docker. – Режим доступа: https://www.docker.com/. Дата доступа: 12.08.2019.
3.2. Хабр-Хабр. – Режим доступу: http://habrahabr.ru. - Дата доступа: 10.01.2017.
3.3. Client-Server Programming and Applications. — Department of Computer
Sciences, Purdue University, West Lafayette, IN 47907: Prentice Hall, 1993.
3.4. Офіційний сайт документації Docker. – Режим доступа: https://docs.docker.com/ - Дата доступа: 08.07.2019.
3.5. Офіційний Git репозиторій Docker. – Режим доступа: https://github.com/docker/docker/ - Дата доступа: 13.09.2019.
3.6. Вікіпедія. – Режим доступа: https://uk.wikipedia.org - Дата доступа: 15.10.2019.
3.7. Записки программиста. Режим доступа: http://eax.me/docker/ - Дата доступа: 12.09.2019.
3.8. DOU. Режим доступа: https://dou.ua - Дата доступа: 07.09.2019.
3.9. Хакер ру. Режим доступа: https://xakep.ru - Дата доступа: 05.10.2019.
3.10. Docker: Creating Structured Containers. - Pethuru Raj et al, 2016.

3.11. The Docker Book. - James Turnbull, 2016.
3.12. Docker Cookbook. Solutions and Examples for Building Distributed Applications. - O'Reilly Media, 2015.
3.13. XGU ru. Режим доступа: http://xgu.ru/ - Дата доступа: 13.08.2019.
3.14 Бортник Г.Г., Васильківський М.В. Методичні вказівки до підготовки магістерських кваліфікаційних робіт для студентів спеціальності «Телекомунікації та радіотехніка» усіх форм навчання.- Вінниця : ВНТУ, 2018.- 50 с.

4 ВИКОНАВЕЦЬ

Вінницький національний технічний університет, кафедра телекомунікаційних систем та телебачення, студент групи ТТК-18м Гринчук В.В.

5 ВИМОГИ ДО ВИКОНАННЯ МКР

Пропонується виконати дослідження методів підвищення інформаційної безпеки контейнерних мереж шляхом розробки програмного забезпечення для перевірки конфігурації серверного програмного забезпечення.

6 ЕТАПИ МКР І ТЕРМІНИ ЇХ ВИКОНАННЯ

№	Назва та зміст етапу	Термін виконання		Очікувані результати	Звітна документ-тація
		початок	закінчення		
1.	Розробка технічного завдання (ТЗ)	02.09.2019р.	06.09.2019р.	Розроблене ТЗ	Додаток А
2.	Аналіз особливостей ахітектури Docker	09.09.2019р.	13.09.2019р.	Проведений аналіз	Вступ. Розділ 1.
3.	Створення математичної моделі для оцінки безпеки Docker мереж	16.09.2019р.	04.10.2019р.	Досліджено метод покращення параметрів та характеристик	Розділ 2
4.	Аналіз спектральної основних вразливостей Docker	07.10.2019р	25.10.2019р.	Частотні та енергетичні характеристики	Розділ 3
5.	Створення програмного запезпечення для покращення інформаційної безпеки Docker мереж	28.10.2019р.	08.11.2019р.	Модифікований алгоритм розподілення частотних ресурсів радіоканалу	Розділ 4
6.	Аналіз економічної ефективності	11.11.2019р.	15.11.2019р.	Економічна частина МКР	Розділ 5
7.	Охорона праці та безпека в надзвичай-них ситуаціях	18.11.2019р.	22.11.2019р.	Частина ОТ та БНС	Розділ 6
8.	Оформлення поясню-вальної записки (ПЗ) та графічної частини	25.11.2019р.	29.11.2019р.	Оформлена документа-ція	ПЗ та графічна частина
9.	Нормоконтроль, попередній захист, рецензування МКР	02.12. 2019р.	06.12.2019р.	Позитивні відзиви	Відзив. рецензія
10.	Захист МКР ЕК		09.12. 2019р.	Позитивний захист	Протокол ЕК

7 ОЧІКУВАНІ РЕЗУЛЬТАТИ ТА ПОРЯДОК РЕАЛІЗАЦІЇ МКР

В результаті виконання роботи будуть розроблені:

- програмне забезпечення для перевірки конфігурацій інформаційної безпеки Docker мережі;
- набір рекомендацій для покращення інформаційної безпеки Docker мережі;
- економічна частина МКР;
- розділ ОП та БНС;
- рекомендації щодо подальшого використання розробленого програмного забезпечення.

Очікуваний техніко-економічний ефект. При впроваджені результатів досліджень очікується зменшення загроз інформаційної безпеки при використанні Docker мереж

8 МАТЕРІАЛИ, ЯКІ ПОДАЮТЬ ПІСЛЯ ЗАКІНЧЕННЯ РОБОТИ ТА ПІД ЧАС ЕТАПІВ

За результатами виконання МКР до ЕК подаються пояснювальна записка, графічна частина МКР, відзив і рецензія.

9 ПОРЯДОК ПРИЙМАННЯ МКР ТА ЇЇ ЕТАПІВ

Поетапно результати виконання МКР розглядаються керівником роботи та обговорюються на засіданні кафедри.

Захист магістерської кваліфікаційної роботи відбувається на відкритому засіданні ЕК.

10 ВИМОГИ ДО РОЗРОБЛЮВАНОЇ ДОКУМЕНТАЦІЇ

Документація, що розробляється в процесі виконання досліджень повинна містити:

- техніко-економічне обґрунтування розробки;
- архітектуру Docker мережі;
- алгоритм оцінки інформаційної безпеки Docker мережі;
- економічну частину та розділ БЖД і ЦЗ;
- рекомендації щодо подальшого використання результатів дослідження.

11 ВИМОГИ ЩОДО ТЕХНІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ З ОБМЕЖЕНИМ ДОСТУПОМ

У зв’язку з тим, що інформація не є конфіденційною, заходи з її технічного захисту не передбачаються.

Додаток Б

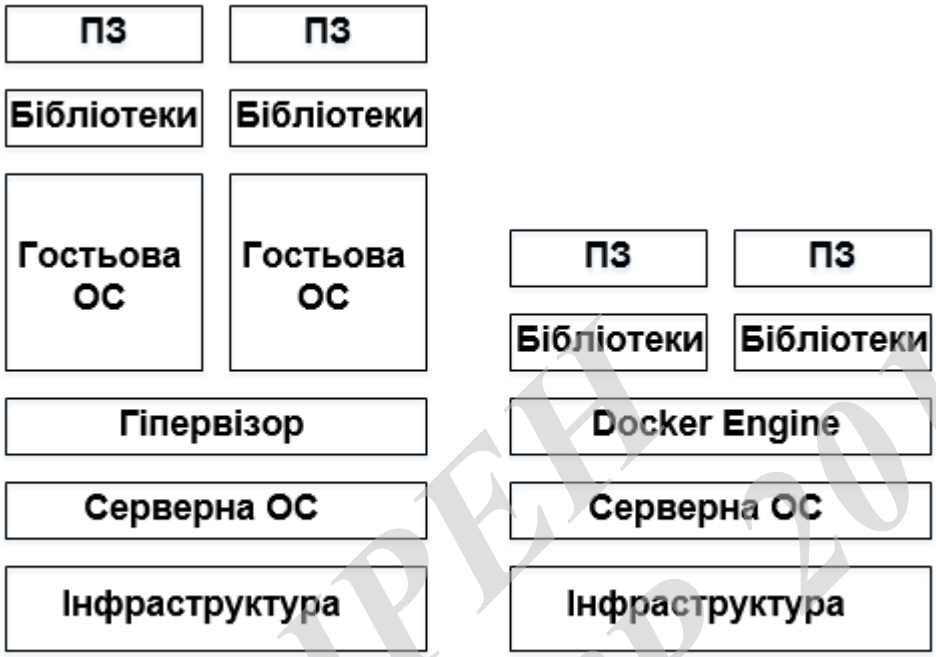
Код програми для покращення інформаційної безпеки Docker мереж

```
ID=$(id -u)
if [ "x$ID" != "x0" ]; then
  echo "Будь ласка, запустіть програму використовуючи root"
fi


if ! docker ps -q >/dev/null 2>&1; then
  printf "Помилка з'єднання з docker\n"
  exit 1

docker_version=$(docker version | grep -i -A2 '^server' | grep ' Version:' \
  | awk '{print $NF; exit}' | tr -d '[:alpha:]-,')
docker_current_version="$(date +%y.%m.0 -d @$(( $(date +%s) - 2592000)))"
do_version_check "$docker_current_version" "$docker_version"
if [ $? -eq 11 ]; then
  echo "Ви використовуєте застарілу версію docker"
else
  echo "Ви використовуєте найновішу версію docker"
fi
docker_users=$(getent group docker)
echo "docker використовується такими користувачами:"
for u in $docker_users; do
  echo "$u \t"
done
  echo "\n"


 if [ "x$DOCKER_CONTENT_TRUST" = "x1" ]; then
   echo "Docker Content Trust увімкнено\n"
 else
   echo "Docker Content Trust вимкнено\n"
 fi

fail=0
no_health_images=""
for img in $images; do
  if docker inspect --format='{{.Config.Healthcheck}}' "$img" 2>/dev/null
  | grep -e "<nil>" >/dev/null 2>&1; then
    if [ $fail -eq 0 ]; then
      fail=1
    fi
    imgName=$(docker inspect --format='{{.RepoTags}}' "$img" 2>/dev/null)
    if ! [ "$imgName" = '[]' ]; then
  no_health_images="$no_health_images $imgName"
    fi
  fi
done
if [ $fail -eq 0 ]; then
  echo "healthcheck зконфігуровано вірно \n"
else
  echo "наявні проблеми в конфігурації healthcheck \n"
fi
```

```
fail=0
add_images=""
for img in $images; do
  if docker history --format "{{ .CreatedBy }}" --no-trunc "$img" | \
    sed '$d' | grep -q 'ADD'; then
    if [ $fail -eq 0 ]; then
      fail=1

    fi
    imgName=$(docker inspect --format='{{.RepoTags}}' "$img" 2>/dev/null)
    if ! [ "$imgName" = '[]' ]; then
      add_images="$add_images $imgName"
    fi
  fi
done
if [ $fail -eq 1 ]; then
  resulttestjson "Спробуйте використати команду COPY замість ADD \n"
fi


total_containers=$(docker info 2>/dev/null | grep "Containers" | awk '{print $2}')
running_containers=$(docker ps -q | wc -l | awk '{print $1}')
diff="$((total_containers - running_containers))"
if [ "$diff" -gt 25 ]; then
  echo "В системі присутньо $total_containers контейнерів,
  з яких лише $running_containers є активними \n"
else
  echo "В системі присутньо $total_containers контейнерів,
  з них $running_containers активні \n"
fi

if [ grep -i 'tcp://' "$CONFIG_FILE" 2>/dev/null 1>&2 ] || \
  [ $(get_docker_cumulative_command_line_args '-H' | grep -vE '(unix|fd)://') >/dev/null 2>&1 ];
  then
  if [ $(get_docker_configuration_file_args '"tlsverify":' | grep 'true') ] || \
      [ $(get_docker_cumulative_command_line_args '--tlsverify' | grep 'tlsverify') >/dev/null 2>&1 ];
      then

    echo "TLS зконфігуровано вірно \n"

  elif [ $(get_docker_configuration_file_args '"tls":' | grep 'true') ] || \
      [ $(get_docker_cumulative_command_line_args '--tls' | grep 'tls$') >/dev/null 2>&1 ];
      then
    echo "Docker daemon працює з TLS через TCP, але не отримав верифікації \n"
  else
    echo "Docker daemon працює через TCP без використання TLS \n"
  fi
else
  echo "Docker daemon не працює через TCP \n"
fi

images=$(docker images -q | sort -u | wc -l | awk '{print $1}')
active_images=0

for c in $(docker inspect --format "{{.Image}}" $(docker ps -qa) 2>/dev/null); do
  if docker images --no-trunc -a | grep "$c" > /dev/null ; then
    active_images=$(( active_images += 1 ))
  fi
done

  echo   "В системі присутньо: $images образів \n"

if [ "$active_images" -lt "$((images / 2))" ]; then
  echo   " Лише $active_images з $images використовується \n"
fi
```

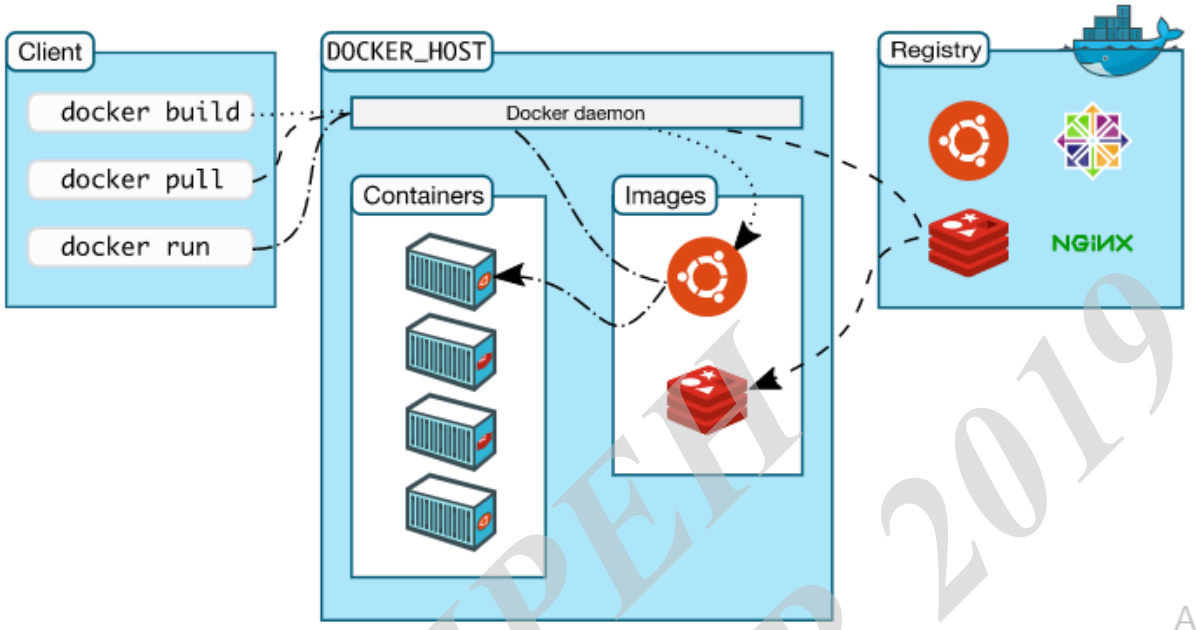
Додаток В

Схема для порівняння технологій контейнеризації та віртуалізації



Додаток Г

Архітектура технології Docker

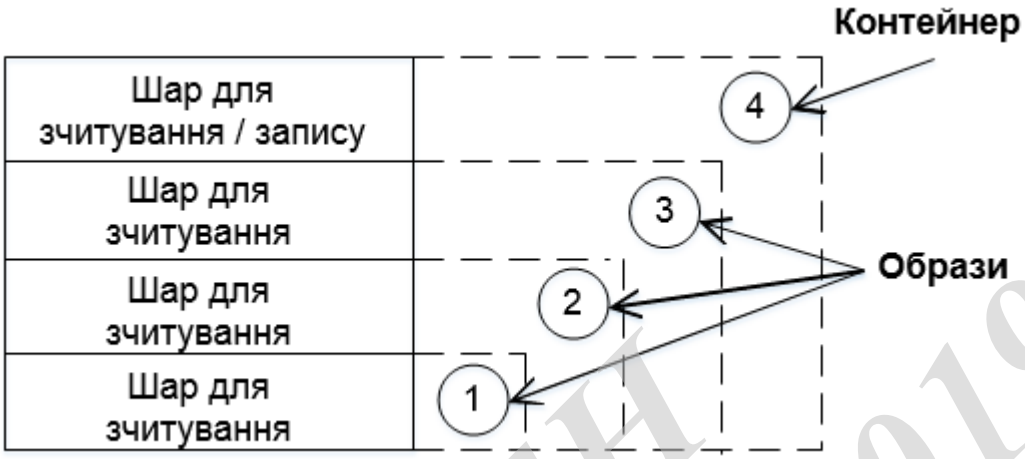


Додаток Д

Рівнева структура Docker образу

Додаток Е

Структура шарів Doker контейнера



Додаток Ж

Граф станів вузла елементу Docker мережі