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

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Системного проектування
(повна назва кафедри)

«На правах рукопису»
УДК 004:004.453

«До захисту допущено»
Завідувач кафедри
__________ А.І.Петренко
(підпис) (ініціали, прізвище)
"___"____________20__ р.

Магістерська дисертація

зі спеціальності (спеціалізації) 122 - комп'ютерні науки та інформаційні технології (Системне проектування сервісів)
(код і назва спеціальності)

на тему: Дослідження технології контейнеризації у системах керування потоками даних

Виконав (-ла): студент (-ка) 6 курсу, групи ДА-62м
(шифр групи)

Загороднюк Андрій Олександрович __________
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник к.т.н., доц., Харченко К.В.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Розробка стартап-проекту к.т.н., доц., Харченко К.В.
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент д.т.н., професор, Тарасенко В.П. __________
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль к.т.н., доцент, с.н.с. Кисельов Г.Д. __________
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент ____________
(підпис)

Київ – 2018 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Інститут/факультет Інститут прикладного системного аналіз
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною (освітньо-науковою) програмою

Спеціальність 122 - комп’ютерні науки та інформаційні технології (Системне проектування сервісів)
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
__________ А.І. Петренко
(підпис) (ініціали, прізвище)

«___»_____________20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

____
(прізвище, ім’я, по батькові)

1. Тема дисертації Дослідження технології контейнеризації у системах керування потоками даних

науковий керівник дисертації Харченко К.В. к.т.н., доц.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27»03 2018 р. №1028-с

2. Строк подання студентом дисертації ______________________________

3. Об’єкт дослідження технологія контейнеризації.
4. Предмет дослідження алгоритми багаторівневого навчання для побудови класифікаційних моделей

5. Перелік завдань, які потрібно розробити 1. Аналіз концепції віртуалізації та контейнеризації. 2. Встановлення та налаштування інфраструктури на базі платформи Docker і Kubernetes. 3. Застосування Docker і Kubernetes для управління потоками даних. 4. Розробка стартап-проекту. 5. Зробити висновки щодо отриманих результатів.

6. Орієнтовний перелік публікацій

1. Загороднюк А.О. “Контейнеризація” - Молодой исследователь: вызовы и перспективы: сб. ст. по материалам LXIX междунар. науч.-практ. конф. — № 16(69). — М., Изд. «Интернаука», 2018. – 482 с.

2. Загороднюк А.О. “Архитектура Docker” - Молодой исследователь: вызовы и перспективы: сб. ст. по материалам LXIX междунар. науч.-практ. конф. — № 16(69). — М., Изд. «Интернаука», 2018. – 482 с.

7. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

8. Дата видачі завдання 01.02.2018

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.02.2018	
2	Збір інформації та аналіз літератури	15.02.2018	
3	Проведення огляду теоретичного базису контейнеризації	28.02.2018	
4	Дослідження архітектури Docker.	11.03.2018	
5	Огляд програмних інструментів для керування потоками даних	13.04.2018	
6	Дослідження можливості використання Docker та Kubernetes в IoT	25.04.2018	
7	Оформлення дипломної роботи	30.04.2018	
8	Отримання допуску до захисту та подача роботи в ДЕК	09.05.2018	

Студент ____________ ____Загороднюк А.О._
(підпис) (ініціали, прізвище)

Науковий керівник дисертації ____________ ____Харченко К.В.___
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника

РЕФЕРАТ

магістерської дисертації Загороднюка Андрія Олександровича на тему:
«Дослідження технології контейнеризації у системах керування потоками даних»

Дана магістерська дисертація присвячена дослідженню технології контейнеризації у системах керування потоками даних. Метою роботи є дослідження існуючого програмного забезпечення контейнеризації і його використання у системах керування потоками даних.

В роботі розглянуто існуючі способи контейнеризації та програмне забезпечення для керуванням контейнерами. Була досліджена можливість використання знайдених рішень в IoT сфері, а також була спроектована стратегія стартапу, яка описала бізнес-модель продукту, який спроможний зайняти перші позиції.

Актуальною науково-прикладною проблемою є знаходження рішення для керування потоками даних використовуючи контейнеризацію.

Об’єкт дослідження – технологія контейнеризації.

Предмет дослідження – система керування потоками даних завдяки технології контейнеризації.

Мета і задачі дослідження. Метою роботи є розгортання мікросервісів за допомогою контейнерів, автоматичне керування контейнерами, а також розгортання їх на IoT-девайсі.

Загальний обсяг роботи: 84 сторінки, 37 рисунків, 23 таблиць, 2 додаток на 2 стр., 15 посилань.

Ключові слова: КОНТЕЙНЕРИЗАЦІЯ, МІКРОСЕРВІС, DOCKER, KUBERNETES, IOT.

РЕФЕРАТ

магистерской диссертации Загороднюка Андрея Александровича на тему: «Исследование технологии контейнеризации в системах управления потоками данных»

Данная магистерская диссертация посвящена исследованию технологии контейнеризации в системах управления потоками данных. Целью работы является исследование существующего программного обеспечения контейнеризации и использования в системах управления потоками данных.

В работе рассмотрены существующие способы контейнеризации и программное обеспечение для управлениея контейнерами. Была исследована возможность использования найденных решений в IoT сфере, а также была спроектирована стратегия стартапа, которая описала бизнес-модель продукта, который способен занять первые позиции.

Актуальностью научно-прикладной проблемой является нахождение решения для управления потоками данных используя контейнеризацию.

Объект исследования - технология контейнеризации.

Предмет исследования - система управления потоками данных благодаря технологии контейнеризации.

Цель и задачи исследования. Целью работы является развертывание микросервисов с помощью контейнеров, автоматическое управление контейнерами, а также развертывания их на IoT-девайсе.

Общий объем работы: 84 страницы, 37 рисунков, 23 таблиц, 2 приложение на 2 стр., 15 ссылок

Ключевые слова: КОНТЕЙНЕРИЗАЦИЯ, МИКРОСЕРВИС, DOCKER, KUBERNETES, IOT.

ABSTRACT

for Master's thesis of Zahorodniuk Andrii

On «Investigation of the technology of containerization in data flow control systems»

This master's dissertation is devoted to research of containerization technology in data flow management systems. The purpose of the work is to study existing containerization software and its use in data flow management systems.

The paper considers existing containerization and container management software. The possibility of using the solutions found in the IoT field was explored, and a startup strategy was developed that described the business model of a product that was able to take the first position.

The actual scientific and applied problem is finding a solution for managing data flows using containerization.

The object of research is the technology of containerization.

The subject of research is a data flow management system due to the technology of containerization.

The purpose and tasks of the study. The purpose of the work is to deploy microservices through containers, automatic container management, and also deploy them on the IoT-device.

The total amount of work: 84 pages, 37 figures, 23 tables, 2 appendix for 2p., 15 references.

Keywords: CONTAINERIZATION, MICROSERVICE, DOCKER, KUBERNETES, IOT

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 9

ВСТУП 10

1 РОЗГЛЯД ЗАДАЧІ ВІРТУАЛІЗАЦІЇ ТА КОНТЕЙНЕРИЗАЦІЇ 12

1.1 Задачі дипломної роботи 12

1.2 Ціль дипломної роботи 12

1.3 Історія виникнення 14

1.4 Використання віртуалізації 16

1.4.1 Накладні витрати 17

1.4.2 Гнучкість 17

1.4.3 Зберігання 18

1.5 Основа віртуалізації 18

1.5.1 Технологія Xen 19

1.5.2 Контейнери LXC 20

1.5.3 Сучасний Docker 21

1.6 Переваги використання віртуалізації 24

1.7 Висновки 26

2 НАЛАШТУВАННЯ ТА ЗАПУСК DOCKER ТА KUBERNETES 27

2.1 Використання Docker 27

2.2 Головні компоненти Docker 29

2.2.1 Архітектура Docker 29

2.2.2 Робота зображення 31

2.2.3 Робота Docker реєстра 31

2.2.4 Робота контейнерів 31

2.3 Встановлення Docker 32

2.3.1 Відносини з Docker Machine 32

2.3.2 Системні вимоги 32

2.3.3 Завантаження Docker 33

2.4 Тестове налаштування Docker 36

2.4.1 Використання Dockerfile 36

2.5 Загальні налаштування Docker 39

2.5.1 Основні 39

2.5.2 Доступ до файлів 41

2.5.3 Розширені налаштування 42

2.5.4 Диск ... 43

2.6 Огляд Kubernetes ... 44

2.7 Висновки ... 46

3 КЕРУВАННЯ ПОТОКАМИ ДАНИХ У СЕРЕДІ IOT ЗА ДОПОМГОЮ DOCKER ТА KUBERNETES ... 47

3.1 Dataflow managements systems ... 47

3.2 Розробка мікросервісів з використанням NodeJS ... 47

3.3 Контейнеризація мікросервісу ... 49

3.4 Налаштування Kubernetes для мікросервісу ... 51

3.5 Розробка прикладу для dataflow ... 54

3.6 Концепція “Один мікросервіс – один контейнер” ... 56

3.7 Використання контейнеризації в IoT ... 59

3.8 Висновки ... 62

4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ ... 64

4.1 Опис ідеї проекту ... 64

4.2 Технологічний аудит ідеї проекту ... 65

4.3 Аналіз ринкових можливостей запуску стартап-проекту ... 66

4.4 Розроблення ринкової стратегії проекту ... 75

4.5 Розробка маркетингової програми ... 77

4.6 Висновки ... 81

ВИСНОВКИ ... 83

ПЕРЕЛІК ПОСИЛАНЬ ... 85

ДОДАТОК А ... 86

ДОДАТОК Б ... 87

ПЕРЕЛІК СКОРОЧЕНЬ

IoT – інтернет речей

IT– інформаційні технології

ОС – операційна система

ПП – програмний продукт

ПЗ – програмне забезпечення

Docker - інструментарій для управління ізольованими Linux-контейнерами.

ОС – операційна система

vendor-lock-in – бізнес модель, в якій встановлюється залежність користувача від продукту однієї компанії

ПЗ – програмне забезпечення

CPU – центральний процесор

Машина – Електро́нна обчи́слювальна маши́на

VM (віртуальна машина) - модель обчислювальної машини, створеної шляхом віртуалізації обчислювальних ресурсів: процесора, оперативної пам'яті, пристроїв зберігання та вводу і виводу інформації.

Гіпервізор або Монітор віртуальних машин — комп'ютерна програма або обладнання, що забезпечує одночасне, паралельне виконання декількох операційних систем на одному і тому ж комп'ютері

Daemon (демон) – клієнт докер, що доступен по REST API.

Дистрибутив — форма розповсюдження програмного забезпечення. Дистрибутив зазвичай містить програми для початкової ініціалізації системи.

ВСТУП

На шляху просування Вашого додатку через цикл розробки найчастіше зустрічається безліч перешкод. Крім роботи з підготовки програми до роботи в різних умовах, Ви також можете зіткнутися з проблемами відстеження залежностей, масштабування додатки і оновлень окремих компонентів, що не зачіпають безпосередньо сам додаток.

Docker-контейнеризація і сервіс-орієнтоване проектування намагаються вирішити багато з цих проблем. Додатки можуть бути розбиті на керовані функціональні компоненти, індивідуально упаковані разом з усіма своїми залежностями, а потім легко розгорнуті до нестандартної архітектури. Це також спрощує масштабування і оновлення компонентів.

Ми обговоримо переваги контейнеризації і те, як Docker допомагає вирішити багато зі згаданих вище проблем. Docker - ключовий компонент для розподіленого розгортання контейнерів, що надає можливість легкого масштабування і управління.

Контейнеризація і ізолювання компонентів не нові концепції в світі обчислень. Деякі Unix-подібні операційні системи використовують "зрілі" лінійні технології вже більше 10 років.

Система LXC (Linux Containers) - основа наступних технологій контейнеризації - була додана в ядро Linux в 2008 році. LXC використовує комбінацію таких функцій ядра, як cgroups (дозволяє ізолювати і здійснювати контроль за використанням ресурсів) і простору імен (дозволяють розділяти групи так, щоб вони не могли "бачити" один одного), для реалізації легковагій ізоляції процесів.

Docker, що з'явився трохи пізніше, позиціонувався, як інструмент для спрощення роботи зі створення та управління контейнерами. Спочатку Docker використовував LXC як драйвер виконання за замовчуванням (з тих пір для цих цілей була розроблена бібліотека під назвою libcontainer). Docker, що не приносячи великої кількості нових ідей, зробив контейнери доступними для звичайного розробника і системного адміністратора шляхом спрощення процесу і стандартизації інтерфейсу.

Це стимулювало відродження інтересу до контейнеризації серед розробників в Linux-світі.

При проектуванні додатків, які будуть розгортатися в контейнерах, одне з перших питань, що виникають - це архітектура програми. Зазвичай, контейнеризовані додатки найкраще працюють при сервіс-орієнтованій архітектурі.

Сервіс-орієнтовані додатки розбивають функціональність системи на окремі компоненти, які взаємодіють між собою через чітко визначені інтерфейси. Сама контейнерна технологія заохочує такий тип проектування, тому що він дозволяє незалежно масштабувати або оновлювати кожен компонент.

Додатки, що реалізують цей підхід до проектування, повинні мати такі характеристики:

- Вони не повинні покладатися на особливості хост-системи.
- Кожен компонент повинен надавати сумісний API, який користувачі можуть використовувати як доступ до сервісу.
- Кожен сервіс повинен брати до уваги змінні оточення в процесі початкового налаштування.
- Дані програми повинні зберігатися поза контейнера на примонтованих томах або в окремих контейнерах з даними.

Ці стратегії дозволяють замінювати або змінювати кожен компонент незалежно (до тих пір, поки підтримується API). Вони також фокусуються на горизонтальній масштабованості, з огляду на те, що кожен компонент може бути масштабований.

1 РОЗГЛЯД ЗАДАЧІ ВІРТУАЛІЗАЦІЇ ТА КОНТЕЙНЕРИЗАЦІЇ

1.1 Задачі дипломної роботи

Задачами дипломної роботи є:

1. Аналіз концепції віртуалізації та контейнеризації.
2. Інсталяція та налаштування інфраструктури на базі платформи Docker та Kubernetes.
3. Застосування Docker та Kubernetes для управління потоками даних. Дослідити можливість використання у середі IoT.
4. Розроблення стартап-проекту.
5. Зробити висновки щодо отриманих результатів.

1.2 Ціль дипломної роботи

Основною ціллю дипломної роботи є дослідження технології контейнеризації у системах керування потоками даних.

Віртуалізація на рівні операційної системи — “метод віртуалізації, при якому ядро операційної системи підтримує декілька ізольованих примірників простору користувача, замість одного. Ці примірники (часто звані контейнерами або зонами) з точки зору користувача повністю ідентичні реальному серверові. Ядро забезпечує повну ізольованість контейнерів, тому програми з різних контейнерів не можуть впливати одна на одну. Для систем на базі UNIX ця технологія може розглядатися як поліпшена реалізація механізму chroot.”[1]

Docker — “інструментарій для управління ізольованими Linux-контейнерами. Docker доповнює інструментарій LXC більш високорівневим API, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і

потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів.”[2]

У звичайних операційних системах для персональних комп'ютерів комп'ютерна програма може бачити всі ресурси системи (навіть якщо вона може не мати доступу).

Вони включають:

- Можливості апаратного забезпечення, такі як процесор та мережеве з'єднання
- Дані, які можна читати чи писати, наприклад файли, папки та спільні ресурси мережі
- З'єднані периферійні пристрої, які можна взаємодіяти, такі як веб-камера, принтер, сканер або факс

Операційна система може мати можливість дозволити або заборонити доступ до таких ресурсів, на підставі яких програма запитує їх та обліковий запис користувача, в контексті якої він працює. Операційна система може також приховувати ці ресурси, тому, коли комп'ютерна програма перераховує їх, вони не відображаються в результатах переліку. Тим не менше, з точки зору програмування комп'ютерна програма взаємодіє з цими ресурсами, а операційна система керувала акт взаємодії.

При операційній системі-віртуалізації або контейнерах можна запускати програми в контейнерах, до яких виділяються лише частини цих ресурсів. Програма, яка очікує бачити весь комп'ютер, коли він запускається всередині контейнера, може бачити лише виділені ресурси і вважає їх доступними. У кожній операційній системі можна створити декілька контейнерів, до кожного з яких виділяється підмножина ресурсів комп'ютера. Кожен контейнер може містити будь-яку кількість комп'ютерних програм. Ці програми можуть працювати одночасно або окремо, навіть взаємодіяти один з одним.

Контейнеризація має подібність до віртуалізації додатків: у останній лише одна комп'ютерна програма розміщується в ізольованому контейнері, і ізоляція застосовується лише до файлової системи.

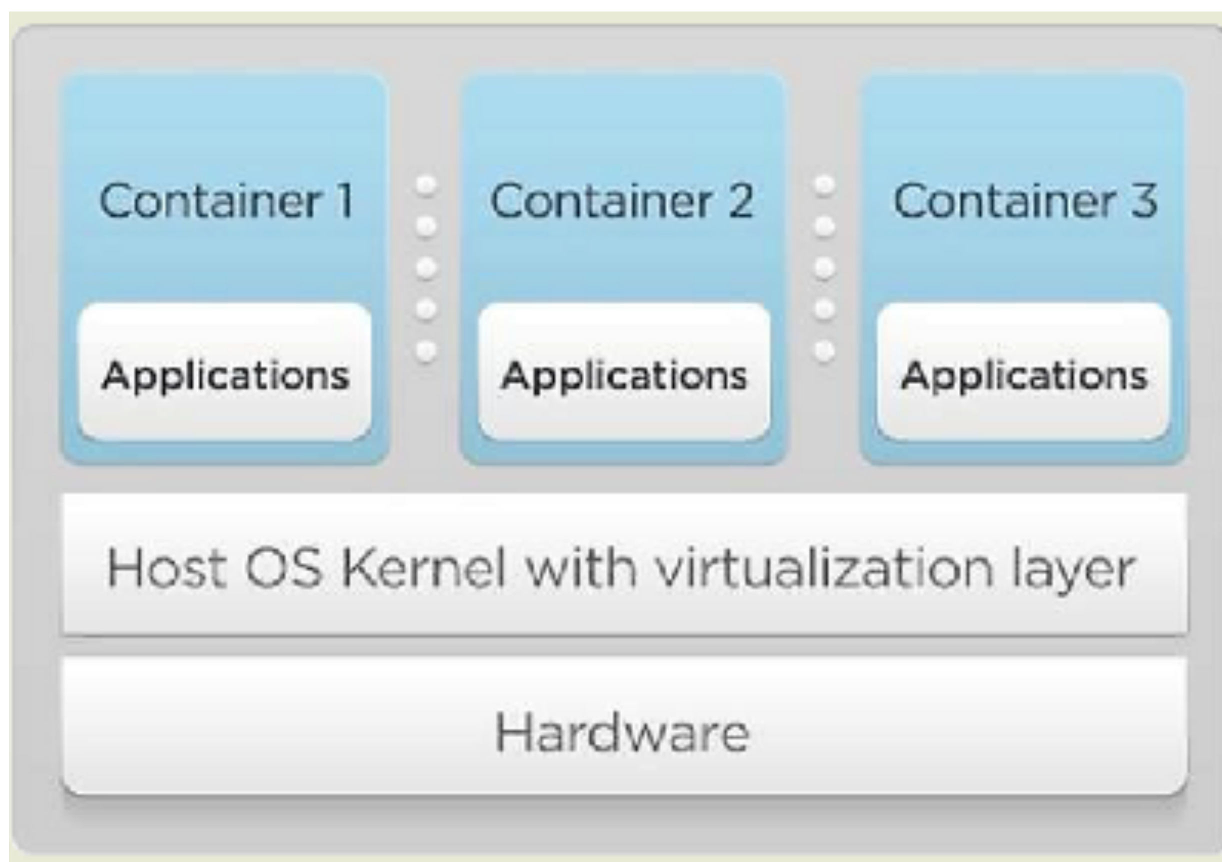

Рисунок 1.1 – Віртуалізація на рівні операційної системи

1.3 Історія виникнення

У середині 1960-х рр. Дослідницький центр IBM Watson був домом для проекту M44 / 44X, метою якого було оцінити концепції, що склалися, у часі. Архітектура була заснована на віртуальних машинах: основна машина була IBM 7044 (M44), і кожна віртуальна машина була експериментальним зображенням основної машини (44X). Адресний простір 44X був резидентним в ієрархії пам'яті M44, реалізований через віртуальну пам'ять і мультипрограмування.

IBM надав комп'ютер IBM 704, серію оновлень (наприклад, до 709, 7090 та 7094), а також доступ до деяких його системних інженерів до MIT у 1950-х роках. Саме на комп'ютерах IBM була розроблена система сумісності спільного доступу до часу (CTSS) на MIT. Наглядова програма CTSS обробляла введення / виводу консолі, планування роботи переднього та заднього фонового режиму (офлайн-ініціативи), тимчасове зберігання та відновлення програм під час запланованого обміну, монітор дискового вводу / виводу тощо. Орган нагляду мав безпосередній контроль над всіма пастками переривання.

Приблизно в той же час, IBM створила 360 комп'ютерів. Проект MAC MIT, заснований восени 1963 р., Був великою та добре профінансованою організацією, яка пізніше перетворилася в лабораторію комп'ютерних наук MIT. Цілі проекту MAC включали розробку та впровадження кращої системи розподілу часу на основі ідей від CTSS. Це дослідження привело б до Multics, хоча IBM втратить пропозицію і GE Electric 645 буде використовуватися замість нього.

Незалежно від цієї "втрати", IBM була, мабуть, найважливішою силою в цій галузі. Було розроблено декілька систем віртуальних машин на базі IBM: CP-40 (розроблений для модифікованої версії IBM 360/40), CP-67 (розроблений для IBM 360/67), відомі VM / 370 і набагато більше. Як правило, віртуальні машини IBM були ідентичними "копіями" базового апаратного забезпечення. Компонент, який називається монітором віртуальної машини (VMM), запускається безпосередньо на "реальному" апаратному забезпеченні. Потім через VMM можна створити декілька віртуальних машин, і кожен примірник може запустити власну операційну систему. IBM VM пропозиції сьогодні - це дуже шановані та надійні обчислювальні платформи.

Роберт П. Голдберг описує тодішнє стан речей у своєму документі 1974 р. Під назвою Обстеження досліджень віртуальних машин. Він каже: "Системи віртуальних машин спочатку були розроблені для виправлення деяких недоліків типових архітектур третіх поколінь та багатопрограмних операційних систем - наприклад, OS / 360". Він зазначив, що в таких системах існує подвійна державна апаратна організація - привілейований та непривілейований режим, який теж поширюється і

сьогодні. У привілейованому режимі всі інструкції доступні для програмного забезпечення, тоді як у непривілейованому режимі вони не є доступними. ОС надає невелику програму резидентів, яка називається привілейованим ядром програмного забезпечення (аналогічним ядру). Користувацькі програми можуть виконувати непривілейовані апаратні інструкції або здійснювати наглядові дзвінки - наприклад, SVC - (аналогічно системним викликам) до привілейованого ядра програмного забезпечення, щоб мати привілейовані функції, наприклад, введення / виводу - від їх імені. Хоча це добре працює для багатьох цілей, існують суттєві недоліки підходу. Розглянемо кілька:

- Відкрито лише один "голий інтерфейс машини". Таким чином, можна запустити лише одне ядро. Неможливо запустити щось, незалежно від того, чи це інше ядро (що належить до тієї ж чи іншої операційної системи), чи довільна програма, яка вимагає спілкування з голом комп'ютером (наприклад, тестування, налагодження або діагностична програма низького рівня). поряд з завантаженим ядром.
- Не можна виконувати жодної діяльності, яка б порушувала працюючу систему (наприклад, оновлення, міграцію, налагодження системи тощо). Також неможливо безпечно запустити ненадійні програми.
- Неможливо легко забезпечити ілюзію апаратної конфігурації, якої не має (декілька процесорів, довільну пам'ять і конфігурації зберігання тощо) до деякого програмного забезпечення.

1.4 Використання віртуалізації

Віртуалізація на рівні операційної системи зазвичай використовується в середовищах віртуального хостингу, де це корисно для надійного розподілу ресурсів кінцевого обладнання серед великої кількості взаємно недовіряючих користувачів. Системні адміністратори також можуть використовувати його для консолідації серверного обладнання, перенесення послуг на окремі хоста в контейнери на одному сервері.

Інші типові сценарії включають розділення декількох програм на окремі контейнери для підвищення безпеки, незалежність апаратних засобів та додаткові функції керування ресурсами. Покращена безпека, що забезпечується використанням chroot-механізму, однак, недалеко від залізної плитки. Необхідна цитата. Реалізації віртуалізації операційної системи на рівні, здатної жити міграцією, також можна використовувати для динамічного навантаження контейнерів між вузлами в кластері.

1.4.1 Накладні витрати

Віртуалізація на рівні операційної системи зазвичай нав'язує менші накладні витрати, оскільки програми в віртуальних розділах використовують звичайний інтерфейс системного виклику операційної системи і не потрібно піддавати емуляції або бути запущені на проміжній віртуальній машині, як це відбувається при повній віртуалізації (наприклад як VMware ESXi, QEMU або Hyper-V) та паравіртуалізація (наприклад, Xen або UML). Така форма віртуалізації також не вимагає підтримки в апаратній частині, щоб ефективно працювати.

1.4.2 Гнучкість

Віртуалізація на рівні операційної системи не настільки ж гнучка, як і інші підходи до віртуалізації, оскільки вона не може приймати гостьову операційну систему, на відмінну від хоста, або іншого ядра гостя.

Solaris частково подолає описане вище обмеження з функцією фірмових зон, яка забезпечує можливість запуску середовища в контейнері, який імітує більш стару версія Solaris 8 або 9 у хості Solaris 10. Ліцензовані зони Linux (називаються "фірмовими зонами" lx) також доступні на системах Solaris на основі x86, що забезпечують повне користувальницький простір Linux та підтримку виконання додатків Linux; Крім того, Solaris надає утиліти, необхідні для встановлення дистрибутивів Red Hat Enterprise Linux 3.x або CentOS 3.x всередині "lx" зон. Проте в 2010 році з Solaris вилучено фірмові зони Linux; в 2014 році вони були знову введені в Illamus, що є відкритим вихідним кодом Solaris fork, що підтримує 32-розрядні ядра Linux.

1.4.3 Зберігання

Деякі реалізації віртуалізації на рівні операційної системи забезпечують механізми copy-on-write (CoW) на рівні файлів. (Найчастіше стандартна файлова система розподіляється між розділами, а ті розділи, які змінюють файли, автоматично створюють власні копії). Це легше для створення резервної копії, більш просторого та простішого для кешування, ніж копіювання на рівні блоку -писати схеми, загальні для віртуалізаторів цілісної системи. Однак віртуалізатори цілісної системи можуть працювати з нерідними файловими системами, а також створювати і відкачати моментальні знімки всього стану системи. Дослідження принципу та вимог розподілення обчислень.

1.5 Основа віртуалізації

Сучасні технології та методологія інфраструктури додатків стимулюють прискорене впровадження технологій хмарних обчислень, а також різних технологій віртуалізації. Наприклад, методологія розробки програмного забезпечення DevOps, яка потребує інфраструктури для сценаріїв. Виявлені технології віртуалізації зосереджені переважно на ядрі Linux і можуть бути розділені на три категорії: повна віртуалізація, пара-віртуалізація та віртуалізація на базі контейнерів. Пара-віртуалізація трохи покращує ядро віртуальних машин, щоб оптимізувати продуктивність у віртуальному середовищі. Повна віртуалізація не потребує налаштування ядра. Контейнерна віртуалізація взагалі не використовує ядро. Технології, що використовуються в рішеннях IaaS (інфраструктура як сервіс) та рішення PaaS (платформа як сервіс), такі як Amazone Elastic Compute Cloud, Google App Engine, DotCloud та Open Shift. Оскільки інфраструктура додатків може бути різноманітною, немає єдиного найкращого рішення для всіх цих служб. Швидше за все, кожна служба або додаток мають свої специфічні вимоги. Ми будемо порівнювати дві різні технології з різними архітектурами: Xen, гіпервізор пара-віртуалізації та LXC, конфігурація ізоляції ядра Linux. У цьому розділі ми коротко

пояснимо обидві архітектури. У наступних розділах ми детально розглянемо їх ефективність та оперативну гнучкість.

1.5.1 Технологія Xen

Xen заснований на пара-віртуалізації. Віртуальна машина на гіпервізоні Xen могла б запустити модифіковане ядро, щоб забезпечити кращу продуктивність і зменшити накладні витрати. Гіпервізор встановлюється безпосередньо в завантажувач. Віртуальні машини, що працюють на вершині гіпервізора, називаються доменами або гостями. Спеціальний домен, званий domain0, керує системою (Dom0). Цей домен має можливість налаштування середовища. Вона може містити інструменти для налаштування мережі, надання нових віртуальних машин і їх міграції. Інші домени - те, що називається неприйнятним домену. Тому їх називають DomU. Ці домени DomU можуть бути пара-віртуалізованими (PV) або hardware-assisted (HVM). PV-домени вимагають оптимізованого ядра, в той час як HVM-домени не вимагають ядра МОДИФІКАЦІЇ але потрібна підтримка віртуалізації x86 в (Intel VT-X, AMD-SVM). Ця підтримка архітектури не потрібна під час запуску ПВ віртуальної машини. Оскільки Xen лише забезпечує технологію гіпервізора, нам все ще потрібна операційна система управління, яка буде встановлена на Dom0. XenServer - це реалізація для системи управління Dom0. Він надає розширені інструменти для забезпечення, управління, моніторингу та перенесення віртуальних машин. Це домен, на якому можна встановити XenServer. Якщо domain0 є віртуальним середовищем, установка XenServer сама запускається на віртуальній машині. Рисунок 1.2 показує схематичний огляд машини під управлінням гіпервізором Xen з XenServer, встановлений на Dom0. Xen розробляється вже більше 12 років і тому може вважатися зрілою технікою. Xen технологія, що широко використовується, наприклад, Amazone Web Services, Google, Rackspace, Oracle, Cisco і Citrix.

1.5.2 Контейнери LXC

Контейнери Linux (LXC) забезпечують легку віртуалізацію операційної системи і є відносно новими для інших технологій, перелічених у таблиці 1. На відміну від Xen, LXC не вимагає підтримки архітектури. LXC є наступником VServer і OpenVZ, інших контейнерних технологій віртуалізації. Основний принцип контейнера полягає в тому, що він дозволяє виділяти процеси та їх ресурси без будь-якої емуляції устаткування чи апаратних вимог. Контейнери є своєрідною платформою віртуалізації, де кожен контейнер може запускати власну операційну систему, але розділяти ядро. Тому кожний контейнер має свою власну файлову систему та мережевий стек, і кожен контейнер може запустити свій власний дистрибутив Linux. Наприклад, хост CoreOS може запускати Ubuntu, RHEL, Debian, Arch і навіть інші контейнери CoreOS одночасно. Ці абстракції роблять контейнер поводженням як віртуальна машина з окремою файловою системою, мережевими та іншими ресурсами операційної системи. Але насправді вони не є, тому що немає ніякої апаратної емуляції.

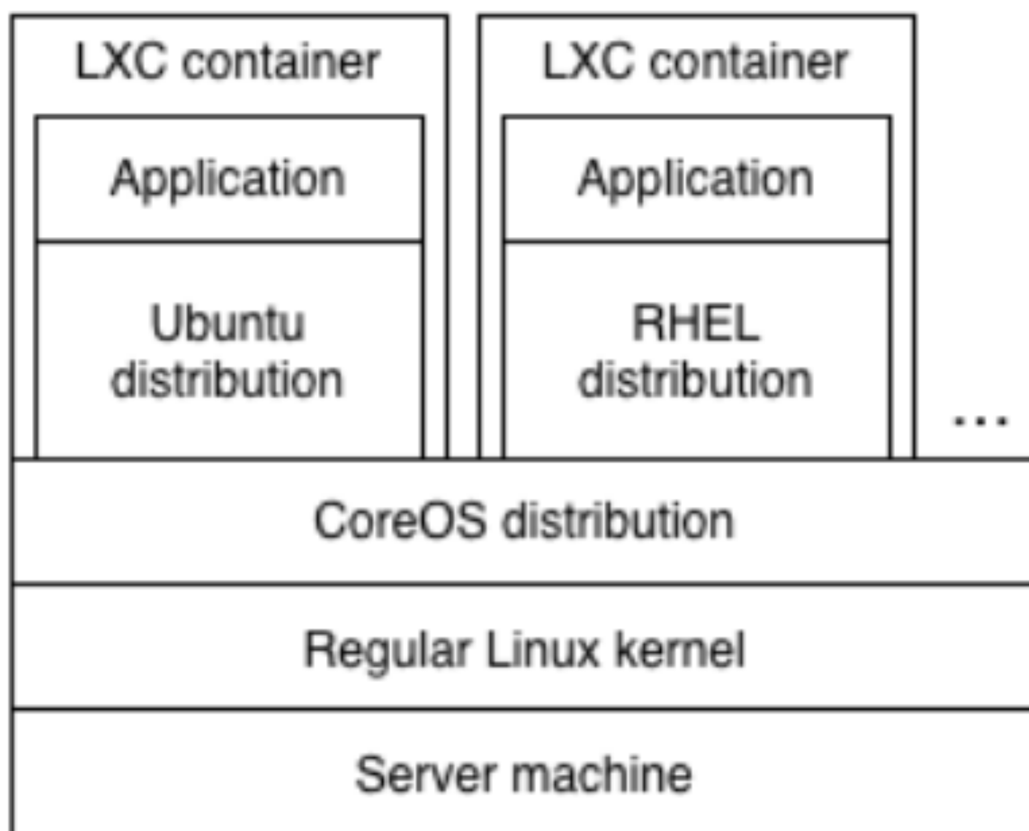

Рисунок 1.2 – Схема роботи машини за CoreOS та LXC

На рисунку 1.2 показаний схематичний огляд машини, що працює з контейнерами CoreOS та двома контейнерами LXC. Ізоляція є важливим аспектом контейнерів, вона забезпечується через Linux груп і просторів імен. Простір імен використовуються для виділення таких ресурсів, як: файлова система, мережа, управління користувачами та ідентифікатори процесу. Cgroups використовуються для розподілу та управління ресурсами. Наприклад, з cgroup може бути обмежена кількість пам'яті, яку може використовувати контейнер. Cgroups - це звичайні групи процесів Linux, які можуть працювати поруч із будь-якими процесами, що ведуть на ОС. Одна важлива відмінність у розподілі ресурсів між LXC та гіпервізорами полягає в тому, що ресурси центрального процесора не можуть бути розподілені на основі базової бази, а потребують визначений пріоритет.

1.5.3 Сучасний Docker

Docker - інструмент який дозволяє повністю ізолювати додаток. Забезпечує автоматизацію розгортання і управління додатками. Надає зручний інтерфейс для роботи з LXC. Дозволяє створювати стерпні контейнери з оточенням і всіма залежностями. Також є можливість управління обчислювальними ресурсами контейнера.

Додатки працюють в ізольованому середовищі (побудованої за допомогою просторів імен, namespaces, і груп процесів, cgroups), з одного боку ізолюючі процеси один від одного, з іншого боку не вдаючись при цьому до таким надлишкових засобів як віртуалізація або емуляція.

Система написана на Go. Працює в різних UNIX / Linux-системах. Ядро Docker може запускати будь-який додаток безпечно ізольованому в контейнері. Ізоляція дозволяє запускати одночасно безліч контейнерів на одному вузлі (Рисунок. 1.3).

Легка природа контейнера, який не вимагає гіпервізора, дозволяє
отримувати більше від хардвару.

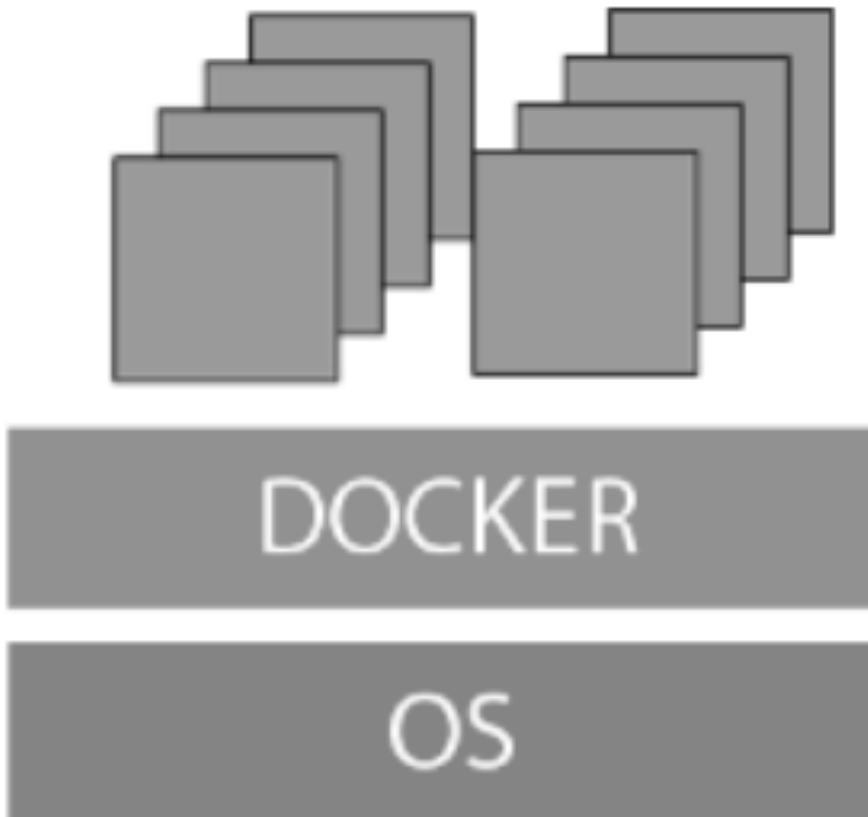

Рисунок 1.3 – Віртуалізація засобами Docker

Docker використовує «листкову» файлову систему AuFS, завдяки якій контейнери можуть спільно використовувати однакові частини файлової системи, доступні тільки на читання (Рисунок. 1.4).

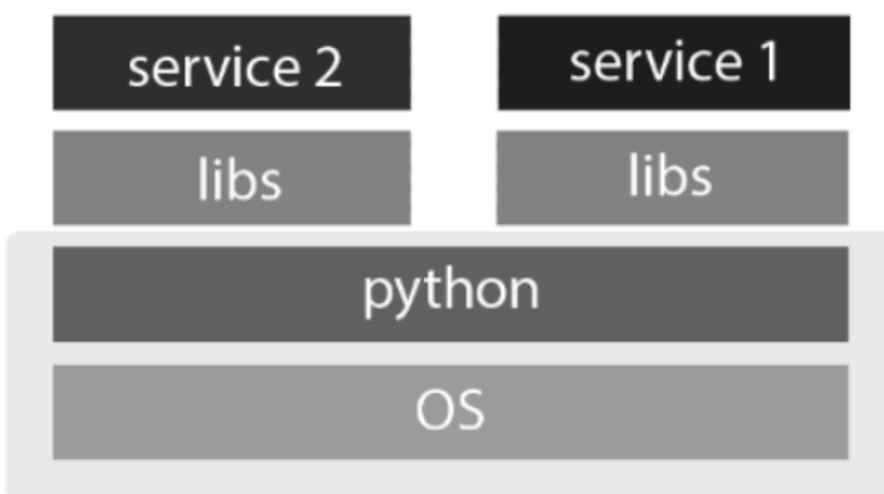

Рисунок 1.4 – Приклад образу Docker

Docker легковагий і швидкий. Він є стійкою, альтернативою віртуальним машинам на основі гіпервізора. Він корисний при високих навантаженнях, наприклад, при розробці власної хмари або PaaS. Його також можна використовувати для середніх і малих додатків, коли потрібно отримати більше від наявних ресурсів. Docker використовує клієнт-серверну архітектуру. Docker клієнт спілкується з сервером Docker, який бере на себе відповідальність створення, запуску, розподілу контейнерів. Клієнт та сервер можуть працювати на одній системі, клієнта можливо підключити до віддаленого серверу Docker.

Клієнт і сервер працюют або по протоколу REST або через сокет. Користувач може взаємодіяти з сервером тільки через клієнт-додаток. Клієнт взаємодіє з сервером отримуючи команди від користувача. Docker складається з трьох компонентів:

1) зображень (images)
2) реєстр (registries)
3) контейнери

Docker-образ - це read-only шаблон. Образ, наприклад, може містити Ubuntu c Nginx і розгорнутим додатком. Образи використовуються для створення контейнерів. Docker з легкістю дозволяє створювати нові образи, оновлювати існуючі, створювати на базі готових образів нові.

Docker-реєстр зберігає образи. Реєстри розподіляють на публічні та приватні. У публічному Docker реєстрі знахотиться база даних образів дуже великого обсягу. Контейнери нагадують директорії. У контейнерах міститься все необхідне для роботи програми. З контейнерами можна виконувати наступні дії: створювати, запускати, зупиняти, переносити та видаляти. Кожен контейнер може бути створен тільки з образу.

Будь-який контейнер має такі властивості як ізольованність і безпечність, заснована платформою додатка. В основі будь-якого образу лежить базовий образ. Мжна навести наступні приклади: debian це базовий образ Debian, mint – Mint, fedora – Dedora, ubuntu – Ubuntu. Використування образу для створення нового – загальна практика. Якщо ми маємо образ flask, то його можна використати як базову для Python

веб-додатків. Docker образи можна створювати з базових образів. Будь-яка інструкція може створити образ. Говорячи про інструкції зазвичай розуміють наступне: робота з файлами та директоріями, створення environment оточення, підказки щодо запуску контейнерів та запуск і виконання Docker-команди.

1.6 Переваги використання віртуалізації

Нижче наведено кілька репрезентативних причин та переваг віртуалізації:

- Віртуальні машини можуть бути використані для консолідації робочих навантажень кількох невикористаних серверів меншою кількістю машин, можливо, однієї машини (консолідація серверів). Пов'язані переваги (сприймаються або реальні, але часто цитуються продавцями) - це заощадження на апаратній, екологічних витратах, управління та адміністрування серверної інфраструктури.
- Необхідність запуску застарілих програм добре обслуговується віртуальними машинами. Спадкова програма просто не може працювати на новіших апаратних і / або операційних системах. Навіть якщо це станеться, якщо він може недостатньо використовувати сервер, то, як зазначено вище, має сенс консолідації кількох програм. Це може бути складно без віртуалізації, оскільки такі програми, як правило, не записуються для співіснування в межах одного середовища виконання (розглядайте програми як жорсткі кодування ключів IPC системи V як тривіальний приклад).
- Віртуальні машини можуть бути використані для забезпечення безпечних, окремих ізольованих програм для запуску ненадійних програм. Ви навіть можете створити таке середовище виконання динамічно - на льоту - як ви завантажуєте щось з Інтернету та запускаєте його. Ви можете думати про творчі схеми, наприклад, ті, що пов'язані з обфускацією адреси. Віртуалізація є важливою концепцією створення безпечних обчислювальних платформ.
- Віртуальні машини можуть бути використані для створення операційних систем або середовищ виконання з обмеженнями ресурсів, а також з

урахуванням правильних планувальників, гарантій ресурсів. Розбиття зазвичай відбувається з рукостисканням якості обслуговування при створенні операційних систем з підтримкою QoS.

- Віртуальні машини можуть забезпечити ілюзію апаратної чи апаратної конфігурації, якої у вас немає (наприклад, пристрої SCSI, кілька процесорів ...). Віртуалізація також може використовуватися для моделювання мереж незалежних комп'ютерів.
- Віртуальні машини можуть використовуватися для одночасного керування кількома операційними системами: різні версії або навіть цілком різні системи, які можуть перебувати у режимі очікування. Деякі такі системи можуть бути важко або неможливо запустити на новому реальному апаратному забезпеченні.
- Віртуальні машини дозволяють проводити потужні налагодження та моніторинг продуктивності. Наприклад, такі інструменти можна розмістити на моніторі віртуальної машини. Операційні системи можуть бути налагоджені без втрати продуктивності або створення більш складних сценаріїв налагодження.
- Віртуальні машини можуть ізолювати те, що вони запускають, тому вони забезпечують усунення несправностей та помилок. Ви можете попередньо ввести недоліки в програмне забезпечення для вивчення його подальшої поведінки.
- Віртуальні машини роблять програмне забезпечення легшим для міграції, тим самим допомагаючи прикладній програмі та мобільності системи.
- Ви можете розглядати аплікації програми як пристрій за допомогою "упаковки" та запускати кожен на віртуальній машині.
- Віртуальні машини - це чудовий інструмент для досліджень та академічних експериментів. Оскільки вони забезпечують ізоляцію, вони безпечніше працювати. Вони інкапсулюють весь стан працюючої системи: ви можете зберегти стан, перевірити його, змінити його, перезавантажити і так далі.

Держава також забезпечує абстракцію завантажуваного робочого навантаження.

- Віртуалізація може дозволити існуючим операційним системам працювати на мультипроцесорів із загальною пам'яттю.
- Віртуальні машини можуть бути використані для створення довільних тестових сценаріїв і можуть призвести до дуже образного, ефективного забезпечення якості.
- Віртуалізація може бути використана для модернізації нових функцій у існуючих операційних системах без "надто багато" роботи.
- Віртуалізація може зробити завдання, такі як міграція системи, резервне копіювання та відновлення, простішими та більш керованими.
- Віртуалізація може стати ефективним засобом надання двійкової сумісності.
- Віртуалізація на товарному обладнанні була популярною у спільному хостингу. Багато перерахованих вище переваг робить такий хостинг безпечним, економічним і загалом привабливим.
- Віртуалізація це весело.

1.7 Висновки

У даному розділі було розглянуто та досліджено технології віртуалізації та контейнеризації. Дослідження було націлене на описання особливостей та визначення проблематики при реалізації даної задачі, розглянуто популярні рішення, які допоможуть вирішити проблеми задачі.

Лідером у сфері контейнерних технологій сьогодні є Docker. Docker в останні роки став найпопулярнішою технологією і це підтверджується тим, що ним користуються такі гіганти як Google, Facebook, Yahoo, IBM.

Враховуючи предмет дослідження даного завдання, була сформульована задача, у якій описано основні проблеми, які треба вирішити для налаштування середовища, за допомогою якого можна дослідити керування потоками даних.

2 НАЛАШТУВАННЯ ТА ЗАПУСК DOCKER ТА KUBERNETES

2.1 Використання Docker

Docker - opensource-додаток (проект або інфраструктура), яке дозволяє упаковувати, поширювати, встановлювати і використовувати opensource-додатки.

Система написана на Go. Працює в різних UNIX / Linux-системах, очікується, що буде підтримуватися і Windows. Docker зазвичай використовує різні системи віртуалізації на рівні ОС, в даний час найбільшого поширення набула комбінація Docker + LXC, на якій він і був спочатку заснований. Це призвело до того, що багато хто плутає Docker і LXC, в дійсності, це абсолютно різні проекти з різними функціями.

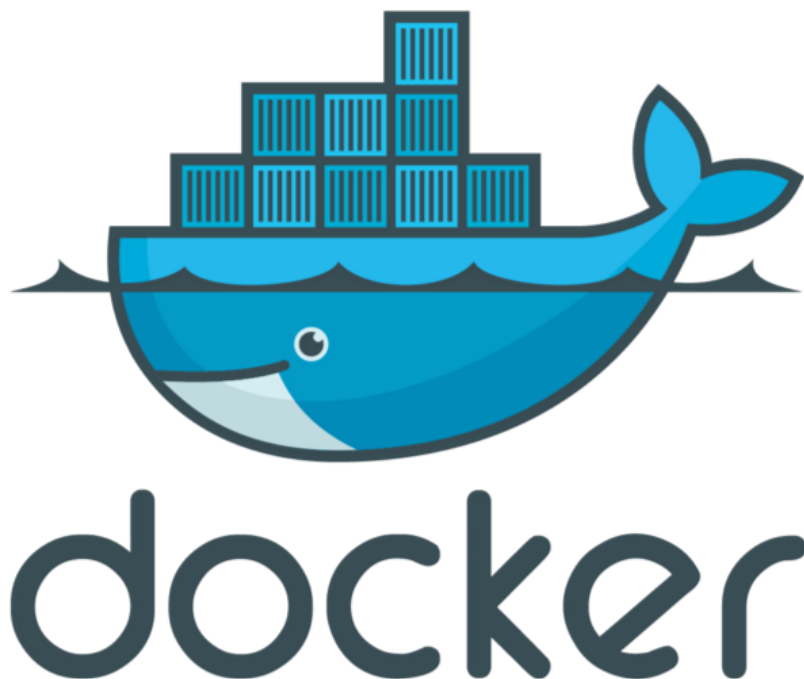

Рисунок 2.1 – Логотип Docker[3]

“У своєму ядрі Docker дозволяє запускати практично будь-який додаток, безпечно ізольоване в контейнері. Безпечна ізоляція дозволяє вам запускати на одному хості багато контейнерів одночасно. Легка природа контейнера, який запускається без додаткового навантаження гіпервізора, дозволяє вам добиватися більше від вашого заліза.”[4]

Швидке викладання ваших додатків. Docker дуже добре використовувати для Countiniuos Developmenta. Docker дозволяє девелоперам використовувати локальні контейнери для розробки мікросервісів та додатків. Надалі це може дозволити виконати інтеграцію та легко викласти мікросервіс у продакшен (CI and CD). Наведемо приклад: девелопери розробляють продукт локально, а потім вони можуть розшарити свій локальніий набір докер образів зі своїми колегами. Коли вони завершують цикл розробки, вони роблять білд образів, а потім заливають їх на тестовий майданчик, де можна зробити перевірку лінтерами, або запустити тести. З тестового плацдарму можна залити образи на продакшен сервери.

Докер не є заміною для LXC. "LXC" ставиться до можливостей ядра Linux (зокрема, імена та контрольні групи), які дозволяє пісочниця процесів один від одного, і контролює розподілу їх ресурсів. На низькому рівні функцій ядра, Docker пропонує інструмент високого рівня з кількома потужними функціональними можливостями:

- Портативний розгортання кількох машин
- Орієнтовані додатки
- Автоматична збірка
- Версії
- Компонент повторного використання
- Обмін
- Інструмент екосистеми

Більш просте викладання і розгортання. Якщо використовувати контейнера Docker, то з вашою платформою можна будет дуже легко працювати. З Docker-

контейнерами можна працювати локально, за домогою віртуальної машини, яка розгорнута локально, також можна розгорнути в хмарному середовищі, або навіть у дата-центрах. Docker дуже легко можна портувати з одного пристрою на іншій, а також дуже легко слідкувати за корисним навантаженням. За домогою Docker можна як запускати, так і зупиняти мікросервіси. Швидкість сучасного Docker наближається до подій реального часу.

Високі навантаження і більше корисних навантажень. Docker дуже легкий та швидкий у використанні. Він є сучасною альтернативою віртуальних машин, але й з багатьма перевагами. Docker неймовірно незамінний в умовах high load, наприклад, при створення власного хмарного середовища або при створенні PaaS. Також Docker корисний при використанні у маленьких та середніх додатків, якщо є необхідність в економії ресурсів.

2.2 Головні компоненти Docker

2.2.1 Архітектура Docker

Docker-demon

На рисунку 2.2 показано як демон запускається на хості. Користувач використовує к лієнт, щоб взаємодіяти із сервером.

Docker-client

Docker-client це головний компонент Docker, він може взаємодіяти з docker-demon за домогою набору команд, які вводить користувач.

Усередині Docker-a

Docker зазвичай складається із трьох основних компонентів:

- **Docker-image – це зазвичай read-only паттерн**. Наприклад, image може у собі містити ОС Mint з Flask додатком на ній. Images зазвичай використовуються для того, щоб створювати нові контейнери з сервісами усередині. Docker створили для того, щоб легко створювати нові образи, редагувати існуючі, та їх видаляти. Так розробники можуть завантажити собі розарені образи інших

людей та використати їх як основу можете завантажити образи створені іншими людьми. Образи або як ще кажуть Images - це головний компонент Docker.

- **Docker-register зберігає існуючі образи**. Існують публічні(public) та приватні(private) реєстри. У цих реєстрах можна знайти та завантажити корисні образи, щоб не створювати дублікати. Public Docker-реєстр називається Docker Hub. Там зберігається неймовірна кількість образів. Образи можна створювати, або використовувати вже існуючі, які були створені іншими людьми. Реєстри називають компонентом поширення образів.
- **Контейнери**. Вони нагадують нам звичайні директорії операційної системи. У контейнерах міститься усе необхідне, що потім буде використано для роботи програми. Щоб отримати контейнер його необхідно створити з образу. З контейнерами можна виконувати наступні дії: створювати, запускати, зупиняти та видаляти. Будь-який контейнер унікальний і є безпечним для усього додатка. Контейнери називають компонентом роботи.

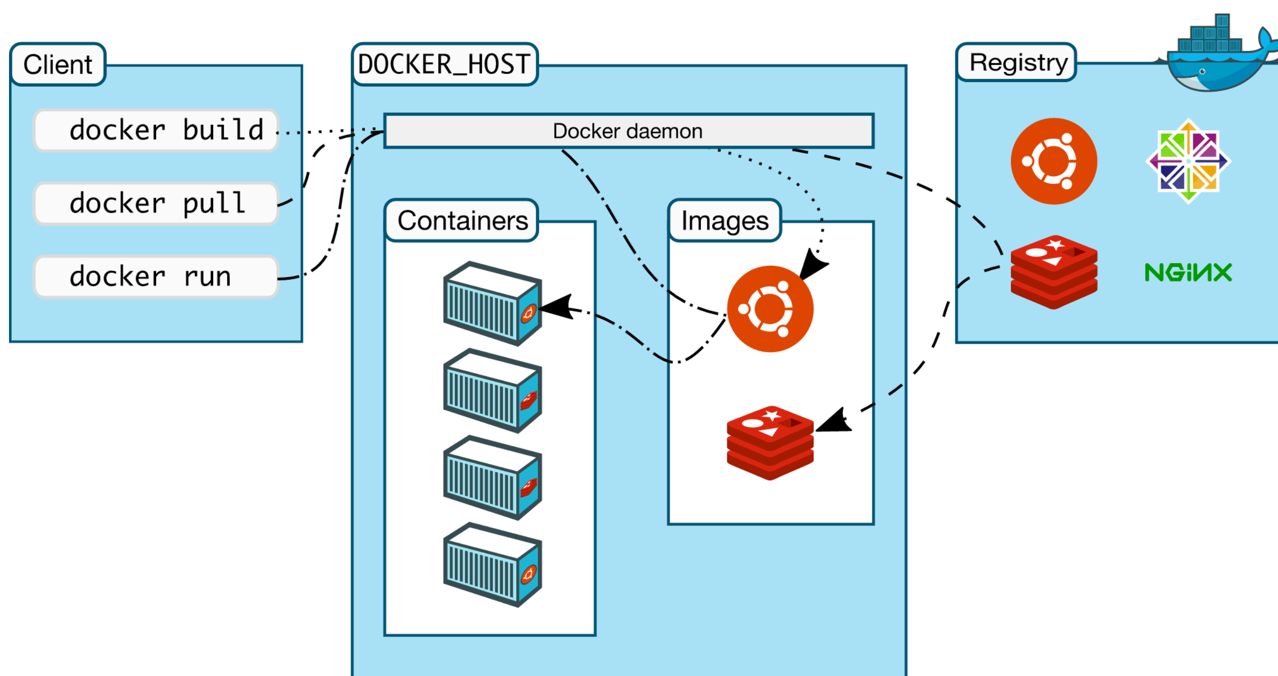

Рисунок 2.2 – Архітектура Docker[5]

2.2.2 Робота зображення

Образ - це read-only паттерн, з якого може бути створен контейнер. Будь-який образ звичайно складається з різного набору рівнів. Docker використовує UFS для того, щоб ці рівні можна було поєднати в один образ. UFS створює свою файлову систему за допомогою прозорого накладання файлів та директорій з різних гілок системи.

Саме завдяки використанню таких рівні Docker такий легкий у розмірах. Для того щоб змінити додаток вам треба всього додати новий рівень. У цьому перевага Docker перед VM – вам треба додати новий рівень і не треба пересбирати увесь додаток. Це дозволяє працювати з образами, і поширювати їх набагато простіше і швидше.

2.2.3 Робота Docker реєстра

Register - це сховище Docker образів. Після створення образу ви можете виконувати різні дії з ним, зокрема опублікувати його на Docker Hub, також ви можете створити свій локальний реєстр, який буде доступним лише усередині команди.

За допомогою Docker ви легко можете знайти раніше опубліковані образи і завантажити їх на локальну машину, де можна їх використати для створення своїх нових контейнерів.

У Docker Hub існують публічні(public) та приватні(private) реєстри. Пошук і завантаження образів з публічних реєстрів доступне для всіх. Образи з приватних репозиторіїв не попадають до результатів пошуку, і їх можуть використовувати тільки обрані люди та створювати з них контейнери.

2.2.4 Робота контейнерів

До складу контейнеру входить операційна система, призначена для користування файлами та метаданими. Будь-який контейнер створюється з образу. Цей образ передає Docker, що він знаходиться усередені його контейнера, а також надає інформацію про конфігураційні дані. Docker образ використовує read-only паттерн. Docker під час запуску контейнеру створює рівень для читання та запису зверху образу у якому може бути запущено додаток.

2.3 Встановлення Docker

2.3.1 Відносини з Docker Machine

У роботі ми будемо використовувати ОС Mac OS.

Встановлення Docker для Mac не впливає на машини, створені вами за допомогою Docker Machine. У вас є можливість копіювати контейнери та зображення з локальної машини за замовчуванням (якщо вона існує) на нову Docker для Mac HyperKit VM. Коли ви використовуєте Docker для Mac, вам не потрібні вузли Docker Machine, які працюють на всіх місцях (або деінде). За допомогою Docker для Mac ви маєте нову систему власної віртуалізації (HyperKit), яка займає місце системи VirtualBox.

2.3.2 Системні вимоги

Докер для Mac запускається, лише якщо виконано всі ці вимоги:

- Обладнання Mac повинно бути 2010 або новішою моделлю, апаратна підтримка Intel для віртуалізації модуля керування пам'яттю (MMU), включаючи розширені таблиці сторінок (EPT) та необмежений режим. Ви можете перевірити, чи має ваша машина таку підтримку, запустивши на терміналі таку команду: sysctl kern.hv_support
- MacOS El Capitan 10.11 та новіші версії MacOS підтримуються. Як мінімум, Docker для Mac вимагає MacOS Yosemite 10.10.3 або новішої версії, з застереженням, що перехід вперед 10.10.x - це власне застосування ризику.
- Починаючи з Docker для Mac Stable release 1.13 та паралельних випусків Edge, ми більше не розглядаємо проблеми, що стосуються macOS Yosemite 10.10. У майбутніх випусках Docker для Mac може припинити працювати на MacOS Yosemite 10.10 через застарілий статус цієї версії macOS. Ми рекомендуємо оновити до останньої версії macOS.
- Принаймні 4 ГБ оперативної пам'яті
- VirtualBox перед версією 4.3.30 НЕ повинен бути встановлений (це несумісне з Docker для Mac). Якщо ви встановили нову версію VirtualBox, це добре.

2.3.3 Завантаження Docker

Щоб завантажити Docker треба перейти за посиланням https://docs.docker.com/install/

About Docker CE

Estimated reading time: 2 minutes

Docker Community Edition (CE) is ideal for developers and small teams looking to get started with Docker and experimenting with container-based apps. Docker CE has two update channels, **stable** and **edge**:

- **Stable** gives you reliable updates every quarter.
- **Edge** gives you new features every month.

For more information about Docker CE, see Docker Community Edition.

Supported platforms

Docker CE is available on multiple platforms, on cloud and on-premises. Use the following tables to choose the best installation path for you.

Desktop

Platform	x86_64
Docker for Mac (macOS)	✔
Docker for Windows (Microsoft Windows 10)	✔

Рисунок 2.3 – Завантаження Docker

Далі необхідно вибрати платформу. У нашому випадку необхідно натиснути на Docker for Mac (macOS). І ви побачите наступне:

Install Docker for Mac

Estimated reading time: 4 minutes

Docker for Mac is the Community Edition (CE) of Docker for MacOS. To download Docker for Mac, head to Docker Store.

Download from Docker Store

Рисунок 2.4 – Завантаження Docker

Необхідно натиснути на Download from Docker Store, після цього почнеться завантаження. Після того як Docker був завантажен треба двічі натиснути на

Docker.dmg, щоб відкрити інсталер. Після цього требе перетянути кита Мобі до папки з додатками.

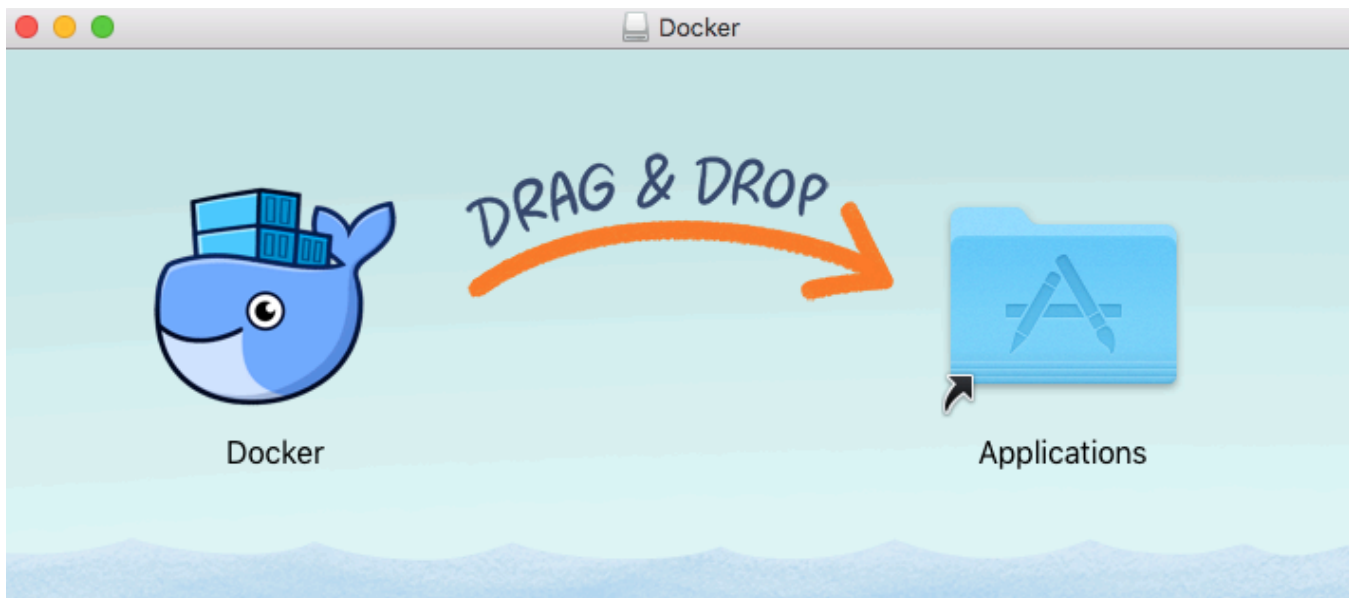

Рисунок 2.5 – Встановлення Docker

Щоб відкрити Docker треба знайти його у додатках та натиснути на іконку Docker

Рисунок 2.6 – Запуск Docker

Після запуску ви побачите зображення Docker у панелі зверху.

Рисунок 2.7 – Піктограма Docker

Через деякий час з’явиться вікно Docker, де можна побачити статус, у якому він зараз знаходиться.

Рисунок 2.8 – Статус Docker

Docker is starting – Docker запускається та ще не готовий для роботи.
Docker is running – Docker готовий для роботи.

2.4 Тестове налаштування Docker

2.4.1 Використання Dockerfile

Dockerfile використовує звичайний DSL з інструкціями для побудови образів Docker. Після цього виконується команда docker build для побудови нового образу з інструкціями в Dockerfile.

Переконайтеся, що ваші версії докерів, докерів та docker-машини є найновішими та сумісні з Docker.app. Ваш результат може відрізнятися, якщо ви використовуєте різні версії.

```
Andreys-Air:~ Andrey$ docker --version
“Docker version 18.02.2-ве, build 8ee2c39”
Andreys-Air:~ Andrey$
Andreys-Air:~ Andrey$ docker-compose --version
docker-compose version 1.21.1, build 5a3f1a3
Andreys-Air:~ Andrey$ docker-machine --version
docker-machine version 0.14.0, build 89b8332
```

Відкрийте термінал командного рядка та перевірте, чи працює ваша установка за допомогою простого зображення Docker, hello world:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest: sha256:f5233545e43561214ca4891fd1157e1c3c563316ed8e237750d59bde73361e77
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

Рисунок 2.9 – Тестовий запуск Docker

Запустіть Dockerized веб-сервер. Як і зображення привітного світу, якщо зображення не знайдено локально, Докер витягує його з Docker Hub.

```
Andreys-Air:~ Andrey$ docker run -d -p 80:80 --name webserver nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
f2aa67a397c4: Pull complete
3c091c23e29d: Pull complete
4a99993b8636: Pull complete
Digest: sha256:0fb320e2a1b1620b4905facb3447e3d84ad36da0b2c8aa8fe3a5a81d1187b884
Status: Downloaded newer image for nginx:latest
e841d41cb4a7140b2c4a5fd0620bc2ea2db4ff1c2c0ee4e8fd0e60593ecd9506
```

Рисунок 2.10 – Запуск Docker серверу

У веб-переглядачі перейдіть на http://localhost/, щоб переглянути домашню сторінку nginx. Оскільки ми вказали типовий порт HTTP, не потрібно додавати: 80 в кінці URL-адреси.

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Рисунок 2.11 – Nginx локальна сторінка

Перегляньте деталі контейнера під час роботи вашого веб-сервера (з контейнером docker ls або docker ps):

```
Andreys-Air:~ Andrey$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                NAMES
e841d41cb4a7        nginx               "nginx -g 'daemon of…"   4 minutes ago       Up 4 minutes        0.0.0.0:80->80/tcp   webserver
```

Рисунок 2.11 – Docker контейнери

Зупиніть та видаліть контейнери та зображення за допомогою наступних команд. Використовуйте прапорець "all" (--all або -a), щоб переглянути зупинені контейнери.

```
Andreys-Air:~ Andrey$ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                NAMES
e841d41cb4a7        nginx               "nginx -g 'daemon of…"   6 minutes ago       Up 6 minutes        0.0.0.0:80->80/tcp   webserver
Andreys-Air:~ Andrey$ docker container stop webserver
webserver
Andreys-Air:~ Andrey$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                             PORTS               NAMES
e841d41cb4a7        nginx               "nginx -g 'daemon of…"   6 minutes ago       Exited (0) Less than a second ago                      webserver
44d6c0bf66fe        hello-world         "/hello"                 9 minutes ago       Exited (0) 9 minutes ago                               tender_swanson
Andreys-Air:~ Andrey$ docker container rm webserver
webserver
Andreys-Air:~ Andrey$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              ae513a47849c        8 days ago          109MB
hello-world         latest              e38bc07ac18e        3 weeks ago         1.85kB
Andreys-Air:~ Andrey$ docker image rm nginx
Untagged: nginx:latest
Untagged: nginx@sha256:0fb320e2a1b1620b4905facb3447e3d84ad36da0b2c8aa8fe3a5a81d1187b884
Deleted: sha256:ae513a47849c895a155ddfb868d6ba247f60240ec8495482eca74c4a2c13a881
Deleted: sha256:160a8bd939a9421818f499ba4fbfaca3dd5c86ad7a6b97b6889149fd39bd91dd
Deleted: sha256:f246685cc80c2faa655ba1ec9f0a35d44e52b6f83863dc16f46c5bca149bfefc
Deleted: sha256:d626a8ad97a1f9c1f2c4db3814751ada64f60aed927764a3f994fcd88363b659
```

Рисунок 2.12 – видалення контейнерів Docker

Таким чином ми познайомиилися з Docker. Ми запустили тестовий приклад, познайомилися з основними командами Docker. Після цього ми видалили наші контейнери.

2.5 Загальні налаштування Docker

2.5.1 Основні

Виберіть меню -> Параметри на панелі меню та налаштуйте параметри часу виконання, описані нижче.

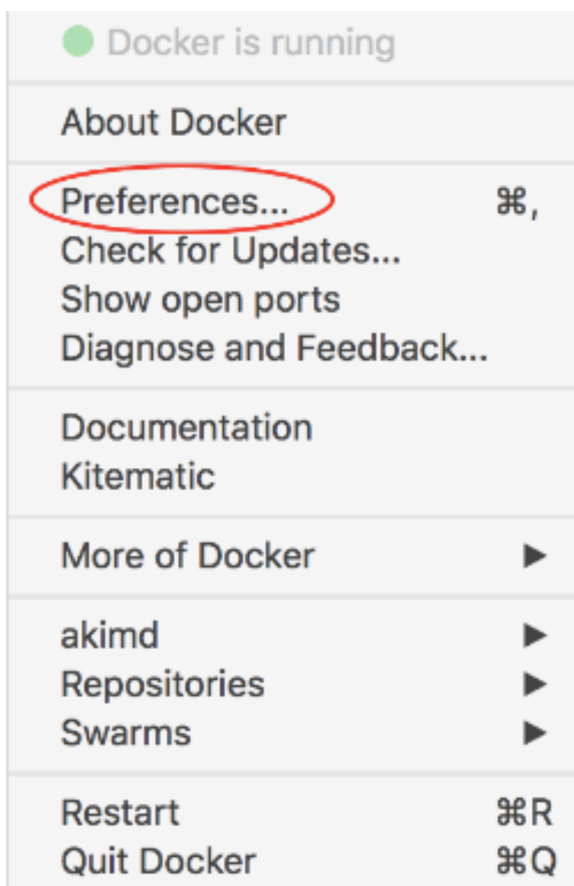

Рисунок 2.13 – Налаштування Docker

Тут ви зможете налаштувати Docker через гарній юзер інтерфейс.

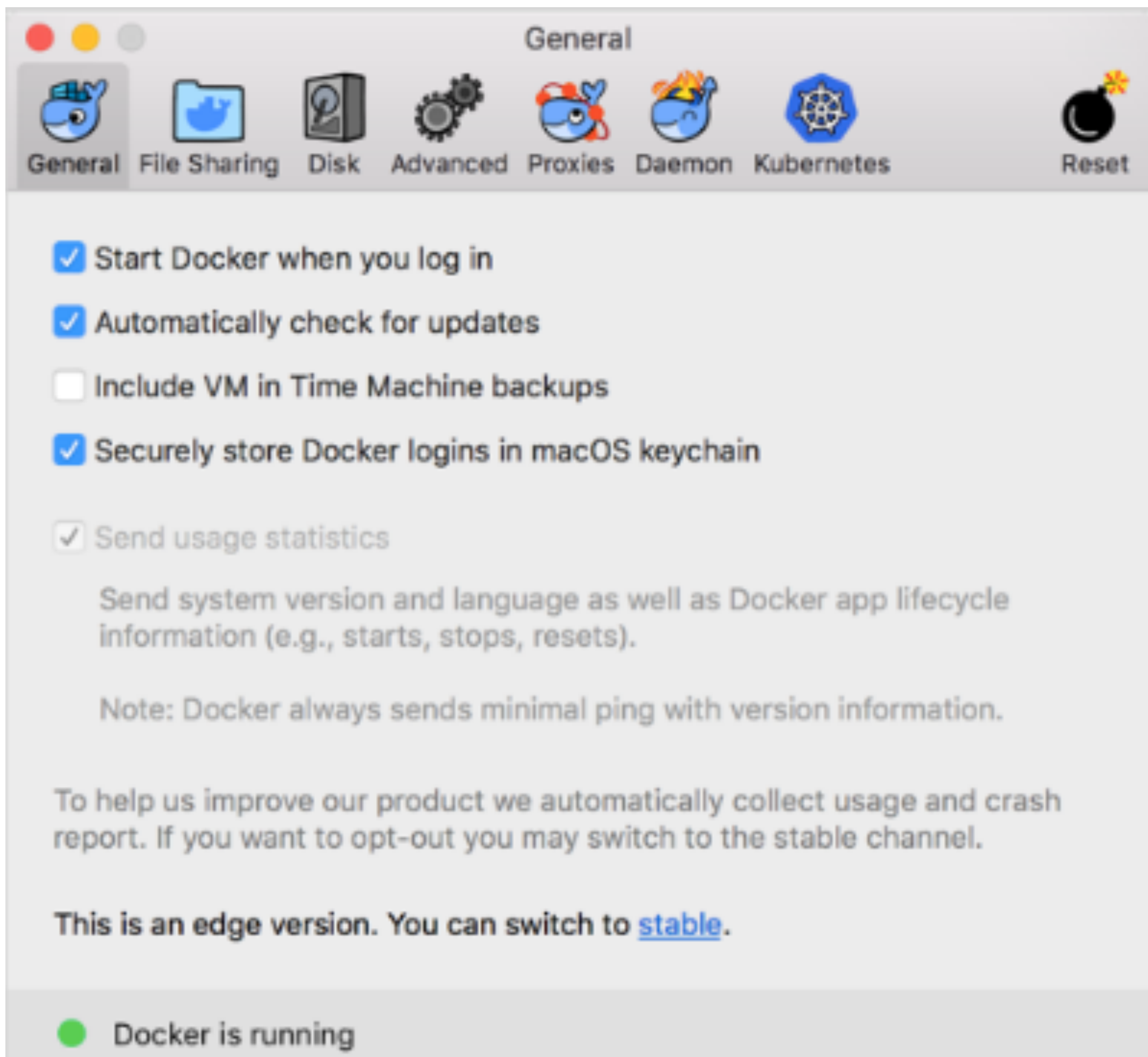

Рисунок 2.14 – Основні налаштування Docker

Загальні параметри:

- **Запустіть Docker під час входу**: Зніміть прапорець біля цієї опції, якщо ви не хочете, щоб Docker запускався, коли ви відкриваєте сеанс.
- **Автоматична перевірка наявності оновлень** повідомляє вас, коли доступне оновлення. Натисніть кнопку ОК, щоб прийняти та встановити оновлення (або скасувати, щоб зберегти поточну версію). Якщо ви вимкнете цей параметр, ви все одно можете дізнатися про оновлення вручну, вибравши меню китів -> Перевірити наявність оновлень.

- **Включення віртуальної машини в резервні копії Time Machine** створює резервну копіювання віртуальної машини Docker для Mac. (За замовчуванням вимкнено).
- **Безпечне зберігання логінів Docker у MacOS keychain** зберігає ваші вхідні дані Docker. (Увімкнено за умовчанням.)
- **Надіслати статистику використання** - Надсилати діагностику, звіти про аварійне завершення роботи та дані про використання до Docker. Ця інформація допомагає Docker покращити програму та отримати більше контексту для вирішення проблем. (Увімкнено за умовчанням.)

2.5.2 Доступ до файлів

Виберіть, які місцеві директорії матимуть доступ до ваших контейнерів. Обмін файлами потрібен для монтажу об'єму, якщо проект знаходиться за межами каталогу/Користувач. У цьому випадку, поділіться диском, де розташовано файл Dockerfile та обсяг. В іншому випадку ви не отримаєте файл або не можете запустити сервісні помилки під час виконання.

Рисунок 2.15 – Доступ до файлів

Параметри спільного доступу до файлу:

- Додайте каталог: натисніть + і перейдіть до каталогу, який хочете додати.
- Застосувати і перезапустити робить каталог доступним для контейнерів за допомогою функції кріплення Docker (-v).

Є деякі обмеження щодо каталогів, якими можна поділитися:

- Вони не можуть бути підкаталогами вже спільного каталогу.
- Вони не можуть існувати всередині Докеру.

2.5.3 Розширенні налаштування

На вкладці Розширенні ви можете обмежити ресурси, доступні для Docker.

Рисунок 2.16 – Розширенні налаштування

Додаткові параметри:

- CPU: за замовчуванням Docker для Mac встановлено половину кількості процесорів, доступних на хост-машині. Щоб збільшити потужність обробки, встановіть її на більш високий номер; зменшити, зменшити кількість.
- Memory: за замовчуванням Docker для Mac встановлено 2 ГБ оперативної пам'яті, виділеної з загальної доступної пам'яті вашого Mac. Щоб збільшити об'єм пам'яті, встановіть його на більш високий номер; зменшити його, зменшити кількість.
- Swap: Налаштуйте розмір файлу підкачки, коли це потрібно. За замовчуванням встановлено 1 Гб.

2.5.4 Диск

Вкажіть розташування диска для об'єму Linux, де зберігаються конфігурації та зображення.

Ви також можете перемістити розташування диска зображення. Якщо ви намагаєтесь перемістити зображення диска до місця, яке вже є, ви отримаєте запит про те, чи хочете ви використати існуюче зображення чи замінити його.

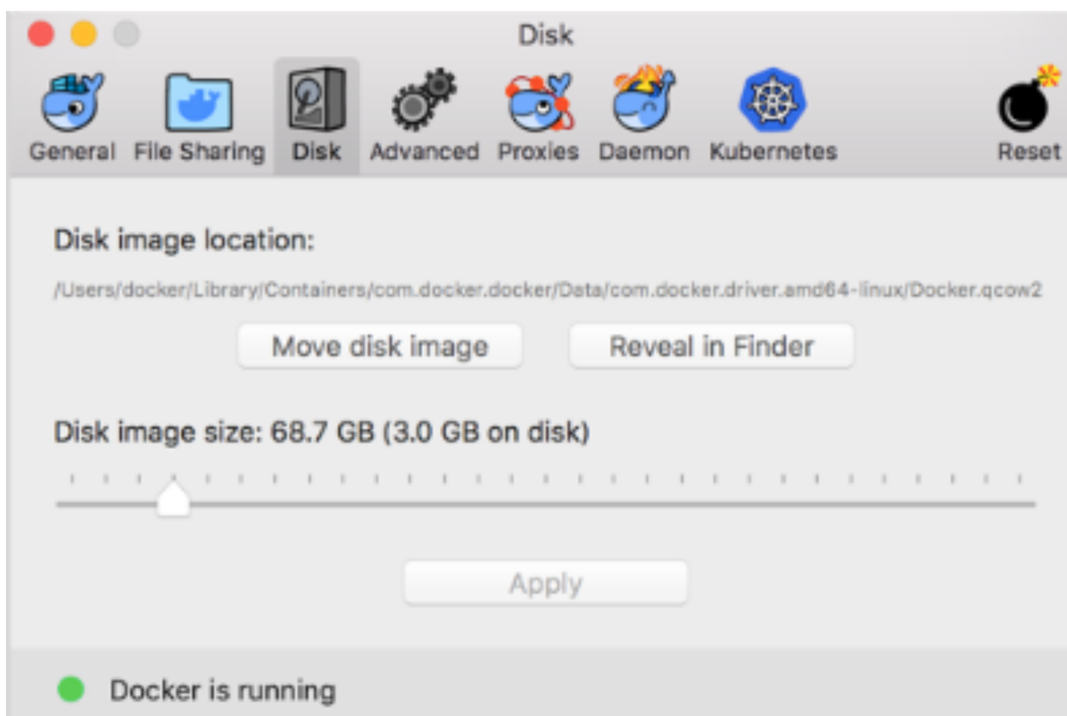

Рисунок 2.17 – Налаштування диску

2.6 Огляд Kubernetes

Kubernetes - це призначений для контейнерної оркестровки фреймворк з відкритим вихідним кодом. Він був створений з урахуванням багатючого досвіду Google в області створення середовищ управління контейнерами і дозволяє виконувати контейнерні додатки у готовому для промислової експлуатації кластера. В механізмі Кубернети багато рушійних частин і способів їх настройки - це різні системні компоненти, драйвери мережевого транспорту, утиліти командного рядка, не говорячи вже про додатки та робочі навантаження.

Рисунок 2.16 – Логотип Kubernetes[6]

Kubernetes опен-сорс проект, який розробили для керування кластерами контейнерів як одни цілим(одна система). Kubernetes може управляти та запускати одночасно велику кількість контейнерів, а також може забезпечувати спільне розміщення та реплікацію. Проект був розроблен у Google і тепер його підтримує багато гігантів, серед яких є такі як Yahoo, Microsoft, Apple, Amazon, Bosch, IBM, Docker та Samsung.

Google користується контейнерної технологією вже більше десяти років. Вона має такий унікальний досвід як запуск близько 2 млрд контейнерів за один тиждень. Google вирішила поділитися своїм досвідом у ствоерні платформи, яка може масштабувати запуск контейнерів – це все увійшло у Kubernetes.

Проект переслідує на меті дві цілі. Kubernetes допомагає запускати велику кількість контейнерів на різних хостах, а також допомагає виконувати балансування навантаження. У проекті існує API, через яке можна виконувати майже усі операції віддалено.

Сьогодні виділяють наступні концепції:

- Node: машина в кластері Kubernetes.
- Pod: група контейнерів які запускаються як єдине ціле.
- Replication Controllers: виконує реплікацію.
- Services: абстракція над подами та їх політикою доступу.
- Volumes: директорія яка доступна в контейнері.
- Labels: пари ключ-значення які прикріплюються до pod.
- kubectl: інтерфейс командного рядка для управління Kubernetes.

Якщо казати про архітектуру, то можна сказати наступне. Працюючи Kubernetes кластер складається з агента, який запускаться на kubelet та master компоненти поверх розподіленого сховища.

Можна розбити архітектуру системи на сервіси, які працюють на кожній ноді і сервіси рівня управління кластера. На кожній ноді Kubernetes запускаються

мікросервіси, які обовязкові для керування нодою через master та для старту додатків. Звичайно, на будь-якій ноді буде запущенно Docker.

Kubelet може керувати подами, образами, а також контейнерами та розділами. На буд-якій ноді буде запущен балансувальник. Цей сервіс працює на кожній ноді у кластері і може бути налаштован через Kubernetes API.

Система керування Kubernetes поділяється на компоненти. Сьогодні усі компоненти запускаються через master, але зовсім скоро це змінять, щоб була можливість опрацьовувати відмови кластеру. Усі компоненти працюють разом, щоб була єдина система(один екземпляр кластеру).

Стан майстра можна перевіриту в **etcd**. Завдяки цьому можна надійно зберігати файли конфігурації та своєчасне сповіщення інших компонентів системи про зміну стану.

Kubernetes API компонент відповідає за роботу api сервера. Він призначений для того, щоб обробляти CRUD операції та виконувати бізнес-логіку, яка була розроблена в інших компонентах. Зазвичай обробляє REST, виконуючи перевірку і оновлюючи стан у etcd.

2.7 Висновки

У цьому розділі було розглянуто архітектуру Docker та Kubernetes. Також було детально розглянуто процес інсталяції та налаштування на тестовому прикладі. У результаті ми побачили, що зараз можна дуже легко та швидко налаштувати Docker та Kubernetes. Як результат, ми завантажили та налаштували середовище, щоб провести дослідження можливості керування потоками даних.

3 КЕРУВАННЯ ПОТОКАМИ ДАНИХ У СЕРЕДІ IOT ЗА ДОПОМГОЮ DOCKER ТА KUBERNETES

3.1 Dataflow managements systems

За ідеалогією Docker ми повині для кожного мікросервісу робити свій контейнер. В першу чергу це пов’язано з fault tolerance. Тому що помилка в одному мікросервісі потягне за собою інші. Тому ми будемо робити кожний мікросервіс в окремому контейнері, а потім об’єднаємо усі контейнера в один кластер за допомогою Kubernetes, який і був розроблен для оркестрацією контейнерних кластерів.

Ми будемо використовувати новий підхід у роботі – дані будуть керувати мікросервісами, а не навпаки. Це означає, що ми будемо використовувати наші мікросервісі у вигляді редюсерів, а функції які будуть виконувати наші редюсери ми будемо задавати у даних. В результаті ми отримуємо реактивне програмування.

3.2 Розробка мікросервісів з використанням NodeJS

Розробимо мікросервіси, які будуть отримувати HTTP-запити через API. Та пересилати ці запити на інші. Мікросервіси будуть розроблені за допомогою NodeJS. Перш за все треба встановити NodeJS. Для цього необхідно встановити **Homebrew**.

ruby -e "$(curl –fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

Після цього треба встановати NodeJS.

brew install node

Після цього створимо папку проекту

mkdir micro

cd micro

Ми будемо використовувати HTTP пакет для створення HTTP-серверу.

npm init

Після цього треба створити index.js файл

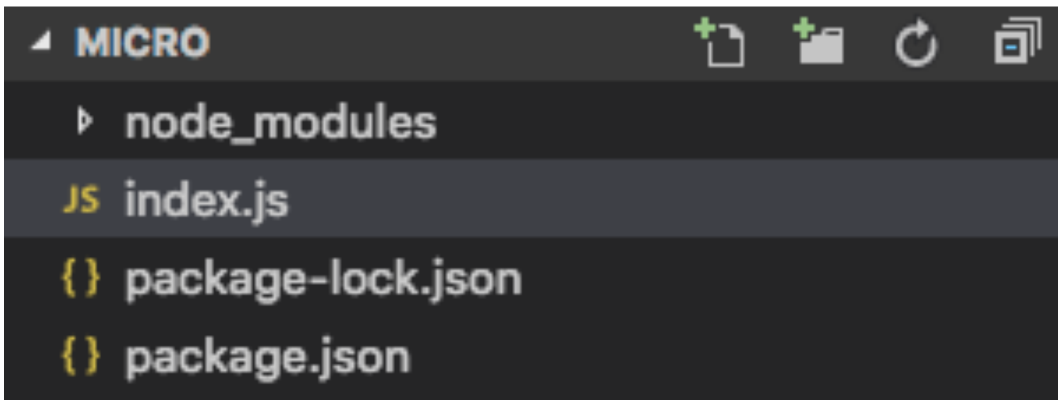

Рисунок 3.1 – Структура проекту на першому етапі

Напишемо наступний код у **index.js**(рисунок 3.2)

```
index.js ×
1  const http = require('http')
2  const request = require('request');
3
4  const port = 3000;
5
6  const requestHandler = (req, res) => {
7    request.get(
8      'http://127.0.0.1:3001',
9      function (error, response, body) {
10       if (!error && response.statusCode == 200) {
11         res.end('Response from server 1;\n' + body);
12       }
13     }
14   );
15 }
16
17 const server = http.createServer(requestHandler);
18
19 server.listen(port, (err) => {
20   if (err) {
21     return console.log('something bad happened', err);
22   }
23   console.log(`server is listening on ${port}`);
24 })
```

Рисунок 3.2 – Код мікросервісу

Після цього треба запустити наш сервер командою

node index.js

Та перейти на localhost:3000

і ми побачимо наступне(рисунок 3.3)

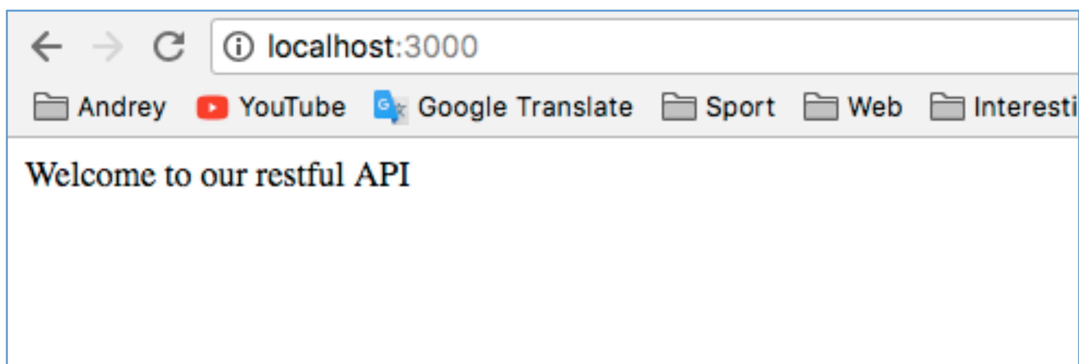

Рисунок 3.3 – Веб-сторінка серверу

Наш мікросервіс готовий для використання. Аналогічно розробимо інші два мікросервіси. Ми будемо робити запит до першого сервісу, який будет робити запит до другого, який у свою чергу буде робити запит до 3 мікросервісу(Рисунок 3.3). Таким чином ми будемо симулювати потік даних.

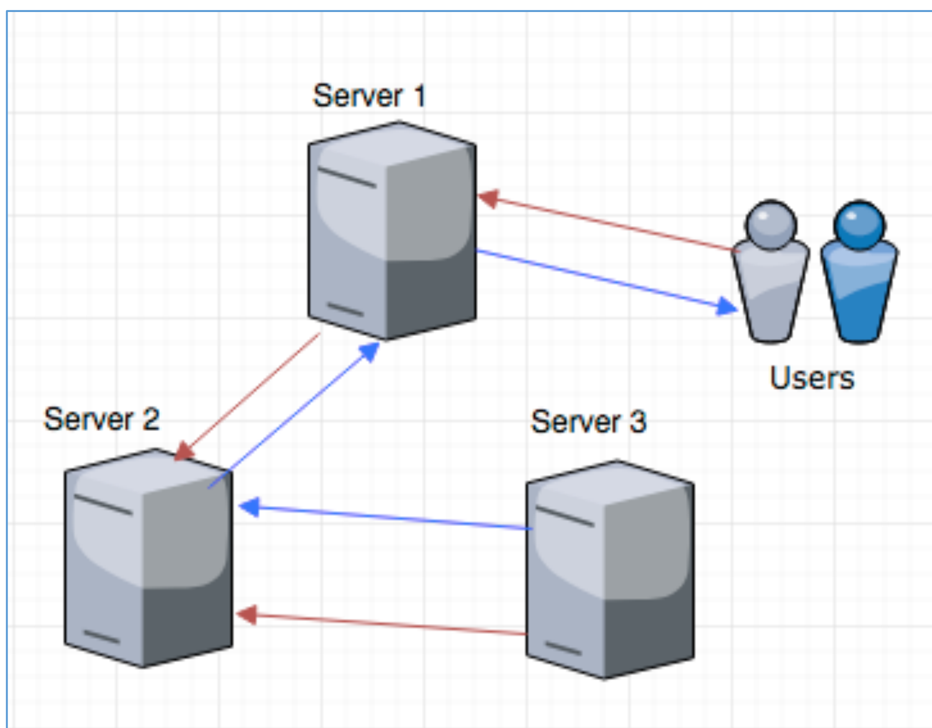

Рисунок 3.4 – Схема роботи мікросервісів

3.3 Контейнеризація мікросервісу

Треба створити Dockerfile та написати наступне:

FROM node: carbon

Створити директорію

WORKDIR / usr / src / app

Встановити залежності

COPY package * .json ./

RUN npm install

COPY. .

EXPOSE 3000

CMD ["npm", "start"]

Також треба додати .dockerignore файл

node_modules

npm-debug.log

Щоб створити образ треба написати наступну команду

docker build -t micro .

Після цього ви можете побачити свій образ завдяки команді

docker images

```
Andreys-Air:micro Andrey$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
micro               latest              f79930792e53        10 seconds ago      676MB
node                carbon              78f8aef50581        9 days ago          673MB
```

Рисунок 3.5 – Результат Docker images

Щоб запустити образ необхідно написати

docker run -p 49160:3000 -d andrey/node-web-app

Get container ID

$ docker ps

Print app output $ docker logs <container id>

Example Running on http://localhost:3000

Якщо треба побачити, що виконується у середині контейнера, то треба наступне:

$ docker exec -it <container id> /bin/bash

Тепер протестуємо REST

```
Andreys-Air:micro Andrey$ curl -i localhost:49160
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 26
ETag: W/"1a-/svn/Pv3s04uL2awf+ZrFpCPKa8"
Date: Tue, 15 May 2018 00:03:15 GMT
Connection: keep-alive

Welcome to our restful APIAndreys-Air:micro Andrey$
```

Рисунок 3.6 – Тестування за допомогою CURL

Як ми бачимо – усе працює.

Аналогічно розробимо образи для наших трьох мікросервісів. Назви будуть наступні: server1, server2, server3.

3.4 Налаштування Kubernetes для мікросервісу

Нам необхідно встановити cubectl для того щоб керувати нашими контейнерами

brew install cubectl

Також необхідно встановити MiniCube:

“curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64 && chmod + x minikube && sudo mv minikube / usr / local / bin /”

Після цього встановімо VirtualBox, або його драйвер. Запустимо virtual box драйвер:

“minikube start”

Потім треба використати конфіг:

“kubectl config use-context minikube”

Можна перевірити статус:

minikube dashboard

Потім треба запустити pod

“kubectl run hello-node --image=hello-node:v1 --port=8080”

Щоб залити нову версію мікросервісу треба використати наступну команду:

“docker build -t hello-node:v2 .”

“kubectl set image deployment/hello-node hello-node=hello-node:v2”

Та перезапустити сікросервіс:

“minikube service hello-node”

Отже ми дослідили можливості Kubernetes. У Kubernetes реалізовані всі функції, необхідні для запуску додатків на основі Docker в конфігурації з високою доступністю (кластери більше 1000 вузлів, з multi-availability і multi-region зонами): управління кластером, планування, виявлення сервісів, моніторинг, управління обліковими даними і багато інше.

Тепер ми будемо створювати pod, у якому об'єднаємо 3 контейнери.

Поперше треба сгенерувати pod конфіг-файл у якому треба описати необхідні параметри.

```
pod.json

{
  "kind": "Pod",
  "apiVersion": "v1",
  "metadata": {
    "name": "master-pod",
    "labels": {
      "app": "webapp"
    }
  },
  "spec": {
    "containers": [
      {
        "name": "server1",
        "image": "server1",
        "ports": [
          {
            "containerPort": 3000
          }
        ]
      },
      {
        "name": "server2",
        "image": "server2",
        "ports": [
          {
            "containerPort": 3001
          }
        ]
      },
      {
        "name": "server3",
        "image": "server3",
        "ports": [
          {
            "containerPort": 3002
          }
        ]
      }
    ]
  }
}
```

Рисунок 3.7 – Конфіг Pod-a

Після цього створюємо под командою

“kubectl create -f ./pod.json”

І ми можеме перевірити усі наші Pod-и командою

“kubectl get all”

Ми запустили наші три мікросервіси і можемо дуже легко ними керувати як одним цілим, так і окремо. Тепер коли ми робимо get запит на http://localhost:3000

то отримаємо у відповідь

Response from server 1;

Response from server 2;

Response from server 3;

Отже, ми отримали інформацію від усіх сервісів. Дослід пройшов успішно.

3.5 Розробка прикладу для dataflow

Використовуючи модель, яку ми побудували у розділі 3.2 зробимо наступний дослід. Ми хочемо, щоб нашими мікросервісами керували дані. Дані будемо передавати у вигляді JSON-файлу, так як NodeJS гарно їх оброблює. Зробимо невеликі допрацювання у наших мікросервісах. Спочатку ми налаштуємо наші мікросервіси, щоб вони могли спілкуватися між собою. Потім ми напишемо маленькі функції, які будуть симулювати дії мікросервісів. У server1 ми зробимо функцію add(x,y), яка буде складати числа, у server2 ми зробимо функцію multiply(x,y) – вона буде меремножувати два числа, та server3 яка буде додавати текст до отриманого числа.

```
JS index.js

1  function add(x, y) {
2    return x + y;
3  }
```

Рисунок 3.8 – Функція add(x,y)

```
JS index.js
1  function multiply(x, y) {
2    return x * y;
3  }
```

Рисунок 3.9 – Функція multiply(x,y)

```
JS index.js
1  function addText(n, text) {
2    return String(b) + ' ' + text;
3  }
```

Рисунок 3.10 – Функція addText(n, text)

Модель опишемо наступним чином(рисунок 3.11)

```
{} model.json ×
1  {
2    "route": ["server1", "server2", "server3"],
3    "initValue": 5,
4    "server1": {
5      "action": "add",
6      "params": [3]
7    },
8    "server2": {
9      "action": "multiply",
10     "params": [2]
11   },
12   "server3": {
13     "action": "addText",
14     "params": ["final result"]
15   }
16 }
```

Рисунок 3.11 – JSON модель

Route- масив, у якому вказано порядок(маршрут) для наших даних

InitValue – число, яке передається першим

Server1…3 – опис команд для кожного мікросервісу, де

Action – метод який буде використовуватися, params – параметри, які необхідні для використання методу(починаючи з 2-го).

Таким чином, ми отримаємо наступний результат:

1) Ми передамо число 5 у функцію add першого мікросервісу, і отримаємо результат 5+3 = 8.
2) Потім ми передамо 8 у другий мікросервіс 8*2 = 16
3) І далі, у 3-му ми зробимо строку з числа 16 та строки “final result”

У результаті виконання усіх операцій ми отримаємо строку: “16 final result”.

Ми можемо змінити route на такий route: [server2, server1, server3]. У результаті отримаємо “13 final result”. Отже, завдяки цьому у нас є гнучкість і ми можемо виконувати різні операції завдяки зміні однієї строки у нашій моделі.

3.6 Концепція “Один мікросервіс – один контейнер”

Service instance per container – це шаблон проектування мікросервісів.

При розробці продукту може виникнути питання “Скільки мікросервісів може вмістити один контейнер?”. Давайте розглянемо приклад:

Отже, якщо ми створимо образ контейнеру, який буде містити наші три мікросервіси у собі, то ми створимо собі багато проблем. Якщо у нас будуть мікросервіси підтримувати різні люди або команди, то ми не зможемо бути впевнені у правильності dockerfile, так як кожний мікросервіс може бути написаний на різній мові програмування. Потім, у нас може виникнути ситуація, коли навантаження на один із мікросервісів йде більше ніж на інші, і нам знадобиться розгорнути ще один інстанс цього сервісу, але у нашій ситуації у нас ще розгорнуться два надлишкових мікросервіси. Мікросервіси повинні бути незалежні та ізальовані, якщо в одному буде проблема з безпекою, то нам буде важко відмежити його від інших всередині одного

контейнеру. Також, у нас буде багато логів з цього контейнеру. І останнє, якщо щось зломається в одному мікросервісі при розгортанні, то у нас не розгорнуться інші два.

Тому ми можемо перерахувати такі переваги використання шаблону Service instance per container:

- Сервіси пишуть з використанням різних мов, рамок і версій основних засобів
- Сервіс повинен бути незалежно розгортаним та масштабованим
- Сервіси потрібно ізолювати один від одного
- Необхідність швидко створювати та розгортати сервіси
- Необхідність обмежувати ресурси (процесор та пам'ять), витрачені сервісом
- Необхідність стежити за поведінкою кожного екземпляра сервісу
- Необхідність у Fault Tolerance

Отже, при використанні цього шаблону ми бачимо велику кількість переваг і жодного недоліку.

Також проведено дослід, де ми вивчали залежність використання CPU та диску від того, який з цих способів ми використовуємо. Ми отримали наступні результати:

Таблиця 3.1 – Використання CPU and Disk під час різних підходів до контейнеризації

№ п/п	*Підхід контейнеризації*	*1 мікросервіс*	*5 мікросервісів*	*10 мікросервісів*
1.	Один мікросервіс – один контейнер	5% CPU	22% CPU	31% CPU
		691 MB	2.5 GB	4.1 GB
2.	Усі мікросервіси в одному контейнері	5% CPU	27%CPU	64% CPU
		689 MB	3.1 GB	7 GB

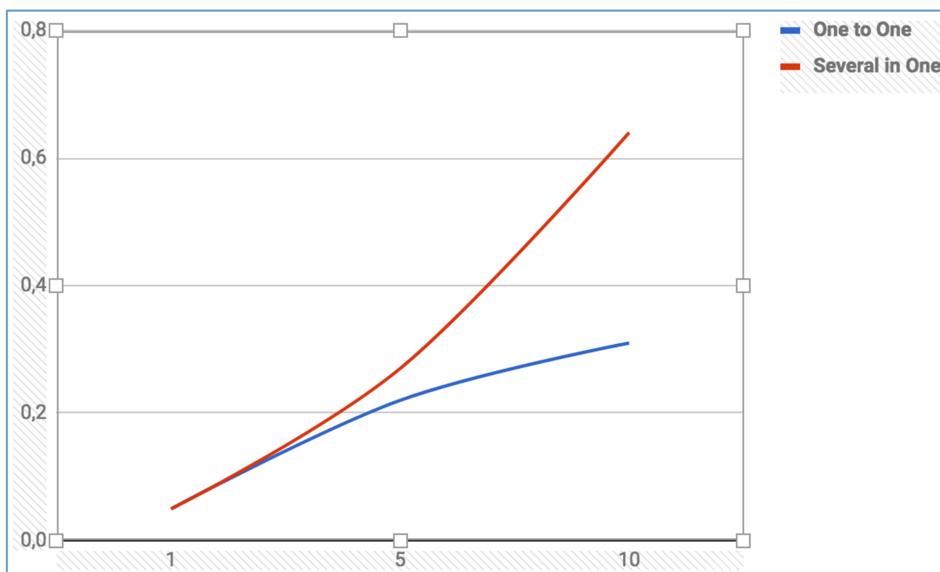

Рисунок 3.12 – Графік залежності між кількістю мікросервісів та використанням CPU

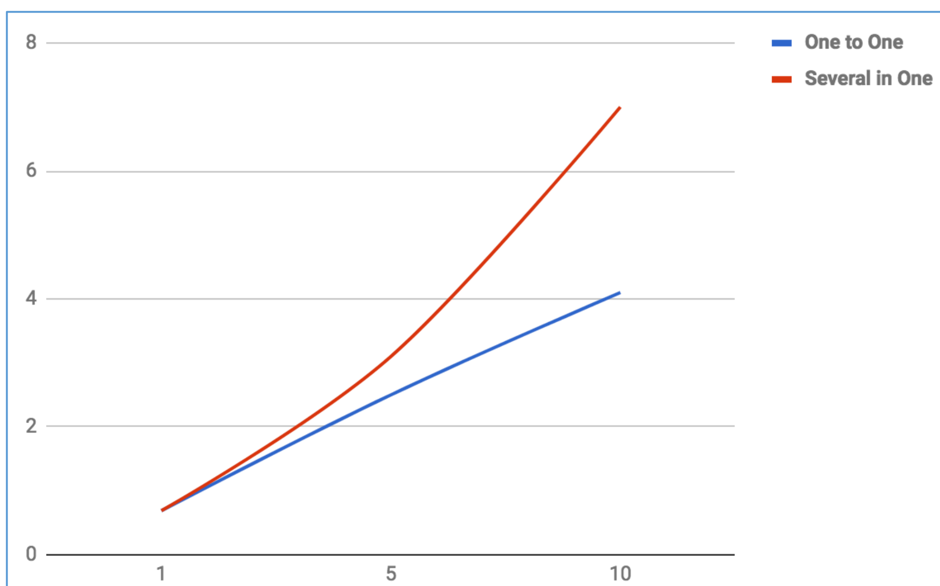

Рисунок 3.13 – Графік залежності між кількістью мікросервісів та використанням диску

Як ми бачимо з досліду, що шаблон “Один мікросервіс – один контейнер” виграє за економією ресурсів. У данному випадку ми бачимо, що є економія CPU та використання диску.

3.7 Використання контейнеризації в IoT

Сьогодні Інтернет речей став дуже популярним. У світі віддалених пристроїв, підключених до Інтернету, час простою особливо дорогий. На відміну від екземпляра хмари, якщо пристрій знижується, ви не можете просто обертати інший, щоб замінити його. Цей пристрій може бути дроном, машиною, розумним замком у чужий будинок або сенсорною станцією в нафтовому полі. Пристрої IoT часто фізично недоступні, тому ручне перезавантаження непросто. Контейнери дозволяють відновити, коли щось стає не так.

Як це працює? Це, по суті, полягає в тому, щоб відокремити основні операції пристрою від прикладного рівня, гарантуючи, що збій програми не вплине на здатність пристрою спілкуватися в мережі. Наприклад, всі пристрої resin.io запускають resinOS, операційну систему OS, яка включає в себе двигун контейнера Docker(рис 3.5).

Ця операційна система керує двома контейнерами: один - керуючий, агент, який забезпечує правильне функціонування пристрою, і може підключитися до resin.io, а інший запускає користувальницькі додатки, що входять до складу своєї базової ОС. Хост-ОС взаємодіє з апаратним сторожевим доглядом, забезпечуючи перезавантаження, якщо виникає проблема з програмним забезпеченням низького рівня. Зрештою, це робить щось вище, ніж цей рівень, проблемою для додатків, яку можна вирішити віддалено.

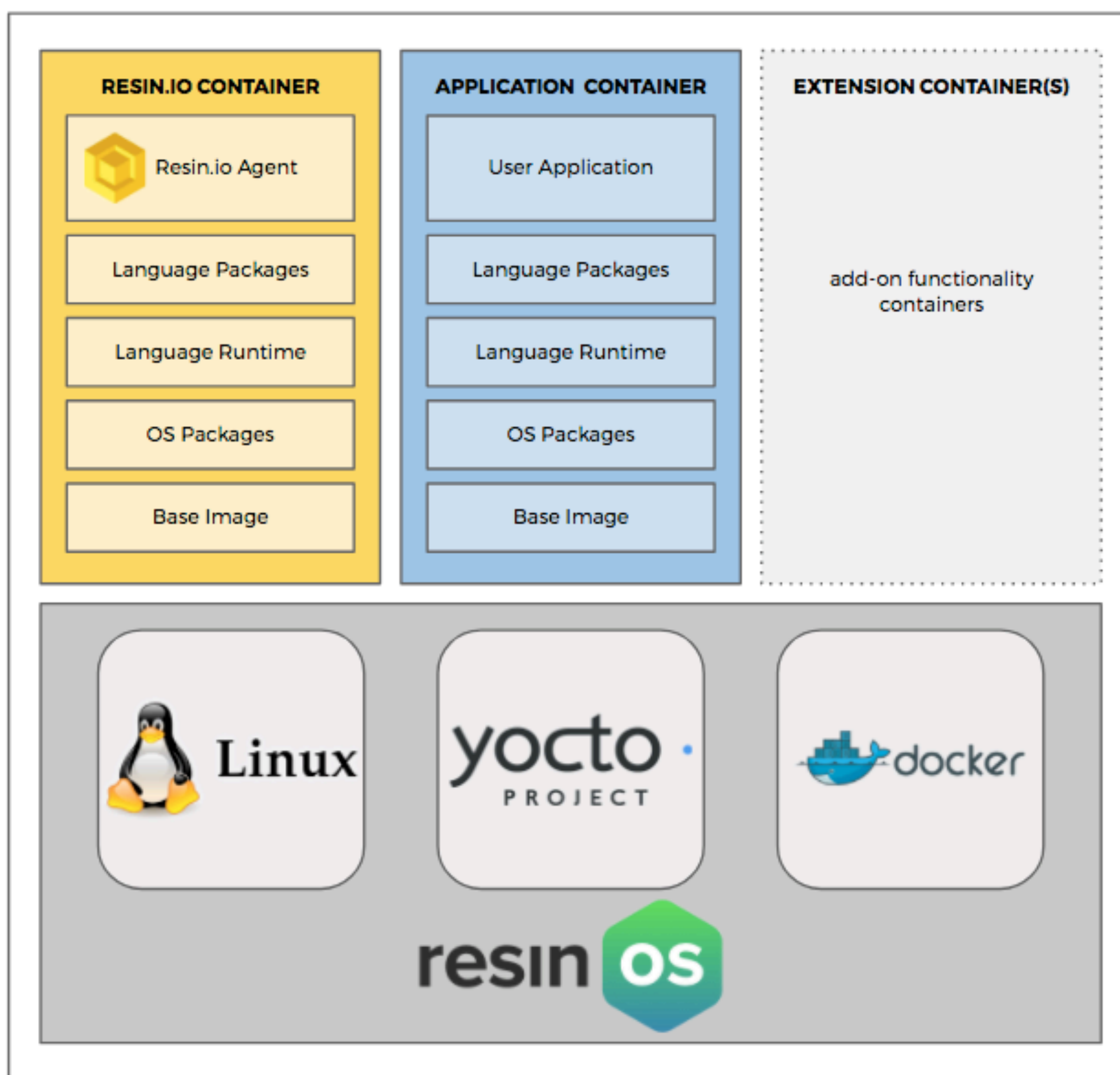

Рисунок 3.14 – Docker IoT[7]

Ще однією перевагою контейнерів є можливість краще керувати оновленнями, включаючи меншу частоту простоїв і зменшення використання простору на диску. Як приклад, давайте подивимося, як оновлення обробляються resinOS. Традиційний підхід до застосування оновлень із можливістю резервування - це стратегія розділу A / B. Це розділяє привід на два, причому половина залишається невикористаною. Оновлення можна завантажувати та встановлювати у порожній розділ без видалення активної ОС і без втрати зв'язку з мережею. Якщо виникають проблеми з переходом на оновлену операційну систему, пристрій можна перезавантажити за допомогою останньої робочої версії, що значно зменшить шанс, що він втратить для мережі. З resinOS більша частина того, що потрібно для запуску користувальницького додатку,

упаковується в контейнер Docker і може бути оновлено без будь-яких простоїв. Це зменшує частоту оновлень, необхідних для ОС хоста. Коли потрібно зробити оновлення для ОС хоста, стратегія розділів A / B все ще використовується, але мінімальний розмір операційної системи хоста дозволяє розділу оновлення бути набагато меншим.

Контейнери відіграють велику роль у подоланні розриву між хмарами та вбудованими робочими процесами. Linux - це широко використовувана і дуже настроювана операційна система, а контейнери Linux забезпечують стандартний набір функцій, одночасно надаючи розробникам свободу вибору інструментів, бібліотек та конфігурацій, з якими вони вже знайомі. Така гнучкість очікується у розробників хмари, а її розширення на вбудовані пристрої дає змогу більше розробникам створювати та підтримувати проекти IoT. Вирівнюючи базові технології через обласні та крайні пристрої, контейнери зменшують тертя для розробників та організацій, що підтримують гібридні робочі процеси.

Програмування на основі потоку, є способом опису поведінки програми як мережі чорних ящиків або “вузлів”, як вони називаються в Node-RED. Кожен вузол має чітко визначену мету; він дає деякі дані, він робить щось із цими даними, а потім передає ці дані. Мережа відповідає за потоки даних між вузлами.

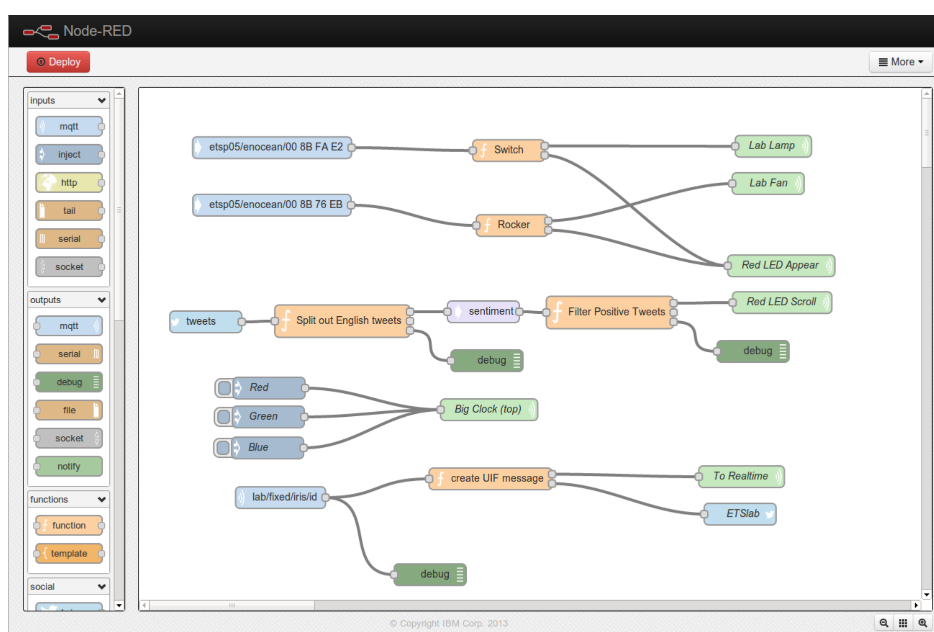

Рисунок 3.15 – Приклад системи IoT [8]

Це модель, яка дуже добре підходить для візуального представлення та робить її більш доступною для широкого кола користувачів. Якщо хтось може розбити про блему на окремих етапах, вони можуть подивитися на потік і отримати відчуття того що він робить, без необхідності розуміти окремі рядки коду всередині кожного вузла.

Node-RED складається з runtime на основі Node.js, до якої ви вказуєте веб-переглядач на, щоб отримати доступ до редактора потоку. В браузері ви створюєте свою програму, перетягуючи вузли з палітри в робочу область і починаючи з'єднувати їх. За допомогою одного кліка програма розгортається назад до часу виконання, де вона запускається.

Вибірку вузлів можна легко розширити, встановивши нові вузли, створені спільнотою, і створений вами потоки можна легко поділитися як файли JSON.

3.8 Висновки

Було досліджено використання Docker та Kubernetes у IoT для керування потоками даних.

Мінімальні апаратні ресурси. Багато пристроїв IoT не мають потужних обчислювальних та пам'ятних ресурсів. Тому їхня здатність обробляти оновлення програмного забезпечення обмежена. Контейнери можуть допомогти на цьому фронті, тому що встановлення нового зображення контейнера не вимагає великої обчислювальної потужності. Пристрій IoT потрібно просто завантажити зображення, помістити його в будь-який момент, де він буде жити, і видалити старе зображення.

Конфігурація обробки мінімальна. Географічний розподіл. У деяких випадках використання пристрої IoT поширюються на велику географічну область. Доставка програмного забезпечення до них з одного центрального сховища може не працювати добре. За допомогою Docker легко створити регістр зображень у кількох місцях, щоб добре обслуговувати всю мережу.

Обмежений або випадковий доступ до мережі. Незважаючи на наслідки терміна "Інтернет речей", не всі пристрої в Інтернеті речей добре підключені до Інтернету.

Вони можуть мати обмежену смугу пропускання в мережі або бути в Інтернеті лише періодично. Докер може допомогти забезпечити оновлення програмного забезпечення за таких обставин, оскільки, коли зображення контейнера оновлюються, Docker завантажує лише ті частини зображення, які були змінені. (Resin.io враховує цю ефективність ще на крок за допомогою дельта-оновлень, які дозволяють ще більш вибірково завантажувати процес під час оновлення зображення контейнера.)

Широко різняться середовища пристрою. Програмне забезпечення, яке працює на пристрої IoT, може бути майже що завгодно. Різновидність конфігурацій програмного забезпечення на пристроях IoT зазвичай ускладнює встановлення програм, тому що додатки повинні бути налаштовані для кожного типу середовища, якщо вони встановлені за допомогою традиційних методів. Однак з контейнерами версія операційної системи та інші змінні програмного забезпечення набагато менш важливі. Поки пристрій виконує якийсь дистрибутив Linux і має час виконання контейнера, ви можете встановити контейнерні додатки на ньому без спеціальної конфігурації.

4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

4.1 Опис ідеї проекту

Розділ має на меті проведення маркетингового аналізу стартап проекту “Контейнеризація у системах керування потоками даних” задля визначення принципової можливості його ринкового впровадження та можливих напрямів реалізації цього впровадження.

Метою розділу є формування інноваційного мислення, підприємницького духу та формування здатностей щодо оцінювання ринкових перспектив і можливостей комерціалізації основних науково-технічних розробок, сформованих у попередній частині магістерської дисертації у вигляді розроблення концепції стартап-проекту “Контейнеризація у системах керування потоками даних” в умовах висококонкурентної ринкової економіки глобалізаційних процесів.

Опис стартап-проекту “Контейнеризація у системах керування потоками даних” наведено у Таблиці 4.1.

Таблиця 4.1. Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
Створення та розгортання системи контейнеризації у хмарі з наданням користувачеві доступу через REST api. Розробити сервіс який зможе автоматично розгортати систему на будь-якіій операційній системі, у тому числі і IoT ОС.	1. Використання для виконання розгортки, масштабування, аварійного перезапуску системи.	REST API надає можливість використовувати систему як сервіс, що значно спрощує доступ до системи. Автоматична розгортка системи, автоматичне масштабування та аварійний перезапуск мікросервісів.
	2. Використання для IoT-пристроїв та систем.	Можливість використання системи для IoT-приладів завдяки контейнеризації.

Отже, проект “Контейнеризація у системах керування потоками даних” може бути використаним як інструментом для повного розгортання будь-якої системи, так і прошарком для розгортання одного маленького мікросервісу в IoT-приладі.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	*Техніко-економічні характерис-тики ідеї*	*(потенційні) товари/концепції конкурентів*				*W (слабка сторона)*	*N (нейтра-льна сторона)*	*S (сильна сторона)*
		Мій проект	*Конкурент 1*	*Конкурент2*	*Конкурент 3*			
1.	Форма виконання	Веб-сервіс	Програма	Веб-додаток	Програма			+
2.	Собівартість	Низька	Низька	Висока	Висока			+
3.	Кросплатформність	Так	Ні	Так	Так			+
4.	Наявність контейнеризації	Так	Так	Так	Так		+	
5.	Застосування Docker	Так	Ні	Так	Так			+
6.	Горизонтальне масштабування	Так	Так	Ні	Так			+

Сильними сторонами проекту є форма виконання у вигляді веб-сервісу, низька собівартість, кросплатформність. Наявність Docker надає переваги для використання сторонніх бібліотек. Також сильною стороною є можливість горизонтального масштабування. Нейтральна сторона - використання контейнеризації. Слабкі сторони – відсутні. Отож, система є конкурентноспроможною.

4.2 Технологічний аудит ідеї проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару).

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

№ п/п	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1.	Створення контейнерів	Docker	Наявна	Безкоштовна, доступна
		Kubernetes	Наявна	Безкоштовна, доступна
2.	Створення REST API для доступу до бази знань	ExpressJS, NodeJS	Наявні	Безкоштовна, доступна
		Amazon API Gateway, AWS CDN	Наявна	Платні
3.	Хмарне розгортання додатку	Heroku	Наявна	Безкоштовна, доступна
		AWS EC2, S3	Наявна	Платні

Обрані технології реалізації ідеї проекту: NodeJS, ExpressJS через повну безкоштовність фреймворку та наявність докладної документації, наявність досвіду роботи розробників з даною технологією; Docker та Kubernetes через простоту використання, безкоштовність та можливість розгортання додатків на основі таких технологій у хмарі; Heroku для розгортання у хмарі через безкоштовність.

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1.	Кількість головних гравців, од	4
2.	Загальний обсяг продаж, грн/ум.од	12000 грн./ум.од
3.	Динаміка ринку (якісна оцінка)	Зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	Немає
5.	Специфічні вимоги до стандартизації та сертифікації	Немає
6.	Середня норма рентабельності в галузі (або по ринку), %	R = (3000000 * 100) / (1000000 * 12) = 25%

Отже, було проаналізовано наявність попиту, обсяг, динаміку розвитку ринку. Обмеження для входу на ринок відсутні, динаміка ринку зростає, галузь є рентабельною.

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (табл. 5).

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1.	Необхідне програмне забезпечення(REST API) для доступу до контейнерів	Потенційними цільовими групами є дослідницькі центри, університети та компанії, специфіка роботи яких потребує автоматичної розгортки ПЗ та його автоматичного масштабування.	Цільова група займається розробкою ПЗ та його розгорткою у хмарах.	Рішення повинне бути придатним до інтеграції в інші більш складні системи, бути здатним надавати автоматичне налаштування, бути розгорнутим у хмарі

Згідно проведеної характеристики потенційних клієнтів стартап-проекту випливає, що на ринку є затребуваним програмне забезпечення(REST API) для доступу до контейнерів і потенційними цільовими групами є дослідницькі центри, університети та компанії, специфіка роботи яких потребує автоматичної розгортки ПЗ та його автоматичного масштабування.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. № 6-7). Фактори в таблиці подавати в порядку зменшення значущості.

Таблиця 4.6 – Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1.	Конкуренція	Вихід на ринок великої компанії	1. Вихід з ринку 2. Запропонувати великій компанії поглинути себе 3. Передбачити додаткові переваги власного ПЗ для того, щоб повідомити про них саме після виходу міжнародної компанії на ринок
2.	Зміна потреб користувачів	Користувачам необхідне програмне забезпечення з іншим функціоналом	1. Передбачити можливість додавання нового функціоналу до створюваного ПЗ
3.	Зміна тарифів провайдера хмарного розгортання на платні	Необхідність оплати послуг провайдера хмари	1. Пошук іншого безкоштовного провайдера 2. Пошук інвестицій для оплати існуючого провайдера
4.	Надходження на ринок альтернативних продуктів	Перехід користувачів нашого товару на інший продукт	Впровадження нового функціоналу, якого немає у конкурентів
5.	Уповільнення росту ринку	Скорочення користувачів продуктів, що тільки виходять на ринок	Інвестиції у впровадження ефективної реклами продукту

Отже, було проаналізовано фактори загроз ринкового впровадження проекту, серед яких: конкуренція, уповільнення росту ринку, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні та надходження на ринок альтернативних продуктів. Було також запропоновано можливі реакції компанії.

Таблиця 4.7 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1.	Стрімкий ріст попиту на інструменти обробки даних у вигляді онтологій та RDF-графів	Наявність попиту на інструменти для обробки даних у вигляді онтологій та RDF-графів	Змога запропонувати продукт більшій кількості потенційних користувачів
2.	Поява нових різонерів	Надання нового функціоналу для надання результатів роботи нового логічного виведення	Розробка нового функціоналу у вигляді нового HTTP запиту для надання користувачам результатів логічного виведення
3.	Стрімке зростання росту ринку	Компаніям, що тільки виходять на ринок, буде простіше отримати клієнтів	Змога запропонувати продукт більшій кількості потенційних користувачів
4.	Обслуговування додаткових груп споживачів	Поява нових потенційних груп споживачів	Змога розширити продукт для подальшого впровадження у нові галузі
5.	Розширення асортименту можливих послуг	Поява нового функціоналу, що приваблить нових користувачів	Розробка нового функціоналу, що є потребою певної групи користувачів

У Таблиці 4.7 наведено фактори можливостей ринкового впровадження проекту, серед яких: стрімкий ріст попиту на інструменти обробки даних у вигляді онтологій та RDF-графів, поява нових різонерів, стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг; було також запропоновано можливі реакції компанії.

Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку (табл. 8).

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Вказати тип конкуренції - досконала	Існує 3 фірми-конкурентки на ринку	Врахувати ціни конкурентних компаній на початкових етапах створення бізнесу, реклама (вказати на конкретні переваги перед конкурентами)
2. За рівнем конкурентної боротьби - міжнародний	Всі компанії – з інших країн.	Додати можливість вибору мови ПЗ, щоб легше було у майбутньому вийти на міжнародний ринок
3. За галузевою ознакою - внутрішньогалузева	Конкуренти мають ПЗ, яке використовується лише всередині даної галузі	Створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях
4. Конкуренція за видами товарів: - товарно-видова	Види товарів є однаковими, а саме – програмне забезпечення	Створити ПЗ, враховуючи недоліки конкурентів
5. За характером конкурентних переваг - нецінова	Вдосконалення технології створення ПЗ, щоб собівартість була нижчою	Використання менш дорогих технологій для розробки, ніж використовують конкуренти
6. За інтенсивністю - не марочна	Бренди відсутні	-

У Таблиці 4.8 наведено ступеневий аналіз конкуренції на ринку, де було визначено особливості конкурентного середовища та їх вплив а діяльність підприємства. Однією з найбільш важливих дій компанії для досягнення конкурентоспроможності є необхідність створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (Табл. 9).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	*Прямі конкуренти в галузі*	*Потенційні конкуренти*	*Постачальники*	*Клієнти*	*Товари-замінники*
	Навести перелік прямих конкурентів	*Визначити бар'єри входження в ринок*	*Визначити фактори сили постачальників*	*Визначити фактори сили споживачі в*	*Фактори загроз з боку замінників*
Висновки:	Існує 3 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 2, так як його рішення також представлене у вигляді веб-сервісу	Так, можливості для входу на ринок є, бо наше рішення має дуже зручний Middleware та REST API	Постачальники відсутні.	Важливим для користувача є швидкість та надійність роботи ПЗ	Товари-замінники можуть використати більш дешеву технологію створення ПЗ та зменшити собівартість товару

Було здійснено аналіз конкуренції в галузі за М. Портером, в результаті чого було визначено, що існує 3 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 2, так як його рішення також представлене у вигляді веб-сервісу, але можливості для входу на ринок є, бо наше рішення має зручний Middleware та REST API.

За результатами аналізу таблиці робиться висновок щодо принципової можливості роботи на ринку з огляду на конкурентну ситуацію. Також робиться висновок щодо характеристик (сильних сторін), які повинен мати проект, щоб бути конкурентоспроможним на ринку. Другий висновок враховується при формулюванні переліку факторів конкурентоспромож- ності у п. 3.6. 3.6) На основі аналізу конкуренції, проведеного в п. 3.5 (табл. 9), а та- кож із урахуванням характеристик ідеї проекту (табл. 2), вимог споживачів до товару (табл. 5) та факторів маркетингового середовища (табл. № 6-7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлюється за табл. 10

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1.	Наявність Middleware	Дозволяє користувачам здійснювати гнучкий доступ до мікросервісів
2.	Наявність REST API	Дозволяє інтегрувати сервіс у складні системи завдяки універсальному API
3.	Хмарне розгортання	Дозволяє звертатись до ПЗ як до сервісу
4.	Горизонтальне масштабування	Можливість гнучкого масштабування за допомогою додавання апаратних компонентів

У Таблиці 7 наведено обґрунтування факторів конкурентоспроможності, серед яких: наявність Middleware для гнучкого доступу до мікросервісів, наявність REST API та хмарне розгортання. Було також наведено обґрунтування цих факторів.

За визначеними факторами конкурентоспроможності (табл. 10) проводиться аналіз сильних та слабких сторін стартап-проекту (табл. 11).

У наступній таблиці наведено проведення аналізу сильних та слабких сторін стартап-проекту, факторами конкурентоспроможності виступили такі: наявність наявність Middleware для гнучкого доступу до мікросервісів, наявність REST API, хмарне розгортання, горизонтальне масштабування.

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з нашим підприємством*						
			-3	-2	-1	0	1	2	3
1.	Наявність Middleware	20	+						
2.	Наявність REST API	15			+				
3.	Хмарне розгортання	15				+			
4.	Горизонтальне масштабування	10				+			

Отже, серед сильних сторін проекту можна виділити наступні: наявність Middleware для гнучкого доступу до мікросервісів, наявність REST API. Серед нейтральних сторін можна виділити можливість хмарного розгортання та можливості горизонтального масштабування.

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити 103 прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

У наступній таблиці буде проілюстровано SWOT-аналіз стартап-проекту, тобто його слабкі та сильні сторони, можливості та загрози виходу на ринок.

Таблиця 12. SWOT- аналіз стартап-проекту

Сильні сторони: наявність Middleware для гнучкого доступу до мікросервісів, наявність REST API, можливість хмарного розгортання	Слабкі сторони: можливість зміни тарифів провайдером хмарного розгортання на платні, відсутність можливості горизонтального масштабування
Можливості: стрімкий ріст попиту на інструменти автоматичної розгортки, можливість впровадження нових контейнерів, стрімке зростання росту ринку, обслуговування додаткових груп споживачів,розширення асортименту можливих послуг	Загрози: конкуренція, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні, надходження на ринок альтернативних продуктів, уповільнення росту ринку

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів.

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Створення додатку з використанням NodeJS та ExpressJS	90%	3 місяці
2.	Створення програми на основі без використання будь- яких фреймворків	35%	1.5 роки

З означених альтернатив обирається та, для якої: а) отримання ресурсів є більш простим та ймовірним; б) строки реалізації – більш стислими. Тому обираємо альтернативу (створення додатку з використанням NodeJS та ExpressJS).

4.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 14).

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів.

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1.	Дослідницькі центри	Спрощення роботи з автоматичною розгорткою та масштабуванням	Великий	Існує 3 конкуренти, які надають схожі, але більш вузькі і дорогі рішення.	Наявність REST API, Middleware, інтеграція з будь-якими контейнерами
2.	Підприємства	Спрощення роботи з автоматичною розгорткою та масштабуванням	Великий		Можливість інтеграції в уже існуючі системи завдяки REST API, зручне хмарне розгортання, наявність Middleware
Які цільові групи обрано: обираємо підприємства та дослідницькі центри					

За результатами аналізу потенційних груп споживачів (сегментів) автори ідеї обирають цільові групи, для яких вони пропонуватимуть свій товар, та визначають стратегію охоплення ринку. Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку. За М. Портером, існують три базові стратегії розвитку, що відрізняються за ступенем охоплення цільового ринку та типом

конкурентної переваги, що має бути реалізована на ринку (за витратами або визначними якостями товару). Отже, проілюструвати базову стратегію розвитку можна у виляді Таблиці 4.15

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспроможні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку*
1.	Створення веб-сервісу, використовуючи NodeJS, ExpressJS	Ринкове позиціонування	Можливість інтеграції в уже існуючі системи завдяки REST API, зручне хмарне розгортання, наявність Middleware	Диференціація

Було обрано таку альтернативу розвитку проекту: створення веб-сервісу, використовуючи NodeJS та ExpressJS, адже завдяки цим технологіям можна досягнути ключових конкурентноспроможних позицій кінцевого продукту.

Таблиця 4.16 - Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки*
1.	Ні	Так	Буде, а саме: основною задачею є розробка ПЗ з використанням Docker-контейнерів(конкуренти 1, 2, 3), форма виконання - веб-сервіс (конкурент 2)	Зайняття конкурентної ніші

Отже, було визначено базову стратегію конкурентної поведінки як зайняття конкурентної ніші.

Визначимо стратегію позиціонування у Таблиці 4.17, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17 - Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегі я розвитку*	*Ключові конкурентоспроможні позиції власного стартап- проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)*
1.	Наявність універсального API, зручне хмарне розгортання, Middleware	Диференціація	Можливість інтеграції в уже існуючі системи завдяки REST API, зручне хмарне розгортання, наявність Middleware	Інтеграція, хмарне розгортання, Middleware

Отже, було вибрано такі асоціації, які мають сформувати комплексну позицію власного проекту:інтеграція (адже завдяки REST API сервіс просто інтегрувати у існуючі системи), хмарне розгортання (оскільки ExpressJS додатки легко розгортаються в хмарах), Middleware (у системі є повноцінний Middleware).

4.5 Розробка маркетингової програми

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. 4.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18 - Визначення ключових переваг концепції потенційного товару

№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)*
1.	Наявність універсального API	Додаток реалізований у вигляді RESTful сервісу, що надає відповіді у вигляді JSON, що дає змогу користувачам звертатись до сервісу за допомогою стандартних HTTP POST та GET запитів	Перевага в універсальності на можливості інтегрувати сервіс у існуючі системи.
2.	Можливість зручного хмарного розгортання	Можливість розгорнути додаток всюди, де є функціонал розгортання Spring Boot додатків, а така можливість є у більшості хмарних провайдерів	Користувачі мають змогу працювати з системою віддалено у хмарі
3.	Наявність Middleware	Можливість виконання будь-яких Middleware функцій	Гнучкість веб-сервісу

Отже бачимо, що проект має ключові переваги перед конкурентами, які повністю відповідають потребам цільової аудиторії. Додаток реалізований у вигляді RESTful сервісу, що надає відповіді у вигляді JSON, що дає змогу користувачам звертатись до сервісу за допомогою стандартних HTTP POST та GET запитів, а це є досить універсальним способом для подальшої інтеграції сервісу в інші системи.

Далі у Таблиці 4.19 проілюстрована трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання.

Таблиця 4.19 - Опис трьох рівнів моделі товару

<table>
<tr><th>Рівні товару</th><th colspan="3">Сутність та складові</th></tr>
<tr><td>І. Товар за задумом</td><td colspan="3">Веб-сервіс, що надає доступ до контейнерів мікросервісів за допомогою HTTP запитів, дозволяє працювати зі Middleware та надає можливості хмарного розгортання</td></tr>
<tr><td rowspan="5">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Наявність універсального API
2. Можливість зручного хмарного розгортання
3. Наявність Middleware</td><td>1.Нм
2.Нм
3.Нм</td><td>1.Технологічна
2.Технологічна
3.Технологічна</td></tr>
<tr><td colspan="3">Якість: згідно до стандарту ISO 4444 буде проведено тестування</td></tr>
<tr><td colspan="3">Маркування відсутнє</td></tr>
<tr><td colspan="3">Моя компанія: “Containerization”</td></tr>
<tr><td rowspan="2">III. Товар із підкріплюнням</td><td colspan="3">1-місячна пробна безкоштовна версія</td></tr>
<tr><td colspan="3">Постійна підтримка для користувачів</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: патент</td></tr>
</table>

Було описано три рівні моделі товару, з чого можна зробити висновок, що основні властивості товару у реальному виконанні є нематеріальними та технологічними. Також було надано сутність та складові товару у задумці та товару з підкріпленням.

Після формування маркетингової моделі товару слід особливо відмітити – чим саме проект буде захищено від копіювання. У даному випадку найбільш вірогідним гарантом буде патент.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-

аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (табл. 4.20). Аналіз проводиться експертним методом.

Таблиця 4.20 - Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники, грн.*	*Рівень цін на товари-аналоги, грн.*	*Рівень доходів цільової групи споживачів, грн.*	*Верхня та нижня межі встановлення ціни на товар/послугу, грн.*
1.	15000	8000	150000	6000-20000

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 4.21).

Таблиця 4.21 - Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1.	Придбання підписки та оплата щомісячних внесків для продовження ліцензії	Продаж	0(напряму), 1(через одного посередника)	Власна та через посередників

Отже, система приноситиме прибуток завдяки щомісячним внескам для подовження ліцензії та придбанням підписок, продаж будк виконуватись напряму або через одного посередника.

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 4.22).

Таблиця 4.22 - Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуютьс я цільові клієнти*	*Ключові позиції, обрані для позиціону вання*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
1.	Придбання ліцензії на користування в мережі Інтернет, щомісячне її продовження, користування сервісом у хмарі або ж на власних серверах.	Інтернет	Інтеграція, хмарне розгортання, Middleware	Показати переваги сервісу, у тому числі і перед конкурентами	Демо-ролик і використання, рекламні оголошення н популярних сайтах.

Отже, в Таблиці 4.22 наведено концепцію маркетингових комунікацій, було визначено, що придбання ліцензії на користування буде здійснюватись в мережі Інтернет, необхідним буде щомісячне її продовження, користування сервісом можливе у хмарі або ж на власних серверах.

4.6 Висновки

Згідно до проведених досліджень існує можливість ринкової комерціалізації проекту. Також, варто відмітити, що існують перспективи впровадження з огляду на потенційні групи клієнтів, бар‘єри входження не є високими, а проект має дві значні переваги перед конкурентами. Для успішного виконання проекту необхідно реалізувати програму із використанням засобів NodeJS та ExpressJS. Для успішного виходу на ринок у продукту повинні бути наступні характеристики:

- наявність універсального API
- можливість зручного хмарного розгортання
- наявність Middleware

В рамках даного дослідження були розраховані основні фінансово-економічні показники проекту, а також проведений менеджмент потенційних ризиків. Проаналізувавши отримані результати, можна зробити висновок, що подальша імплементація є доцільною.

Було визначено такі сильні сторони: наявність Middleware для гнучкого доступу до мікросервісів, наявність REST API, можливість хмарного розгортання, горизонтальне масштабування. Серед нейтральних сторін можна виділити можливість зміни тарифів провайдером хмарного розгортання на платні.

Можливості для виходу на ринок включають стрімкий ріст попиту на інструменти автоматичної розгортки Docker-контейнерів, стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг. Наявні такі фактори загроз: конкуренція, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні, надходження на ринок альтернативних продуктів, уповільнення росту ринку.

ВИСНОВКИ

В першому розділі в процесі порівняння віртуалізації та контейнерезації було виявлено як недоліки так і переваги контейнерів. Недоліком контейнерів стало питання безпеки, конфіденційності, перевагою те, що вони займають менше місця на системному диску. Тим часом віртуалізація більш підходить до ізольованих систем. При запуску максимальної кількості додатків на мінімальній кількості серверів, все ж таки краще використовувати контейнери.

Не зважаючи на велику кількість переваг віртуальних машин, більшість з нас все ж таки використовують контейнери. Адже не можна ігнорувати можливість використання менших ресурсів.

Порівнянням сучасних контейнерних систем, стало зрозуміло, що найпопулярнішою контейнерною технологією, є - Docker. За остані роки попит на неї значно збільшився, адже у всьому світі збільшились розробки та розгортання додатків.

У другому розділі виявленно, що користувальницький інтерфейс містить у собі значну кількість команд. Основний файл налаштування dockerfile маючи багато інструкцій, щодо виконання налаштувань, витрачає багато зусиль кля команд терміналу. Зручності додає великі, навіть, обширні функціональний можливості, що дозволяють без зайвих зусиль налаштовувати, відлагоджувати та використовувати систему.

В третьому розділі були продемонстровані кроки зі встановлення, налаштування та запуску контейнерів в системі Docker та Kubernetes. Зокрема:

- Встановлення Docker на Mac OS
- Налаштування NodeJS мікросервіу
- Встановлення та перший запуск Docker на Mac OS
- Налаштування Docker для запуску контейнерів NodeJS додатку.
- Створення образу
- Запуск контейнера з різними налаштуваннями

- Налаштування Kubernetes
- Керування мікросервісом
- Дослідження використання Docker в IoT

Матеріал третього розділу є досить детальною інструкцією з використання можливостей системи, зі всіма кроками, командами та їх тонкощами, який може бути досить корисним при вивчені Docker та Kubernetes.

В останньому – четвертому розділі, був розроблен стартап-проект. Було проаналізовано технології, конкуренти, час необхідний для виходу на ринок. У результаті було отримано позитивний результат.

ПЕРЕЛІК ПОСИЛАНЬ

1. Сайт Вікіпедія. — Режим доступа : https://goo.gl/1EoZk2. — Дата доступа : 25.04.18.
2. Сайт Docker. — Режим доступа : https://uk.wikipedia.org/wiki/Docker. — Дата доступа : 27.04.18.
3. Сайт docker.com — Режим доступа : docker.com. — Дата доступа : 27.04.18.
4. Сайт bmstu. — Режим доступа : https://goo.gl/Cj3zYE. — Дата доступа : 29.04.18.
5. Магістерська дисертація СП. — Режим доступа : http://cad.kpi.ua/attachments/093_2017d_Melnychuk.pdf. — Дата доступа : 15.04.18.
6. Сайт kubernetes. — Режим доступа : kubernetes.com. — Дата доступа : 30.04.18.
7. Сайт resin.io. — Режим доступа : resin.io. — Дата доступа : 30.04.18.
8. Сайт node-red— Режим доступа: https://goo.gl/SigQhB.
9. Загороднюк А.О. “Архитектура Docker” - Молодой исследователь: вызовы и перспективы: сб. ст. по материалам LXIX междунар. науч.-практ. конф. — № 16(69). — М., Изд. «Интернаука», 2018. – 482 с.
10. Сайт bistu. — Режим доступа : https://goo.gl/Cj2zYE. — Дата доступа : 22.04.18.
11. Сайт kubernetes. — Режим доступа : kubernetes.com. — Дата доступа : 05.05.18.
12. Сайт docker.com — Режим доступа : docker.com. — Дата доступа : 27.04.18.
13. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. Погружение в мир нейронных сетей. 978-5-496-02536-2 изд. СПб. Питер: ООО Издательство "Питер", 2018. 480 с.
14. Nikhil Buduma Fundamentals of Deep Learning. Designing next-generation machine intelligence algorithms. 978-1-491-92561-4 O’Reilly Media, 2017. 277 с.
15. Francois Chollet Deep Learning with Python. 978-1617294433 изд. Manning Publications, December 22, 2017. 384 с.

ДОДАТОК А

ТЕКСТ ПРОГРАМИ СЕРВЕРУ

```
const http = require('http')
const request = require('request');
const port = 3000;
const requestHandler = (req, res) => {
  request.get(
    'http://127.0.0.1:3001',
    function (error, response, body) {
        if (!error && response.statusCode == 200) {
            res.end('Response from server 1;\n' + body);
        }
    }
  );
}
const server = http.createServer(requestHandler);
server.listen(port, (err) => {
  if (err) {
      return console.log('something bad happened', err);
  }
  console.log(`server is listening on ${port}`);
})
```

ДОДАТОК Б

ТЕКСТ КОНФІГУ DOCKER

```
FROM node:carbon

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm install --only=production

# Bundle app source
COPY . .

EXPOSE 8080
CMD [ "npm", "start" ]
```