DOI 10.34185/1562-9945-6-143-2022-09
УДК 004.75

К.Ю. Островська, М.О. Шерстяних, І.В. Стовпченко, Ю.О. Каліберда

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПЛАТФОРМ УПРАВЛІННЯ ОБЧИСЛЮВАЛЬНИМИ СЕРВІСАМИ ПРИ ОРГАНІЗАЦІЇ FOG COMPUTING

Анотація. Робота присвячена дослідженню ефективності платформ управління обчислювальними сервісами при організації Fog computing.
В рамках роботи провадиться дослідження ефективності платформ контейнерної оркестрації з організацією Fog computing.
У ході проведення дослідження необхідно виконати такі завдання: 1) зробити підбір літератури, наукових публікацій та Інтернет статей, необхідних для проведення дослідження; 2) здійснити огляд платформ контейнерної оркестрації; 3) визначити ключові вимоги та критерії проведення дослідження; 4) спроектувати та реалізувати утиліту автоматичного проведення випробувань; 5) виконати дослідження ефективності платформ контейнерної оркестрації з організацією туманних обчислень; 6) проаналізувати отримані результати та зробити супутні висновки.
Організується розгортання Docker контейнерів. Для створення кластера використовується Docker Swarm. Вирішуються завдання вимірювання наступних параметрів: час розгортання одного контейнера, час розгортання групи контейнерів, час відгуку задачі горизонтального масштабування, час затримки передачі. Проводиться аналіз одержаних результатів випробувань.
Ключові слова: fog computing, утиліта, docker, docker контейнер, docker swarm, горизонтальне масштабування, контейнер, платформа, інтернет речей.

Актуальність та постановка проблеми. На сьогоднішній день, коли технології інтернету речей та надання хмарних обчислень як послуг впевнено знаходять своє застосування, розвиваються нові обчислювальні концепції, зокрема технології туманних обчислень. Даний вид обчислень досі є новою технологією, яка тільки починає набирати свою популярність.

Парадигма хмарних обчислень сприяла розвитку надання обчислювальних сервісів та інфраструктур як послуг, що дозволило заощадити кошти на створення та обслуговування власної обчислювальної інфраструктури. У цьому середовищі спрощено та автоматизовано масштабування додатків для задово-

© Островська К.Ю., Шерстяних М.О., Стовпченко І.В., Каліберда Ю.О., 2022

лення потреб в умовах високого навантаження. Віртуалізація є ключовою технологією, що забезпечує ці можливості. В даний час контейнеризація стала популярною альтернативою віртуальним машинам і набула широкого застосування, у результаті інструменти оркестрації стали невід'ємною частиною хмарних обчислень. Незважаючи на успішне застосування технології контейнеризації, досі хмарні обчислення не забезпечують належного відповідності критеріям технологій Інтернету речей [1]. Управління службами, розгорнутими в туманному обчислювальному середовищі, є складним завдання, а інструменти контейнеризації та оркестрації реалізують її безпроблемне впровадження та використання [2].

Таким чином, дослідження ефективності платформ контейнерної оркестрації при організації туманних обчислень є актуальним завданням розвитку концепції туманних обчислень.

В рамках роботи організується розгортання Docker-контейнерів [3]. Для створення кластера використовується Docker Swarm. Вирішуються завдання вимірювання наступних параметрів: час розгортання одного контейнера, час розгортання групи контейнерів, час відгуку задачі горизонтального масштабування, час затримки передачі. Проводиться аналіз одержаних результатів випробувань.

Основні матеріали дослідження. Для достовірного дослідження для різних інструментів контейнерної оркестрації необхідно створити ідентичні умови виконання тестувань і провести ряд повторень кожного випробування для досягнення достовірних результатів. Для цього скористаємося віртуальними машинами, що надаються Amazon Web Services.

Для достовірності результатів проведення випробування необхідно розробити утиліту автоматичного проведення випробувань, яка має забезпечувати наступний функціонал:

- можливість налаштування, вибір та запуск випробування;
- моніторинг поточного випробування;
- демонстрація результатів випробування.

Для опису варіантів використання утиліти розроблено діаграму варіантів використання (див. рисунок 1) – діаграма, що відображає відносини між акторами та прецедентами.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

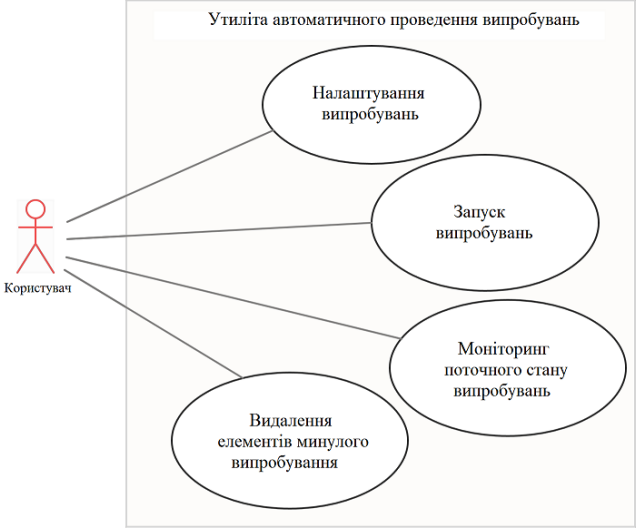

Рисунок 1 – Діаграма прецедентів утиліти автоматичного проведення випробувань

Для того, щоб виміряти час проведення випробувань, було обрано принцип зворотного відгуку кожного розгорнутого контейнера [4], який полягає в наступному:

- запускається слухач (сервер), який приймає повідомлення від контейнерів (клієнтів);

- слухач фіксує час початку випробування та приймає всі повідомлення, надіслані контейнерами під час розгортання, та показує час відгуку.

Щоб отримати час випробування, необхідно від часу останнього відгуку відняти час початку випробування.

Наочно принцип зворотного відгуку кожного контейнера можна побачити на рисунку 2.

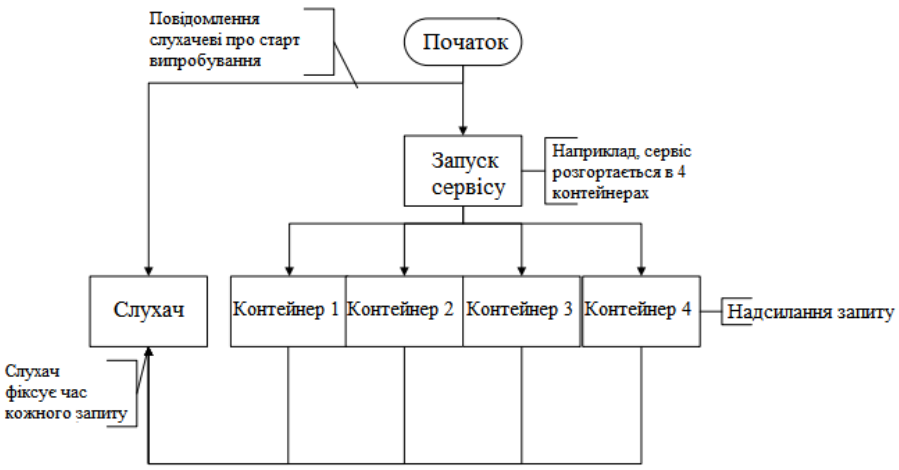

Рисунок 2 – Принцип зворотного відгуку кожного контейнера

Для автоматизації проведення випробувань було розроблено утиліту автоматичного розгортання та масштабування контейнерів мовою bash.

Блок-схему утиліти продемонстровано на рисунках 3– 4.

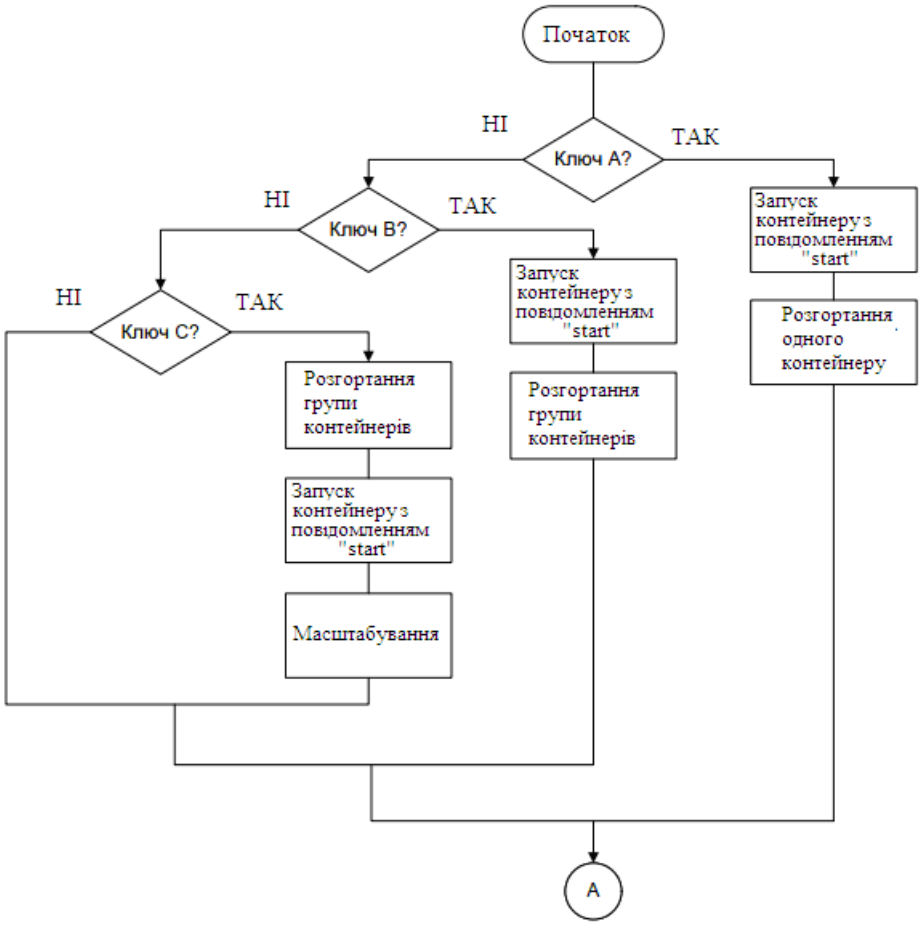

Рисунок 3 – Блок-схема утиліти (фрагмент 1)

Рисунок 4 – Блок-схема утиліти (фрагмент 2)

Для проведення дослідження було обрано Amazon Web Service як сервісу, що пропонує послуги використання обчислювальних потужностей.

Для проведення випробувань нам знадобляться віртуальні машини рівня t2.medium (vCPU: 2, RAM: 4 Гб) у кількості 3 екземплярів з встановлену операційною системою Ubuntu Server 16.04 LTS.

Створимо та проініціалізуємо віртуальні машини (рисунок 5).

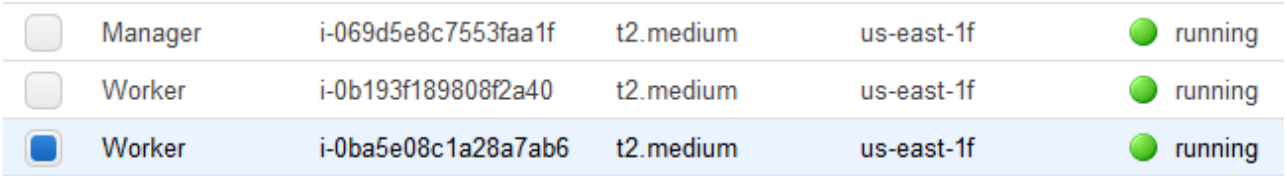

Рисунок 5 – Віртуальні машини в AWS

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Так як віртуальні машини не містять необхідного попередньо встановленого програмного забезпечення (Docker), встановимо на кожній віртуальної машини його вручну.

Для встановлення Docker на віртуальні машини звернемося до офіційної документації за інструкцією із встановлення.

Виконаємо по черзі наступні команди кожної з віртуальних машин (рисунки 6 - 7):

```
$ sudo apt-get update

$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

Рисунок 6 – Команди установки Docker (фрагмент 1)

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Рисунок 7 – Команди установки Docker (фрагмент 2)

Перевіримо стан встановленого програмного забезпечення на кожній віртуальній машині:

```
ubuntu@ip-172-31-67-190:~$ docker info
   Profile: default
 Kernel Version: 4.4.0-1105-aws
 Operating System: Ubuntu 16.04.6 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 3.858GiB
 Name: ip-172-31-67-190
```

Рисунок 8 – Виконання команди docker info на Manager (фрагмент)

```
ubuntu@ip-172-31-64-148:~$ docker info
   Profile: default
 Kernel Version: 4.4.0-1105-aws
 Operating System: Ubuntu 16.04.6 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 3.858GiB
 Name: ip-172-31-64-148
```

Рисунок 9 – Виконання команди docker info на Worker 1 (фрагмент)

```
ubuntu@ip-172-31-69-190:~$ docker info
   Profile: default
 Kernel Version: 4.4.0-1105-aws
 Operating System: Ubuntu 16.04.6 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 3.858GiB
 Name: ip-172-31-69-190
```

Рисунок 10 – Виконання команди docker info на Worker 2 (фрагмент)

Як ми бачимо з рисунків 8 – 10, програмне забезпечення Docker успішно встановлено на кожній віртуальній машині.

Щоб створити кластер, необхідно виконати команду docker swarm init --advertice-addr [advertise-ip]:2377, де advertise-ip – IP-адреса віртуальної машини, де передбачається запуск управляючої ноди.

```
ubuntu@ip-172-31-67-190:~$ docker swarm init --advertise-addr 172.31.67.190
Swarm initialized: current node (1axxwd7ck94i60dq8411t0vma) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-
34oppkufdai92hzcmcbpjfz75n71nqwhngkmxvrlckx2kl7dtu-07npi2zct83il9smhqffzum63
172.31.67.190:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow
the instructions.
```

Рисунок 11 – Запуск кластера Docker Swarm

Керуюча нода запущена, кластер готовий до додавання робочих нод.

Щоб додати робочі ноди в кластер, виконуємо команду з висновку результату команди лістингу N: docker swarm join –token [token] [manager ip]:[manager port].

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

```
ubuntu@ip-172-31-64-148:~$ docker swarm join --token SWMTKN-1-
34oppkufdai92hzcmcbpjfz75n7lnqwhngkmxvrlckx2kl7dtu-07npi2zct83il9smhqffzum63
172.31.67.190:2377
This node joined a swarm as a worker.
```

Рисунок 12 – Додавання робочої ноди 1

```
ubuntu@ip-172-31-69-190:~$ docker swarm join --token SWMTKN-1-
34oppkufdai92hzcmcbpjfz75n7lnqwhngkmxvrlckx2kl7dtu-07npi2zct83il9smhqffzum63
172.31.67.190:2377
This node joined a swarm as a worker.
```

Рисунок 13 – Додавання робочої ноди 2

Перевіримо ноди кластера командою: docker node ls.

```
ubuntu@ip-172-31-67-190:~$ docker node ls
ID                            HOSTNAME            STATUS  AVAILABILITY MANAGER STATUS   ENGINE VERSION
pcu632mf2mqwglrvw2s432ib3     ip-172-31-64-148    Ready   Active                        19.03.9
1axxwd7ck94i60dq8411t0vma *   ip-172-31-67-190    Ready   Active       Leader           19.03.9
zobub3a0zs7i8ygimloyonlc5     ip-172-31-69-190    Ready   Active                        19.03.9
```

Рисунок 14 – Виконання команди docker node ls

З рисунків 11 – 14 видно, що кластер створено. Як ми можемо бачити на рисунку 14, нода ip-172-31-67-190 є керуючою, а ноди ip-172-31-64-148 та ip-172-31-69-190 є робітниками.

Зрештою, ми можемо зробити висновок про те, що кластер запущений і готовий до проведення дослідження.

Для організації обміну повідомленнями використовуємо мережеву утиліту операційної системи Linux Netcat.

Створимо слухача, який прийматиме повідомлення і фіксуватиме час приходу повідомлення записом у файл result.txt. Код слухача продемонстровано на рисунку 15.

```
#!/bin/bash
ifconfig
while true; do ((nc -l 4789 >> result.txt | exit) && printf "$(date +%s%N | cut
-b1-13) \n" >> result.txt); done
```

Рисунок 15 – Реалізація слухача (server.sh)

За допомогою Dockerfile створюється контейнер-клієнт (client:v2).

Контейнер-клієнт виконує дві функції:

- Встановлює з'єднання зі слухачем;
- Надсилає йому повідомлення «start».

Код контейнера-клієнта (client:v2) показано на рисунку 16.

```
FROM ubuntu
RUN apt update && apt-get install -y netcat
CMD ((echo "start") | (nc 172.31.67.190 4789 | exit)) && while true; do echo
"msg"; sleep 1; done;
```

Рисунок 16 – Реалізація контейнера клієнта (client:v2)

За допомогою Dockerfile створюється контейнер-клієнт (client:v1).

Контейнер-клієнт виконує дві функції:

- Встановлює з'єднання з контейнером-слухачем;
- Надсилає йому повідомлення «message».

Код контейнера-клієнта (client:v1) продемонстровано на рисунку 17.

```
FROM ubuntu
RUN apt update && apt-get install -y netcat
CMD ((echo "message") | (nc 172.31.67.190 4789 | exit)) && while true; do echo
"msg"; sleep 1; done;
```

Рисунок 17 – Реалізація контейнера клієнта (client:v1)

Для проведення досліджень використовуємо утиліту автоматичного проведення випробувань із ключами:

- A (випробування "Один контейнер");
- B (випробування "Група контейнерів");
– C (випробування "Горизонтальне масштабування").

Продемонструємо процес розгортання групи контейнерів на діаграмі розгортання (рисунок 18).

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

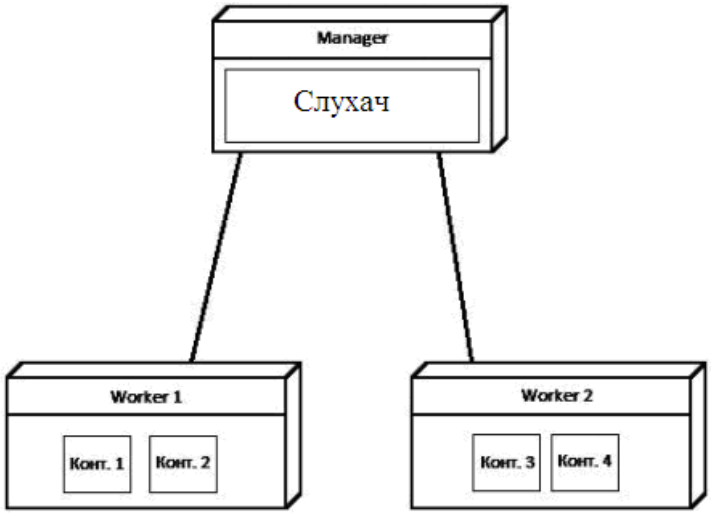

Рисунок 18 – Діаграма розгортання для випробування "Група контейнерів"

На рисунках 19 – 20 можемо бачити проведення випробування «Один контейнер» за допомогою утиліти автоматичного проведення випробувань.

```
ubuntu@ip-172-31-67-190:~$ bash auto-research.sh -A

overall progress: 1 out of 1 tasks
1/1: running
verify: Service converged
```

Рисунок 19 – Розгортання контейнера за допомогою утиліти

```
start
1591593712606
message
1591593714458
```

Рисунок 20 – Перегляд слухачів

Повторимо випробування 10 разів і представимо отримані дані табличному вигляді (див. таблицю 1).

Таблиця 1

Результати випробування "Один контейнер"

N	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
1	1852	1260	1866	2480	1072	1773	1130	2496	1857	2216

де N – кількість контейнерів, $t_{1\text{-}10}$ – час виконання випробування у мілісекундах.

На рисунках 21 – 22 можемо бачити проведення випробування "Група контейнерів" за допомогою утиліти автоматичного проведення випробувань.

```
ubuntu@ip-172-31-67-190:~$ bash auto-research.sh -B 4

overall progress: 4 out of 4 tasks
1/4: running
2/4: running
3/4: running
4/4: running
verify: Service converged
```

Рисунок 21 – Розгортання групи контейнерів за допомогою утиліти

```
start
1591594448401
message
1591594449507
message
1591594449833
message
1591594449923
message
1591594450322
```

Рисунок 22 – Перегляд даних слухачів

Повторимо кожне випробування 10 разів і представимо отримані дані в табличному вигляді (див. таблицю 2).

Таблиця 2

Результати випробування "Група контейнерів"

N	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
2	1814	1793	1797	2384	2009	1800	1761	1139	1904	2168
4	1921	2386	2069	1914	1829	2875	1785	1822	2350	1736
8	2624	3385	2585	2531	3317	2575	2226	2960	2771	2538
16	4805	3728	5300	4217	3719	3906	4251	3973	4219	4793
32	6026	6288	6429	6191	6511	6813	6938	5894	6103	7178
48	8442	9132	7911	8437	8628	8071	7822	8365	7681	8411
64	9693	9903	10774	9766	10535	9943	10811	9836	9678	10234
128	18759	19690	19318	19266	19306	19416	18939	18859	19400	19056
150	23761	23284	23616	22577	23173	23256	23902	22913	23553	23640

де N – кількість контейнерів, $t_{1\text{-}10}$ – час виконання випробування у мілісекундах.

На рисунку 24 можемо бачити, що зі збільшенням кількості контейнерів, що розгортаються до 180, керуюча нода не справляється з даним завданням через навантаження на ядра процесора, яка досягає 100%, що призводить до примусового завершення команди розгортання.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

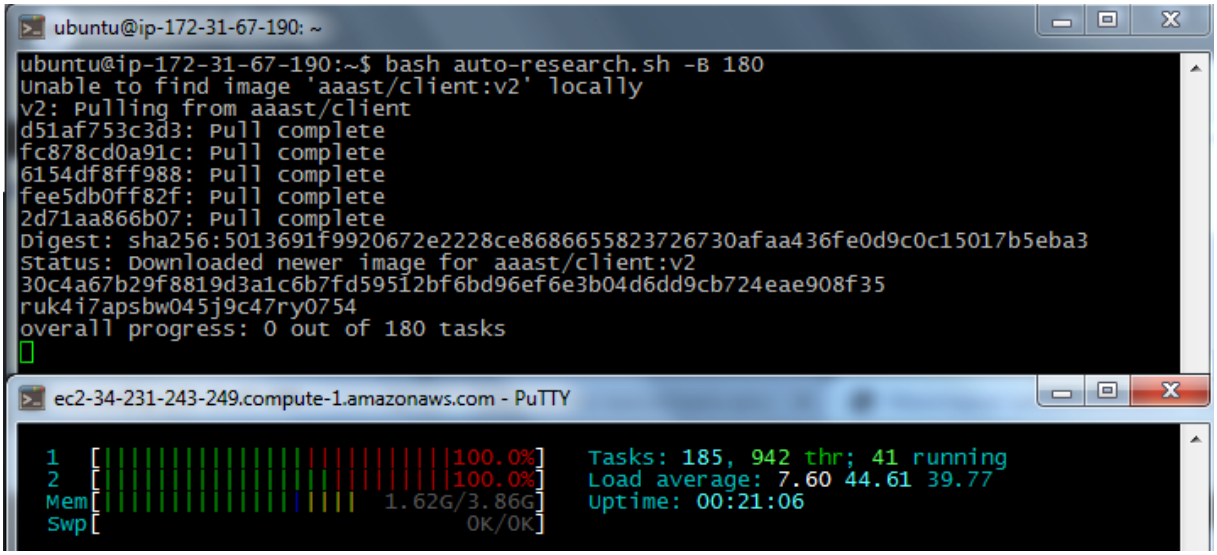

Рисунок 24 – Розгортання 180 контейнерів

На рисунках 25 – 26 можемо бачити проведення випробування «Горизонтальне масштабування» за допомогою утиліти автоматичного випробування.

```
ubuntu@ip-172-31-67-190:~$ bash auto-research.sh -C 1 16
kv5smel0swnfqp26wzub2dke0
overall progress: 1 out of 1 tasks
1/1: running
verify: Service converged

my-client scaled to 16
overall progress: 16 out of 16 tasks
1/16: running
2/16: running
3/16: running
4/16: running
5/16: running
6/16: running
7/16: running
8/16: running
9/16: running
10/16: running
11/16: running
12/16: running
13/16: running
14/16: running
15/16: running
16/16: running
verify: Service converged
```

Рисунок 25 – Горизонтальне масштабування за допомогою утиліти

```
message
1591598241328

start
1591598247980
message
1591598249379
message
1591598249475
message
1591598249707
message
1591598249882
message
1591598249907
message
1591598249982
message
1591598250150
message
1591598250211
message
1591598250475
message
1591598250570
message
1591598250719
message
1591598250803
message
1591598251066
message
1591598251086
message
1591598251570
message
1591598251978
```

Рисунок 26 – Перегляд даних слухачів

Повторимо кожне випробування 10 разів і представимо отримані дані в табличному вигляді (див. таблицю 3).

Таблиця 3

Результати випробування "Горизонтальне масштабування"

N-M	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
1-16	4043	4028	3073	3922	3914	3693	4388	3089	3338	3590
16-64	7525	6972	7770	7636	7837	7079	8734	6912	6772	6279

де N – вихідна кількість контейнерів, M – кінцева кількість контейнерів, $t_{1\text{-}10}$ – час виконання випробування у мілісекундах.

На рисунках 27 – 35 можемо бачити проведення випробування «Latency».

```
ubuntu@ip-172-31-64-148:~$ docker network create \
>   --driver overlay \
>   --subnet 10.0.9.0/24 \
>   --opt encrypted \
>   my-network
toq9fcmcbwwyq2ollstjv9t47
```

Рисунок 27 – Створення мережі

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

```
ubuntu@ip-172-31-64-148:~$ docker network ls
NETWORK ID   NAME            DRIVER   SCOPE
0eba6ee9f6da bridge          bridge   local
cb519c9b6ceb docker_gwbridge bridge   local
8abea94289d5 host            host     local
c0bi3i7g366b ingress         overlay  swarm
toq9fcmcbwwy my-network      overlay  swarm
e509e9407e6f none            null     local
```

Рисунок 28 – Перевірка мережі

```
ubuntu@ip-172-31-64-148:~$ docker service create \
>   --replicas 3 \
>   --name my-web \
>   --network my-network \
>   nginx
vkl22s0fq8gk8u2htkje8632s
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service converged
```

Рисунок 29 – Створення та підключення першого сервісу до мережі

```
ubuntu@ip-172-31-64-148:~$ docker service inspect \
>   --format='{{json .Endpoint.VirtualIPs}}' \
>   my-web
[{"NetworkID":"toq9fcmcbwwyq2ollstjv9t47","Addr":"10.0.9.2/24"}]
```

Рисунок 30 – Знаходження IP-адреси першого сервісу

```
ubuntu@ip-172-31-64-148:~$ docker service create --replicas 3 --name my-
web2 --network my-network nginx
adtfg4qvpt0r2c20yasc5ac0u
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
```

Рисунок 31 – Створення та підключення другого сервісу до мережі

```
ubuntu@ip-172-31-64-148:~$   docker   service   inspect   --format='{{json
.Endpoint.VirtualIPs}}' my-web2
[{"NetworkID":"toq9fcmcbwwyq2ollstjv9t47","Addr":"10.0.9.8/24"}]
```

Рисунок 32 – Знаходження IP-адреси другого сервісу

```
ubuntu@ip-172-31-64-148:~$ docker exec -it 60e8d8b04bff bash
```

Рисунок 33 – Підключення до сервісу

```
root@60e8d8b04bff:/# apt-get install iputils-ping
```

Рисунок 34 – Встановлення iputilits-ping

```
root@60e8d8b04bff:/# ping -c 10 -i 0.1 -v 10.0.9.2
PING 10.0.9.2 (10.0.9.2) 56(84) bytes of data.
64 bytes from 10.0.9.2: icmp_seq=1 ttl=64 time=0.127 ms
64 bytes from 10.0.9.2: icmp_seq=2 ttl=64 time=0.112 ms
64 bytes from 10.0.9.2: icmp_seq=3 ttl=64 time=0.108 ms
64 bytes from 10.0.9.2: icmp_seq=4 ttl=64 time=0.096 ms
64 bytes from 10.0.9.2: icmp_seq=5 ttl=64 time=0.109 ms
64 bytes from 10.0.9.2: icmp_seq=6 ttl=64 time=0.109 ms
64 bytes from 10.0.9.2: icmp_seq=7 ttl=64 time=0.111 ms
64 bytes from 10.0.9.2: icmp_seq=8 ttl=64 time=0.110 ms
64 bytes from 10.0.9.2: icmp_seq=9 ttl=64 time=0.108 ms
64 bytes from 10.0.9.2: icmp_seq=10 ttl=64 time=0.110 ms

--- 10.0.9.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 936ms
rtt min/avg/max/mdev = 0.096/0.110/0.127/0.007 ms
```

Рисунок 35 – Виконання команди ping –c 10 –i 0.1 –v 10.0.9.2

Таблиця 4

Результати випробування "Latency"

N	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
10	0.127	0.112	0.108	0.096	0.109	0.109	0.111	0.110	0.108	0.110

де N – кількість запитів, $t_{1\text{-}10}$ – час виконання запитів мілісекундах.

Проаналізуємо отримані дані з раніше проведених випробувань та представимо отримані значення у табличному вигляді (таблиця 5).

Таблиця 5

Підсумкові результати випробувань

Найменування випробувань	**N**	**Середній час, мс**	**Середній час розгортання одного контейнера, мс**
Час розгортання одного контейнера в кластері	1	1800,2	1800,20
	2	1856,9	928,45
	4	2 068,7	517,18
	8	2 751,2	343,90
	16	4 291,1	268,19
	32	6 437,1	201,16
	48	8 290,0	172,71

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

	64	10 117,3	158,08
	128	19 255,6	150,43
	150	23 533,6	157,02
	180	N/A	N/A
Час горизонтального масштабування існуючих контейнерів	1-16	3 707,8	247,19
	16-64	7 351,6	153,16
Тимчасові затримки передачі повідомлень між хостами в кластері	10	0,11	N/A

Графіки одержаних результатів.

Рисунок 36 – Середній час розгортання

Рисунок 37 – Середній час розгортання одного контейнера

З рисунка 36 можемо бачити, що зі збільшенням кількості розгортання контейнерів час розгортання збільшується.

Так, припустимо, зі збільшенням кількості контейнерів з 1 до 16, час розгортання збільшується в 2,4 рази, зі збільшенням кількості контейнерів з 16 до 64, час розгортання також збільшується в 2,4 рази, при збільшенні кількості з 64 до 150, час розгортання збільшується в 2,3 рази.

На рисунку 37 спостерігаємо, що зі збільшенням числа розгорнутих контейнерів середній час розгортання одного контейнера зменшується.

Так, для розгортання одного контейнера потрібно 1800,2 мс, коли для групи з 64 контейнерів, середній час розгортання одного контейнера займає лише 158,08 мс. Зниження часу розгортання одного контейнера зі збільшенням кількості контейнерів пов'язане з розпаралелюванням виконання завдань розгортання у кластері.

Також варто звернути увагу на те, що при розгортанні групи з 128 контейнерів час розгортання одного контейнера займає 150,01 мс, коли для групи зі 150 контейнерів час розгортання займає 155,78 мс, що говорить нам про те, знайдено точку насичення середнього часу розгортання одного контейнера під час виконання завдання розгортання групи контейнерів.

Окремо варто зазначити, що зі збільшенням кількості контейнерів, що розгортаються до 180, керуюча нода не справляється з даним завданням через навантаження на ядра процесора, що досягає 100%, що призводить до примусового завершення команди розгортання.

З таблиці 5 можемо помітити, що масштабувати контейнеризовані програми вигідніше, ніж розгортати необхідне кількість заново, тому що при еквівалентних задачах (розгортання 1642 контейнерів та горизонтальне масштабування з 1 до 16 контейнерів, а також розгортання 48 контейнерів та горизонтальне масштабування з 16 до 64 контейнерів) бачимо, що масштабування виконується швидше приблизно на 12-14% за рахунок того, що масштабований контейнеризований додаток не потребує додаткового скачування образів.

Середній час затримки при передачі запитів між кластеризованими вузлами склало 0.11 мс.

Тому що лідером у програмному забезпеченні для автоматизації розгортання та управління додатками в середовищах з підтримкою контейнеризації є Docker, а Docker Swarm є рідною системою кластеризації, яка перетворює набір Docker-хостів на один послідовний кластер, можемо відзначити безперешкодну взаємодію кластера Docker Swarm із Docker-контейнерами.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Також варто відзначити, що платформа контейнерної оркестрації Docker Swarm відрізняється простотою освоєння та розгортання кластера.

Висновок. В роботі було розроблено утиліту автоматичного проведення випробувань. Також було проведено випробування «Один контейнер», «Група контейнерів», «Горизонтальне масштабування», «Latency».

Для виконання поставленої мети було вирішено такі завдання:

– здійснено підбір літератури, наукових публікацій та інтернет статей, необхідних для проведення дослідження;

– иконано огляд платформ контейнерної оркестрації;

– визначено ключові вимоги та критерії проведення дослідження;

– спроектовано та реалізовано утиліту автоматичного проведення випробувань;

– виконано дослідження ефективності платформ контейнерної оркестрації з організацією туманних обчислень

ЛІТЕРАТУРА

1. Что такое интернет вещей и как он работает? [Електроний ресурс] URL: https://server-shop.ua/the-internet-of-things-and-the-scope-of-its-use.html (дата звернення 06.12.2022р.)

2. IoT, туман и облака: поговорим про технологии? [Електроний ресурс] URL: https://habr.com/ru/company/cloud4y/blog/467711/ (дата звернення 06.12.2022р.)

3. Моуэт Э. Использование Docker / Э. Моуэт; пер. с англ. А.В. Снастина; науч. ред. А. А. Маркелов. – М.: ДМК Пресс, 2017. – 354 с.

4. Evaluating Container Platforms at Scale. [Електроний ресурс] URL: https://medium.com/on-docker/evaluating-container-platforms-at-scale-5e7b44d93f2c (дата звернення 06.12.2022р.)

5. Колісник Д.Р., Місевич К.С., Коваленко С.В. Системна архітектура IoT-Fog-Cloud для систем аналізу великих даних і кібербезпеки: огляд туманних обчислень, впровадження аудиту інтернету речей. Сучасний захист інформації. 2020. № 3. С. 34–38.

REFERENSE

1. Chto takoe ynternet veshchei y kak on rabotaet? [Elektronyi resurs] URL: https://server-shop.ua/the-internet-of-things-and-the-scope-of-its-use.html (data zvernennia 06.12.2022r.)

2. IoT, tuman y oblaka: pohovorym pro tekhnolohyy? [Elektronyi resurs] URL: https://habr.com/ru/company/cloud4y/blog/467711/ (data zvernennia 06.12.2022r.)

3. Mouэт Э. Yspolzovanye Docker / Э. Mouэт; per. s anhl. A.V. Snastyna; nauch. red. A. A. Markelov. – M.: DMK Press, 2017. – 354 s.
4. Evaluating Container Platforms at Scale. [Elektronyi resurs] URL: https://medium.com/on-docker/evaluating-container-platforms-at-scale-5e7b44d93f2c (data zvernennia 06.12.2022r.)
5. Kolisnyk D.R., Misevych K.S., Kovalenko S.V. Systemna arkhitektura IoT-Fog-Cloud dlia system analizu velykykh danykh i kiberbezpeky: ohliad tumannykh obchys-len, vprovadzhennia audytu internetu rechei. Suchasnyi zakhyst informatsii. 2020. № 3. S. 34–38.

Received 10.12.2022.
Accepted 15.12.2022.

Research of the efficiency of computing services management platforms in the organization of fog computing

The work is devoted to studying the effectiveness of computing service management platforms in the organization of Fog Computing.

As part of the work, the effectiveness of container orchestration platforms with the Fog computing organization is being studied.

During the research, it is necessary to complete the following tasks: 1) select literature, scientific publications and Internet articles necessary for the research; 2) inspect container orchestration platforms; 3) determine the key requirements and criteria for conducting the study; 4) design and implement an automatic testing utility; 5) conduct a study of the effectiveness of container orchestration platforms with the organization of fog computing; 6) analyze the results obtained and draw related conclusions.

Deployment of Docker containers is organized. Docker Swarm is used to create a cluster. The problems of measuring the following parameters are solved: deployment time of one container, deployment time of a group of containers, response time of the horizontal zoom task, transmission delay time. The analysis of the obtained test results is carried out.

Keywords: fog computing, utility, docker, docker container, docker swarm, horizontal scaling, container, platform, Internet of Things.

Островська Катерина Юріївна – к.т.н., доц. кафедри інформаційних технологій і систем ІПБТ УДУНТ.

Шерстяних Микита Олександрович – магістр кафедри інформаційних технологій і систем ІПБТ УДУНТ.

Стовпченко Іван Володимирович – старший викладач кафедри інформаційних технологій і систем ІПБТ УДУНТ.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Каліберда Юрій Олегович – старший викладач кафедри інформаційних технологій і систем ІПБТ УДУНТ.

Ostrovska Kateryna - Ph.D., Assoc. Department of Information Technologies and Systems of Information Technology of USUNT.

Sherstyanikh Mykita – Master of the Department of Information Technologies and Systems of Information Technology and Information Technology of UDUNT.

Stovchenko Ivan - senior lecturer at the Department of Information Technologies and Systems of Information Technology and Information Technology of UDUNT.

Kaliberda Yury - senior lecturer at the Department of Information Technologies and Systems of Information Technology and Information Technology of UDUNT.