УДК 004 DOI: 10.31673/2786-8362.2023.010707

Щербина І.С., к.т.н., **Явор Д.В.**

ВПЛИВ ТЕХНОЛОГІЇ DOCKER НА АРХІТЕКТУРУ МІКРОСЕРВІСІВ

Shcherbyna I.S., Yavor D.V. Influence of Docker technology on microservices architecture. In today's information society, where market dynamics and user requirements are constantly growing, software developers face the challenges of rapid development, maintaining high scalability, and ensuring the reliability of information systems. In this context, microservices architecture has become a popular approach that allows you to decompose software into small functional blocks called microservices.

Recently, Docker has become one of the most popular technologies for containerizing applications. Docker provides the ability to isolate applications and their dependencies in containers that can be easily moved and deployed on a variety of environments, including local servers, cloud infrastructures, and container orchestrators such as Kubernetes.

This article is an in-depth exploration of the impact of using Docker on the architectural aspects of microservice applications. It addresses the issues of scalability, reliability, performance, and provisioning in the context of using Docker in microservices deployment. The article also compares Docker with other containerization approaches and analyzes its impact on architectural decisions.

Keywords: Docker, microservices, containerization, scalability, reliability, performance, infrastructure, deployment, container orchestrators.

Щербина І.С., Явор Д.В. Вплив технології Docker на архітектуру мікросервісів. Дослідження впливу Docker на архітектуру мікросервісів в умовах швидкозмінюючогося інформаційного суспільства. Мікросервісна архітектура - ключ для швидкої розробки та масштабованості. Docker, як популярний інструмент для контейнеризації, забезпечує ізоляцію та переносимість додатків між середовищами, включаючи локальні сервери та хмари. Стаття глибоко аналізує вплив Docker на мікросервіси, розглядаючи масштабованість, надійність та продуктивність систем.

Ключові слова: Docker, мікросервіси, контейнеризація, масштабованість, надійність, продуктивність, інфраструктура, розгортання, контейнерні оркестратори.

Вступ

Постановка задачі. В сучасному світі інформаційних технологій, де швидкість розробки програмного забезпечення є критичною, а вимоги до масштабованості та надійності ростуть експоненційно, архітектура мікросервісів стала основою для створення гнучких та високодіючих додатків. Вона дозволяє розбити складні системи на менші функціональні блоки, мікросервіси, що можуть функціонувати незалежно і бути розробленими різними командами.

Проте ця модульнізація створює нові виклики у сфері інфраструктури для розгортання та керування цими мікросервісами. Вибір правильної інфраструктури є завданням, що вимагає серйозного обговорення і аналізу.

Docker, відомий як популярна технологія контейнеризації додатків, став важливим інструментом для створення інфраструктури мікросервісів. Він дозволяє ізолювати додатки та їх залежності в контейнерах, спрощуючи розгортання та переносимість. Однак з популярністю Docker постають нові питання і виклики, пов'язані з використанням цієї технології в контексті мікросервісів.

Звідси виникає загальна наукова проблема: як використання Docker впливає на архітектурні аспекти мікросервісних додатків та які виклики це створює? Для кращого розуміння, нехай розглянемо це на прикладі. Представимо команду розробників, що працює над одним проектом, і використовує різні операційні системи. Кожне середовище має свої унікальні налаштування, бібліотеки та мови програмування. Внаслідок цього виникають конфлікти та складнощі в управлінні.

© Щербина І.С., Явор Д.В., 2023

Цей сценарій ілюструє загальний виклик, пов'язаний з інтеграцією Docker у розробку та деплоймент мікросервісів, що потребує докладного аналізу. Дослідження впливу Docker на масштабованість, надійність, продуктивність та безпеку системи є критичним. Розуміння переваг та обмежень використання Docker в мікросервісній архітектурі має важливе значення для розробників і архітекторів, що працюють над проектами в галузі інформаційних технологій.

Аналіз досліджень та публікацій. Для аналізу були розглянуті дві ключові публікації, які зосереджені на ролі Docker у сфері розробки програмного забезпечення та контейнеризації.

Перша публікація [1] була спрямована на вивчення впливу Docker на розробку та розгортання програм. Основна проблема, яку вона досліджує, полягає в пошуку оптимальних рішень для розробників у сфері розгортання додатків та забезпечення сумісності між різними середовищами. Автори публікації аналізують, як Docker спростив роботу розробників, забезпечивши їм інструмент командного рядка, що спрощує роботу з контейнерами, незалежно від того, чи це локальна, чи хмарна розробка. Дослідження дійшло висновку, що Docker зробив розробку та розгортання програмного забезпечення більш ефективним та забезпечив єдиний стандарт для роботи з контейнерами.

Друга публікація [2] підкреслює важливу роль Docker у світі контейнеризації. Вона зазначає, що Docker залишається ключовим гравцем, забезпечуючи розробникам інструмент для створення та розгортання додатків без проблем з сумісністю. Дослідження також акцентує увагу на тому, як Docker розв'язав проблему несумісності середовищ, яка раніше заважала ефективній розробці.

Обидві публікації підкреслюють велике значення Docker у розробці програмного забезпечення та контейнеризації, надаючи розробникам інструменти для ефективної роботи та спрощуючи процеси створення мікросервісів. Докер став стандартом для роботи з контейнерами та розв'язав проблеми, які раніше перешкоджали розвитку цих технологій.

Таким чином, аналіз цих публікацій вказує на важливість Docker як інструмента для розробки та розгортання програмного забезпечення та підкреслює його роль у створенні єдиної стандартизованої платформи для контейнеризації та мікросервісної архітектури.

Мета статті. Метою цієї статті є ретельний аналіз важливості Docker у сучасному світі розробки програмного забезпечення та контейнеризації. Стаття спрямована на висвітлення ключових аспектів використання Docker, його впливу на розробку та розгортання програмного забезпечення, а також створення стандартів для мікросервісної архітектури та контейнеризації. Метою статті є також визначення того, як Docker спрощує роботу розробників та дозволяє їм створювати програмне забезпечення, яке працює однаково в усіх середовищах.

Виклад основного матеріалу дослідження.

Останні тенденції у розробці додатків перенаправляють галузі (як технологічні, так і нетехнічні) до більш хмарної та розподіленої моделі зі стратегіями, спрямованими на цифрові технології. Багато організацій впроваджують нові технології та розподілені робочі процеси, що дозволяє їм досягати більш ефективних результатів. Однак ті, хто раніше почали використовувати сучасні стратегії та інструменти розробки додатків, такі як контейнеризація та багатохмаркові середовища, отримують серйозну конкурентну перевагу.

Docker - це сучасна технологія віртуалізації з відкритим вихідним кодом[3], відома як платформа для програмних контейнерів. Вона дозволяє упаковувати додаток у контейнер, включаючи всі його компоненти та середу виконання, створюючи єдиний та переносний пакет. Ці контейнери можуть бути запущені на різних хост-платформах, таких як Amazon Web Services (AWS), Google Cloud, Microsoft Azure та багатьох інших. Далі на Рис.1 наведений абстрактний приклад розміщення Docker контейнера на платформі AWS.

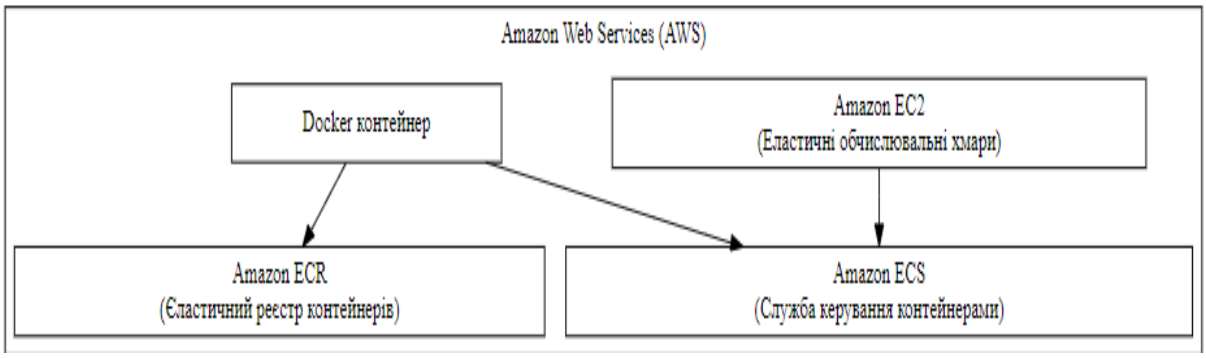

Рис. 1. Приклад розміщення Docker контейнера на платформі AWS

У цій схемі показано три основні компоненти AWS:

Amazon EC2 (Еластичні обчислювальні хмари): Це віртуальні сервери в хмарі AWS. Ви можете розміщувати контейнери на віртуальних машинах Amazon EC2.

Amazon ECS (Служба управління контейнерами): Amazon ECS – це служба оркестрації контейнерів, яка допомагає керувати та розгортати контейнери на Amazon EC2 примірниках.

Amazon ECR (Єластичний регістр контейнерів): Amazon ECR — це служба для зберігання та керування Docker-образами контейнерів. Ваші контейнери можуть бути розміщені у Amazon ECR для подальшого використання.

Для будь-якого запуску контейнера Docker, потрібно мати Docker Engine, який надає середовище виконання для контейнерів. Перш ніж запустити контейнер, потрібно створити файл Docker, який визначає всі налаштування для його запуску, включаючи налаштування мережі та структуру файлової системи. Цей файл Docker дозволяє створити Docker-образ, який представляє собою готовий до запуску статичний пакет для Docker Engine. Docker-образ фактично є набором інструкцій для створення контейнера Docker. На Рис.2 наведений приклад простого Docker файлу, команди якого було взято з офіційної документації [4] для опису контейнера, який запускає додаток у середовищі.

```
# Використовуємо офіційний базовий образ з Python 3.8
FROM python:3.8

# Встановлюємо необхідні залежності для додатку
RUN pip install Flask

# Створюємо робочий каталог у контейнері
WORKDIR /app

# Копіюємо файли додатку в контейнер
COPY . /app

# Запускаємо додаток при старті контейнера
CMD ["python", "app.py"]
```

Рис. 2. Приклад Docker файлу

У цьому Docker-файлі ми використовуємо базовий образ Python 3.8. Ми встановлюємо Flask, як приклад залежностей, і налаштовуємо робочий каталог /app. Потім ми копіюємо файли додатку в контейнер і вказуємо команду для запуску додатку при старті контейнера.

Додатковою перевагою Docker є Docker Hub, велика екосистема контейнерних мікросервісів. Docker Hub містить велику кількість готових Docker-образів, що полегшує пошук та використання готових рішень, таких як веб-сервери та бази даних у контейнерах. Docker також підтримує Ініціативу відкритих контейнерів (OCI), забезпечуючи єдиний та відкритий формат упакування контейнерів.

Сьогодні багато розробників використовують Docker для створення мікросервісів. Давайте розглянемо, що являють собою мікросервіси і чому вони такі популярні

Мікросервіси - це підхід до розробки програмного забезпечення, в якому додаток розглядається як набір невеликих та спеціалізованих сервісів, які працюють разом, щоб забезпечити загальну функціональність. Кожен мікросервіс відповідає за обраний фрагмент функціональності і може незалежно масштабуватися та оновлюватися.

Наприклад, коли користувач робить запит на веб-сторінку, цей запит розбивається на кілька спеціалізованих запитів і кожен з них направляється до відповідного мікросервісу. Кожен мікросервіс функціонує як окремий процес, що дозволяє їм бути незалежними один від одного на Рис.3 можна побачити діаграму, яка зображує цю взаємодію. Це також спрощує моніторинг, створення резервних копій та обробку аварійних ситуацій.

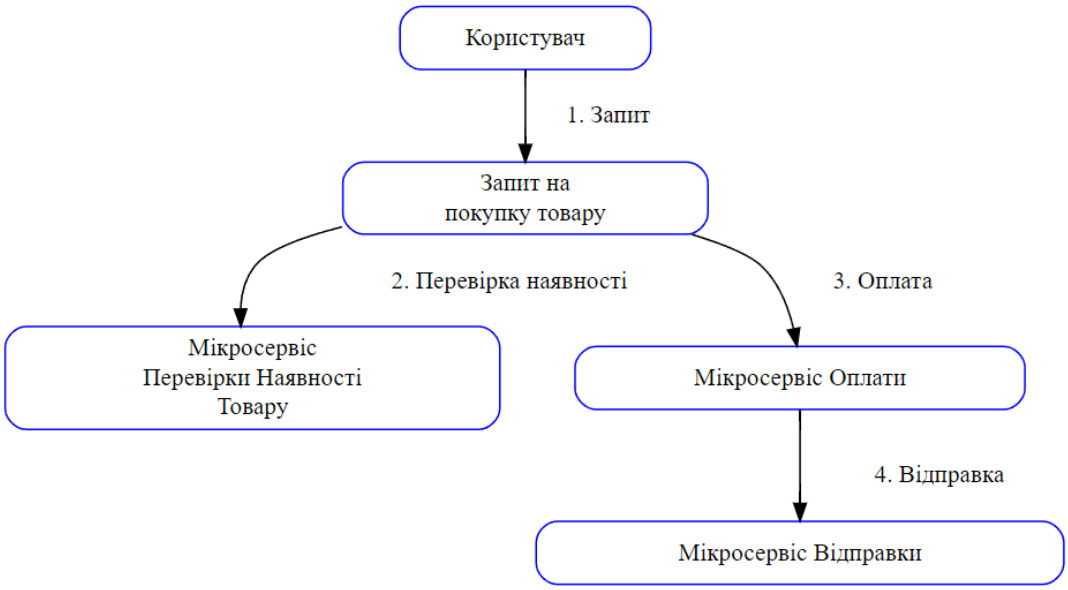

Рис. 3. Приклад взаємодії мікросервісів між собою

Одним із прикладів успішного впровадження мікросервісного підходу є Amazon. Початково веб-сайт Amazon.com в 2001 році був побудований на монолітній архітектурі, де різні компоненти були тісно пов'язані між собою. З часом це призвело до ускладнення та збільшення витрат на розробку програмного забезпечення. Іншими словами, архітектура не могла масштабуватися на відповідному рівні разом зі зростанням клієнтської бази.

Але компанія Amazon вирішила вирішити цю проблему та перетворити свою монолітну архітектуру на мікросервіси. Вони розбили свої додатки на невеликі блоки, кожен з яких відповідав за конкретну функціональність. Ця роздроблена архітектура дозволила Amazon незалежно оновлювати та масштабувати кожен мікросервіс, що стало однією з ключових причин їх успіху. Нині Amazon є однією з найбільших компаній у світі. Це підтверджує статистика за 2019-2020 рік на сайті Canalys Рис.4

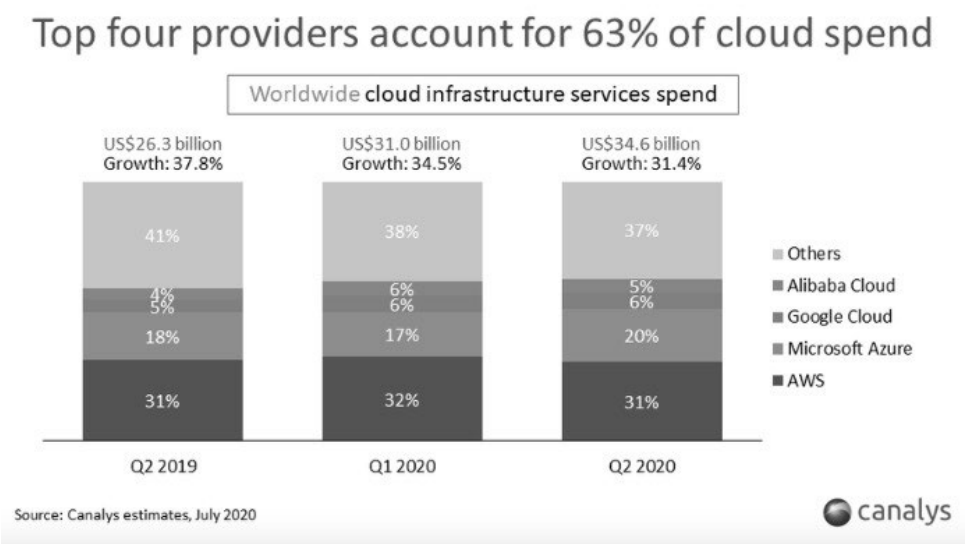

Рис. 4. Статистика витрат на хмарні сервіси

Виходячи зі статистики, наведеної на рисунку 4, можна сказати, що AWS займає 31 відсоток всього ринку хмарних провайдерів, і ці цифри продовжують зростати.

Так як ми розглянули, що таке Docker та мікросервіси, давайте розглянемо чому вони так взаємопов'язані.

Одним з використання Docker в мікросервісах може бути автоматизація розгортання додатків у вигляді портативних самодостатніх контейнерів, як у хмарних середовищах, так і на локальних платформах, включаючи Linux і Windows.

Також Docker має деякі переваги для реалізації мікросервісів:

- Він легший за віртуальні машини, що робить його більш ефективним за ресурсами та дозволяє в більшості випадків заощадити фінанси.
- Забезпечує єдине середовище для розробки та виробництва, що допомагає пришвидшити розробку та тестування програмного забезпечення.
- Спрощує безперервну інтеграцію і розгортання, завдяки тому, що Docker вже має налаштований конфігураційний файл, потрібно тільки розгорнути контейнер у середовищі.
- Інтегрується з популярними інструментами і сервісами, такими як AWS, Microsoft Azure, Ansible, Kubernetes, Istio тощо.

Виходячі зі всіх цих наведених плюсів стає одразу зрозуміло чому Docker на стільки важливий для розробки мікросервісів.

Крім цього, вище ми згадували про таку популярну систему для розгортування контейнерів як Kubernetes багато хто думає, що Docker та Kubernetes не можуть існувати без один одного і є однією системою, але це не так.

Бо Docker дає можливість помістити все необхідне для запуску програмного забезпечення в контейнер, але без можливості керувати ними, як раз для цього і був створений Kubernetes. Він відповідає за безпечне розгортування та керування всіма контейнерами в системі.

Якщо розглянути Kubernetes документацію [5] більш детально то можна виділити, що він має такі обов'язки:

Маштабування: Kubernetes дозволяє масштабувати мікросервіси, адаптуючи їхню кількість відповідно до попиту.

Обробка Помилок: Платформа автоматично виявляє та вирішує помилки, що виникають під час роботи мікросервісів.

Шаблони для Розгортання: Kubernetes надає готові шаблони для розгортання, що полегшують ініціалізацію сервісів.

Управління Ресурсами: Kubernetes дозволяє призначити кожному контейнеру обмеження CPU та RAM.

Заміна Падаючих Контейнерів: В разі некоректної роботи контейнера, Kubernetes

автоматично замінює його на новий.

Завершуючи, важливо підкреслити, що Docker та Kubernetes - це дві важливі технології, які, хоч і розроблені для різних завдань, працюють дуже добре разом.

Висновки

У підсумку, використання Docker та Kubernetes у розробці мікросервісних додатків визнано ключовим для створення стабільних та масштабованих рішень. Docker надає можливість ізолювати додаток у контейнер, забезпечуючи його переносність та легкість розгортання, що є основою розробки мікросервісів в децентралізованій архітектурі. Мікросервіси допомагають поділити додаток на невеликі автономні частини, спрощуючи розробку та масштабування додатків, а Kubernetes допомагає оркеструвати їх роботу та забезпечує високу доступність.

З цими інструментами розробники та організації можуть забезпечити конкурентоспроможність та ефективність у розробці додатків. Дотримання принципів децентралізації та автономності під час створення мікросервісів сприятиме стабільності та гнучкості системи. Використання готових рішень та ресурсів, таких як Docker Hub та AWS ECS, спрощує процес розробки та управління мікросервісами.

Загалом, використання Docker та Kubernetes в розробці мікросервісів відкриває можливості для створення сучасних та надійних додатків, які можуть ефективно взаємодіяти та масштабуватися. Це важливий крок у розвитку сучасних технологій та розробці додатків.

Список використаної літератури:

1. McCarty S. Docker really did change the world. InfoWorld. URL: https://www.infoworld.com/article/3639596/docker-really-did-change-the-world.html

2. Mardeni A. Why is Docker so Popular. Engineering Education (EngEd) Program | Section. URL: https://www.section.io/engineering-education/why-is-docker-so-popular/

3. Canalys Newsroom - Global cloud services market Q2 2020. Canalys. URL: https://www.canalys.com/newsroom/worldwide-cloud-infrastructure-services-Q2-2020

4. Docker documentation. URL: https://docs.docker.com/

5. Kubernetes documentation. URL: https://kubernetes.io/docs/home/

Автори статті

Щербина Ірина – кандидат технічних наук, доцент, Державний університет інформаційно-комунікаційних технологій, Київ, Україна.

Явор Денис - студент, Державний університет інформаційно-комунікаційних технологій, Київ, Україна.

Authors of the article

Shcherbyna Iryna - Candidate of Science (technic), associate professor, State University of Information and Communication Technologies, Kyiv, Ukraine.

Yavor Denys - student, State University of Information and Communication Technologies, Kyiv, Ukraine.