«КОНТЕЙНЕРИЗАЦІЯ»

АНАЛІЗ ПРОЦЕСІВ ВИКОРИСТАННЯ DOCKER ДЛЯ ПОБУДОВИ МІКРОСЕРВІСІВ

Львів – 2017

ЗМІСТ

1. АКТУАЛЬНІСТЬ РОБОТИ ТА ПОСТАНОВКА ПРОБЛЕМИ

Розробка серверних веб-додатків значно змінилася після дебюту Docker'а. Поява Docker спростила створення масштабованих та керованих додатків, побудованих за допомогою мікросервісів. Для кращого розуміння, що таке мікросервіс і як Docker допомагає їх реалізовувати, розглянемо приклад. Уявімо, що у команді веб-розробників працює три програмісти, які для створення одного й того ж додатку використовують різні операційні системи: macOS, Windows та Debian. Перш за все, кожне з перелічених середовищ вимагає окремих унікальних налаштувань. Крім того, розробники повинні встановити та налаштувати різні бібліотеки для своїх мов програмування. Неминучим є той факт, що бібліотеки та мови програмування конфліктуватимуть між різними середовищами. Додаймо ще сервери для тестування та виконання, після чого стає зрозумілим наскільки складно забезпечити однакові умови для всіх середовищ.

2. АНАЛІЗ НАУКОВИХ ДОСЛІДЖЕНЬ ГАЛУЗІ

Починаю чи з 2013 року, коли розробниками Docker Inc. презентована перша версія інструменту для віртуалізації рівня операційної системи, багато розробників почали адаптовувати Docker під власні потреби. Разом з цим проводиться низка наукових досліджень щодо визначення особливостей ефективного використання Docker. В наукові праці [1] розглянуті лабораторії віртуалізації, що дозволяють створювати віртуальні лабораторії на платформі Docker. Також, в оглянутій праці наведено порівняння архітектури віртуальних IT-сервісів на основі Docker та віртуальних машин, що дало підстави до розроблення рекомендацій по створенню віртуальних лабораторій на основі контейнерів під загальним управління Docker.

В роботі [2] проведено аналітичний огляд існуючих хмарних сервісів, а також апаратних, програмних і організаційних особливостей їх реалізації. В

роботі значна частка присвячено огляду Docker-інструментів, що надає можливість усвідомити його місце в технологіях створення хмарних сервісів.

Стаття [3] націлена на огляд та висвітлення переваг процесу віртуалізації, а також його впливу на використання обчислювальної потужності апаратної частини. Особлива увага в роботі приділена огляду переваг нової технології віртуалізації-контейнеризації.

З проведеного аналізу зрозуміло, що питання особливостей використання технології контейнеризації Docker розкрито досить широко, проте все ж таки актуальним залишається низка не розглянутих питань. Ціль нашої роботи полягає у доповненні вже існуючого багажу знань про технології контейнеризації в світлі особливостей використання Docker для побудови мікросервісів.

3. МЕТА РОБОТИ

Зважаючи на окреслену проблему та беручи до уваги результати проведеного аналізу наукових праць в роботі поставлено мету визначити особливості, переваги та недоліки використання Docker-технологій в процесі побудови мікросервісів.

4. ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

Описана проблема на початку статті, є актуальною за умови побудови монолітних програм та стає ще гострішою, якщо слідувати сучасній тенденції розробки додатків на базі мікросервісів.

Оскільки мікросервіси є автономними, незалежними прикладними блоками, кожен з яких виконує лише одну конкретну бізнес-функцію, їх можна розглядати як невеликі самостійні програми. Що станеться, коли створити десяток мікросервісів для одного додатку? І що, коли необхідно побудувати декілька мікросервісів з різними стеками технологій? Дуже швидко команда програмістів зіткнеться з багатьма перешкодами, оскільки розробники повинні

керувати ще більшою кількістю середовищ, ніж у традиційному монолітному додатку.

Однак, існує рішення – використання мікросервісів та контейнерів для інкапсуляції кожного такого сервісу і Docker допомагає керувати цими контейнерами. Docker – це інструмент контейнерування, побудований на основі контейнерів Linux, для забезпечення простого керування контейнерними додатками. Далі розглянемо переваги Docker для розробки мікросервісів на основі аналізу процесів його використання.

Контейнерування, як альтернатива віртуалізації, завжди могло удосконалити спосіб створення додатків, а Docker, як інструмент контейнерування, часто порівнюється з віртуальними машинами.

Як відомо, віртуальні машини (VM) створені для оптимізації використання обчислювальних ресурсів. За умови запуску декількох віртуальних машин на одному сервері та розгортання кожного екземпляру програми на окремій віртуальній машині, забезпечуватиметься стабільне середовище для кожного екземпляра. Однак, на жаль, масштабуючи програму, швидко виникне проблема пов’язана з продуктивністю, оскільки VM споживають багато ресурсів.

Наведена схема (рис.1) демонструє місце hypervisor у системі керування декількома операційними (гостьовими) системами на одному сервері ізолюючи їх одна від одної. У найпростіших випадках, hypervisor допомагає зменшити ресурси, необхідні для запуску декількох операційних систем.

Оскільки мікросервіси подібні до невеликих додатків потрібно розгортати мікроконтролери для власних віртуальних екземплярів (щоб забезпечити дискретне середовище). І слід погодитись з тим фактом, що присвоєння всієї віртуальної машини для розгортання лише невеликої частини додатку – не найефективніший варіант. Однак, за допомогою Docker можна зменшити втрати продуктивності та розгортати тисячі мікросервісів на одному сервері, оскільки контейнер Docker вимагає набагато менше обчислювальних ресурсів [4], ніж віртуальні машини.

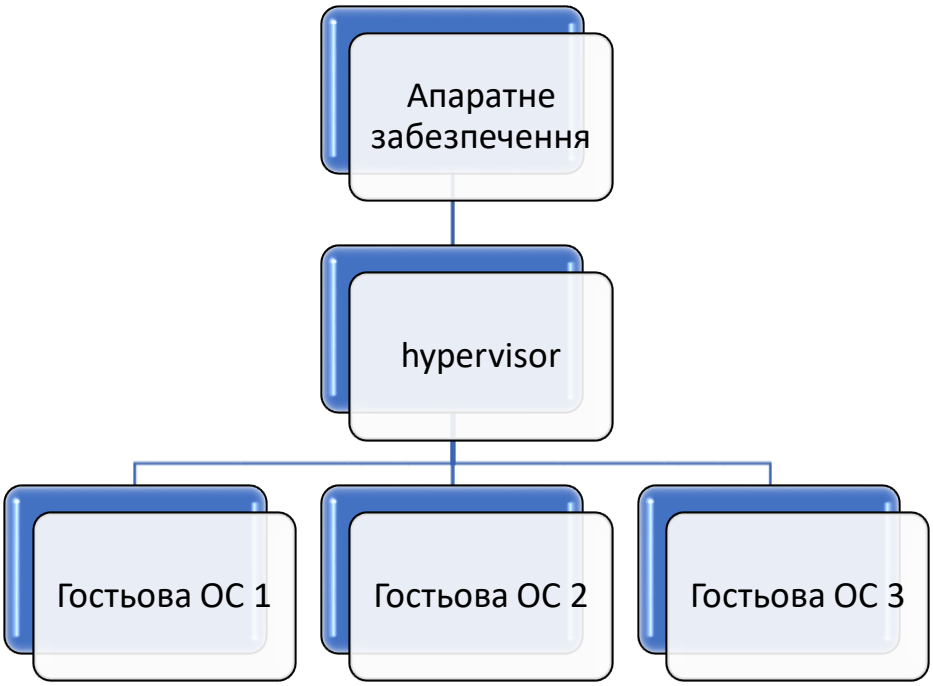

Рисунок 1 – Модель керування роботою VM з допомогою hypervisor

До цього моменту наш огляд зосереджувався на керуванні середовищами для одного додатка. Проте далі слід розглянути випадок розроблення двох різних проектів, або тестування двох різних версій того самого додатку. Конфлікти між версіями додатків або бібліотеками у цьому випадку неминучі, і підтримка двох різних середовищ на одній VM є складною задачею. Тут доречно задати питання: «Чим Docker кращий за віртуальні машини?». На відміну від роботи віртуальних машин, Docker не вимагає постійного створення нових середовищ, намагаючись уникнути конфліктів. Docker гарантує, що мікросервіси додатків працюватимуть у власних середовищах, які повністю відокремлені від операційної системи.

Завдяки Docker, перед командою розробників не виникає необхідності примусово працювати в ідентичних середовищах. Натомість, один розробник може створити стабільне середовище з усіма необхідними бібліотеками та мовами, після чого зберегти це налаштування в Docker Hub. Іншим розробникам залишається лише завантажити налаштування. Завдяки цій особливості Docker може заощадити багато часу.

Зважаючи на описані особливості використання Docker можна узагальнити основні переваги використання цієї технології:

- зменшення часу запуску (контейнер Docker запускається за лічені секунди, оскільки контейнер – це лише процес операційної системи. Запуск віртуальної машини з повною ОС може тривати декілька хвилин);

- швидке розгортання (не має необхідності створювати нове середовище. Членам команди веб-розробки потрібно лише завантажити образ Docker, щоб запустити його на іншому сервері);

- зручне керування та масштабування контейнерів (знищення та запуск контейнерів швидше ніж знищення і запуск віртуальної машини);

- використання обчислювальних ресурсів (можливість запуску більшої кількості контейнерів, ніж віртуальних машин на одному сервері);

- підтримка різних операційних систем (Docker працює під Windows, Mac, Debian та інші ОС).

Для кращої уяви про особливості застосування Docker в процесі розробки програм, що базуються на мікросервісах, слід розглянути їх архітектуру. Почнемо з огляду простого мікросервісу.

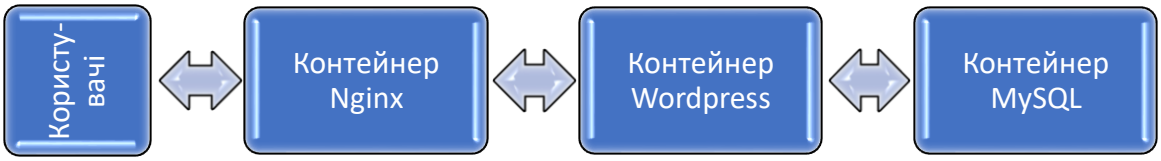

Рисунок 2 – Докеризований додаток (мікросервіс)

Програма (мікросервіс), зображена на рисунку, складається лише з трьох служб і дає змогу реалізувати блог для веб-сайту. Кожна зі служб, Nginx (веб-сервер), MySQL (база даних) і Wordpress (рушій блогу), інкапсулюється в контейнер. Архітектура Docker включає три основні компоненти – образ, контейнери та реєстри. Розглянемо кожен компонент окремо.

Образи (Docker Images) служать шаблонами для створення Docker-контейнерів. Контейнери, своєю чергою, є екземплярами образів та вважаються другим основним компонентом в архітектурі Docker.

Реєстри образів (репозиторії). Отже, як вже зрозуміло, образи та контейнери – два основних компоненти архітектури Docker. Але де вони

розташовані? Всі контейнери в наведеному прикладі побудовані зі стандартних образів, що збережені в Docker Hub (реєстрі). Реєстри наділені можливістю додавання образів та їх завантаження з реєстру.

Далі, для повноти аналізу процесів використання Docker, слід розглянути особливості управління сотнями або тисячами контейнерів Docker на кількох серверах.

Докер дозволяє розгортати мікросервіси один за одним на одному хості (сервері). Невеликий додаток, у прикладі вище, не потребує складного управління. Але що робити у випадку, коли додаток збільшується? Як розгорнути декілька контейнерів у кожному з декількох запущених серверів? Як масштабувати ці сервери «догори» та «донизу»? Для вирішення низки окреслених питань, Docker включає системи оркестрування контейнерами.

Контейнерна оркестрова система (container orchestration system) – це додатковий інструмент, який необхідно використовувати з Docker. До середини 2016 року Docker фактично не надав жодного конкретного способу керування додатками, побудованими з тисяч мікросервісів. Але тепер є Docker Swarm – вбудована платформа для оркестрування контейнерів тощо.

Docker Swarm дозволяє використовувати Docker CLI для запуску команд, тому можна легко ініціалізувати групи контейнерів, додавати та видаляти контейнери з цих груп.

В якості хмарних рішень, які можуть допомогти запустити докеровані додатки та оркеструвати контейнери, слід розглядати такі сервіси:

- *Google Cloud Platform* з підтримкою Kubernetes;
- *Amazon ECS*. Web-сервери Amazon дозволяють використовувати службу Elastic Compute Cloud (EC2) для запуску та обробки контейнерів Docker;
- *Azure Container Service* є хостинговим рішенням, подібним до Amazon ECS, і підтримує різні способи для оркестування докерованих додатків, включаючи Kubernetes, DC / OS з Mesos та Docker Swarm.

Отже, використання мікросервісів та контейнерів вважається ефективним сучасним способом створення масштабованих та керованих веб-

програм. У випадку не контейнеризації мікросервісу виникатимуть труднощі при їх розгортанні та керуванні. Саме тому, для уникнення будь-яких проблем під-час їх розгортання рекомендовано використання Docker. Окрім того, контейнерна система оркестрування надає можливість ефективно керувати комплексними докеризованими програмами.

5. ВИСНОВКИ

За результатами проведеного аналізу процесів використання Docker означено низку переваг, які полягають в оптимізації використання обчислювальних ресурсів, працездатності у відокремлених від ОС середовищах та можливості оркестрування комплексними програмами, що унеможливлює виникнення конфліктів між різними середовищами розробки під час побудови мікросервісів.

СПИСОК ЛІТЕРАТУРИ

1. Musaev A. A. The information infrastructure design of an educational organization using virtualization technologies / A. A. Musaev, S. M. Gazul, I. V. Anantchenko // Известия Санкт-Петербургского государственного технологического института (технического университета): Сб.науч.тр. Санкт-Петерберг : СПГТИ, 2014. – № 27(53). – С. 71-76.

2. Белоножко П. П. Свободные облачные аппаратно -программные платформы. Аналитический обзор / П. П. Белоножко, В. В. Белоус, Н. А. Куцевич, Д. А. Храмов // Науковедение: Интернет-журнал, 2016. – Том 8 (№6)

3. Королёв О.Л. Экономическая роль виртуализации в информационных системах / О. Л. Королёв, И. В. Гавриков, А. Д. Смирнов // International scientific review. Иваново: Олимп, 2017. - №5(36). – с. 69-70

4. Былина А. А. Автоматизация развертывания инфраструктуры для бизниса / А. А. Былина // Проблемы и перспективы современной науки: Материалы VI Республиканского научно-практического семинара молодых ученых. Минск: Минский инновационный университет, 2015. – С. 87-91.

5. Amazon Web Services (AWS) – сервисы облачных вычислений [Электронный ресурс]. – Доступ http://aws.amazon.com/ru/

6. Docker – Build, Ship, and Run Any App, Anywhere. [Электронный ресурс]. – Доступ http://aws.amazon.com/ru/