УДК 004.8:65.05:681.5 DOI: 10.31673/2518-7678.2019.044960

Ільїн О.Ю., д.т.н.; **Катков Ю.І.**, к.т.н., **Вергун Д.С.**, **Шашлов А.В.**

ОСОБЛИВОСТІ РОЗГОРТАННЯ МІКРОСЕРВІСНИХ ДОДАТКІВ ЗА ДОПОМОГОЮ СИСТЕМИ КЕРУВАННЯ КОНТЕЙНЕРАМИ

Ilyin O.Yu., Katkov Yu. I., Vergun D.C., Shashlov A.V. Features of development of microservice additives with the help of the container management system. The article briefly describes that, depending on the needs of the business, you can use a variety of technologies that can increase the efficiency of the company. The important thing is not to forget that every technology has its place and it is important to know about their existence, but it is painstaking to approach the use of one or the other, if you are working in a small business with a noticeably limited budget and not a large number of clients who visit your site, then it is possible you do not need to build a large failover and fast scalable system, but rather to weigh the risks and estimate the potential costs of maintaining and operating the large system and the possible costs of one hour of downtime per month / week; great likelihood that business will be cheaper "lie" in one hour, as such, they focus on offline business. The main thing is that not always modern and fashionable technologies are the key to solving all problems, and when choosing technologies, it is necessary to assess the risks, finances and needs of the company in the use of a particular system. Therefore, the article addresses the problem of finding opportunities to move quickly from a monolithic architecture to a microservice architecture of building computer systems using Docker containerization technology and orchestrating Kubernetes (K8s) containers to evaluate their implementation performance based on business needs. To do this, the advantages and disadvantages of Docker containerization technology and orchestration of Kubernetes containers (K8s) are analyzed. Consideration of this topic is quite relevant because nowadays the functioning of a business depends on the extent to which the existing software and hardware of the business management automation system will be able to adapt quickly to new challenges or threats in the new conditions of business functioning.

Keywords: software and hardware, automation and intellectualization of business management processes, monolithic, service oriented and microservice system architecture, Docker, Kubernetes

Ільїн О.Ю., Катков Ю.І., Вергун Д.С., Шашлов А.В. Особливості розгортання мікросервісних додатків за допомогою системи керування контейнерами. У статті коротко описано, що в залежності від потреб бізнесу можна застосовувати різноманітні технології, які дозволяють підвищувати ефективність роботи компанії. Головне не забувати, що кожна технологія має своє місце і важливо знати про їхнє існування, але кропітливо підходити до підбору використання тієї чи іншої, якщо ви працюєте на невеликому підприємстві з відчутно обмеженим бюджетом та невеликою кількістю клієнтів, які відвідують ваш сайт. Тоді можливо зовсім не потрібно вибудовувати велику відмовостійку та швидкомасштабовану систему, а краще зважити ризики й оцінити потенційні витрати на підтримку та експлуатацію великої системи та можливі витрати при одній годині простою на місяць/тиждень, існує величезна вірогідність, що для бізнесу буде дешевше “полежати” одну годину, так як, наприклад, вони зосереджені на офлайн бізнесі. Основним є те, що не завжди сучасні та модні технології є ключем до вирішення всіх проблем, а при виборі технологій потрібно оцінити ризики, фінанси та потреби компанії в використанні тієї чи іншої систем. Тому у статті розглядається проблема пошуку можливостей швидкого переходу від монолітної архітектури до мікросервісної архітектури побудови комп’ютерних систем за допомогою технології контейнеризації Docker та оркестровці контейнерів Kubernetes (K8s) для оцінки ефективності їх впровадження залежно від потреб бізнесу. Для цього виконується аналіз переваг та недоліків технології контейнеризації Docker, а також оркестровці контейнерів Kubernetes (K8s). Розгляд даної проблематики досить актуальний, тому що в даний час функціонування бізнесу залежить від того, наскільки існуючі програмно-апаратні засоби системи автоматизації управління бізнесом зможуть швидко адаптуватися до нових викликів або загроз у нових умовах функціонування бізнесу.

Ключові слова: програмно-апаратне забезпечення, автоматизація та інтелектуалізація процесів управління бізнесом, монолітна, сервісорієнтована та мікросервісна архітектура системи, Docker, Kubernetes

Ильин О.Ю., Катков Ю.И., Вергун Д.С., Шашлов А.В. Особенности развертывание микросервисних приложений с помощью системы управления контейнерами. В статье кратко описано, что в зависимости от потребностей бизнеса можно применять различные технологии, которые позволяют повысить эффективность работы компании. Главное не забывать, что каждая технология имеет свое место и важно знать об их существовании, но кропотливо подходить к подбору использования той или иной, если вы работает на небольшом предприятии с ощутимо ограниченным бюджетом и не большим количеством клиентов, посещающих ваш сайтю Тогда возможно совсем не нужно выстраивать большую отказоустойчивость и швидко масштабовану систему, а лучше взвесить риски и оценить потенциальные затраты на поддержку и эксплуатацию большой системы и возможные расходы при одной часов простоя в месяц / неделю, су огромная вероятность, что для бизнеса будет дешевле «полежать» один час, так как, например, они сосредоточены на офлайн бизнесе. Основным является то, что не всегда современные и модные технологии являются ключом к решению всех проблем, а при выборе технологий нужно оценить риски, финансы и потребности компании в использовании той или иной системы. Поэтому в статье рассматривается проблема поиска возможностей быстрого перехода от монолитной архитектуры к микросервисной архитектуре построения компьютерных систем с помощью технологии контейнеризации Docker и оркестровке контейнеров Kubernetes (K8s) для оценки эффективности их внедрения в зависимости от потребностей бизнеса. Для этого выполняется анализ преимуществ и недостатков технологии контейнеризации Docker, а также оркестровке контейнеров Kubernetes (K8s). Рассмотрение данной проблематики весьма актуален, так как в настоящее время функционирования бизнеса зависит от того, насколько существующие программно-аппаратные средства системы автоматизации управления бизнесом смогут быстро адаптироваться к новым вызовам или угрозам в новых условиях функционирования бизнеса.

Ключевые слова: программно-аппаратное обеспечение, автоматизация и интеллектуализация процессов управления бизнесом, монолитная, сервисориентованая и микросервисная архитектура системы, Docker, Kubernetes

Вступ

Сьогодні своєчасне адаптування існуючого програмно-апаратного забезпечення до нових вимог функціонування бізнесу складає проблему. Звісно, що такі велетенські компанії як Google, Netflix та інші можуть та мають слідкувати й адаптуватися під нескінченно підростаючий клієнто-потік, що несе за собою високе системне навантаження та потреби швидко масштабуватись. Але, якщо ви працюєте на невеликому підприємстві з відчутно обмеженим бюджетом та невеликою кількістю клієнтів, які відвідують ваш сайт, то можливо зовсім не потрібно вибудовувати швидко масштабовану і велику відмовостійку систему, а краще зважити ризики і оцінити потенційні витрати на підтримку та експлуатацію великої системи та можливі витрати однієї години простою на місяць/тиждень. Існує величезна вірогідність, що для бізнесу буде дешевше “полежати” одну годину, так як, наприклад, вони зосереджені на офлайн бізнесі. Тому у статті розглядається проблема пошуку можливостей швидкого переходу від монолітної архітектури до мікросервісної архітектури побудови комп’ютерних систем за допомогою технології контейнеризації Docker та оркестровці контейнерів Kubernetes (K8s) для оцінки ефективності їх впровадження залежно від потреб бізнесу.

Розгляд даної теми досить актуальний, тому що в даний час функціонування бізнесу залежить від того, наскільки існуючі програмно-апаратні засоби системи автоматизації управління бізнесом зможуть швидко адаптуватися до нових викликів або загроз у нових умовах функціонування бізнесу.

Система управління бізнесом складається з різних складових, серед яких програмно-апаратне забезпечення, яке визначає ступінь автоматизації та інтелектуалізації процесів управління бізнесом, а також відіграє провідну роль у функціонуванні бізнесу. Ця залежність вказує на критичність (уразливість) системи управління бізнесом під час появи нових вимог функціонування. Суть цієї проблеми полягає в наступному: відомо, що в загальному плані своєчасність адаптування програмно-апаратного забезпечення бізнесу залежить в першу чергу від архітектури побудови комп’ютерної системи (далі архітектури системи). Архітектурою комп’ютера називається його опис на деякому загальному рівні, що включає опис призначених для користувача можливостей програмування, системи команд, системи адресації, організації пам’яті тощо. Архітектура визначає принципи дії, інформаційні зв’язки

і взаємне з'єднання основних логічних вузлів комп'ютера: процесора, оперативної і зовнішньої пам'яті та периферійних пристроїв. Спільність архітектури різних комп'ютерів забезпечує їхню сумісність з точки зору користувача. Тому, якщо архітектурою системи вважати набір типів даних, операцій та характеристик кожного окремо взятого рівня (певних структурних компонентів пов'язаних між собою, які задають поведінку всієї системи), то від того як швидко може змінюватися архітектура системи буде залежати час адаптації.

Зрозуміло, адаптація архітектури системи, як процес пристосування до мінливих умов зовнішнього середовища, вимагає нових підходів до принципів побудови архітектури комп'ютерних систем. Дійсно, історично довгий час у комп'ютерних системах провідне місце займала так звана "монолітна архітектура", в якій, з одного боку, програмні аспекти (наприклад, введення і виведення даних, обробка даних, обробка помилок та інтерфейс користувача) виконували не архітектурно окремі компоненти, а компоненти, що пов'язані між собою. З іншого боку, застосування багатоядерних процесорів у "монолітній архітектурі" передбачає, що компоненти інтегровані разом до єдиної інтегральної схеми. При даному підході вся система являє собою моноліт, який фізично розташовується на єдиній машині, запускається в одному процесі та виконує по черзі всі бізнес-операції системи управління бізнесом, тобто має зниження продуктивності внаслідок поганого механізму для блокування записів і обробки файлів по локальній мережі. Тому монолітні архітектури побудови комп'ютерних систем погано працюють з декількома користувачами. Звідси стає зрозуміло, що "монолітна архітектура" погано адаптується до нових вимог щодо розподілення завдань між багатьма користувачами системи управління бізнесом. Тому процес адаптації вимагає пошуку нових підходів до побудови архітектури комп'ютерної системи. Відомі такі види побудови архітектури комп'ютерної системи: класична архітектура Фон Неймана, монолітна, сервісорієнтована та мікросервісна архітектури.

Класична архітектура (фон Нейман) – має пристрій керування, арифметично-логічний пристрій, пам'ять, пристрої вводу-виводу інформації, об'єднані за допомогою каналів зв'язку. В монолітній архітектурі функціонально помітні аспекти (наприклад, введення і виведення даних, обробка даних, обробка помилок та інтерфейс користувача), не архітектурно окремі компоненти, а компоненти, що пов'язані між собою. Монолітна архітектура може бути централізованою (це класична архітектура) або розподілена. Розподілена монолітна архітектура – це набір служб, які необхідно розгортати одночасно, щоб створити єдину функцію. Розподілена монолітна система часто набагато гірша, ніж централізована монолітна система, через складність і вартість координації декількох послуг. Сервісно-орієнтована архітектура (Service-oriented architecture, SOA) – це архітектурний шаблон програмного забезпечення, модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабко пов'язаних замінних компонентів, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами. Архітектурний стиль мікросервісів – це підхід, коли єдиний додаток будується як сукупність невеликих, самодостатніх, незалежних, не тісно пов'язаних сервісів, що спілкуються між собою за допомогою легких механізмів як то HTTP, gRPC, AMQP [1–3].

Необхідність розподілення завдань між багатьма користувачами системи управління бізнесом створило умови появи ідеї побудови SOA. Зміст SOA у тому, що програмне забезпечення складається з набору незалежних сервісів, які фокусуються на власній задачі та може являти собою розподілену систему, в якій відбувається обмін повідомленнями за певним протоколом, що дозволяє працювати будь-яким додаткам на будь-якому апаратному забезпеченні. Але недоліком SOA є обмеження протоколів обміну даними та розділу системи.

Для усунення цього недоліку був запропонований підхід удосконалення SOA, а саме: мікросервісна архітектура (MSA – Measurement System Analysis). Удосконалення полягає у тому, що за допомогою переходу до мікросервісів бізнес отримує можливість швидко щось змінювати, щоб швидше адаптуватися до змін бізнес-вимог. Метою мікросервісної

архітектури є допомога розробникам швидше, безпечніше та якісніше поставляти продукти. Відокремлені сервіси дозволяють виконувати ітерації швидко і з мінімальним впливом на іншу частину системи. Тобто мікросервіси, які не тісно пов'язані між собою, дають можливість проводити зміни в окремих програмно-апаратних засобах системи автоматизації управління бізнесом з більшою частотою ітерацій, мінімізуючи вплив змін на решту частин системи. Звідси головною відмінністю MSA є горизонтальне масштабування – це можливість вирішувати розподілення певного набору задач в умовах різних обставин між множиною комп'ютерних систем, що об'єднані в розподілену мережу. Тому виникає актуальне та своєчасне завдання дослідження проблеми адаптування системи управління за допомогою побудови додатків на базі мікросервісів. Одним з перспективних напрямків дослідження є дослідження методів спрощення розгортання мікросервісних додатків за допомогою системи керування контейнерами K8S та порівняння цих готових повністю керованих додатків у хмарі (Full managed version in the cloud). Тому метою даної роботи є дослідження побудови додатків на базі мікросервісів, огляд існуючих платформ для створення мікросервісів та їх порівняння.

Виклад основного матеріалу дослідження

Дослідження побудови додатків на базі MSA передбачає знаходження обмежень та доцільності використання цієї архітектури. Відомо, що MSA – це варіант SOA програмного забезпечення, орієнтований на взаємодію наскільки це можливо невеликих, слабо пов'язаних і легко змінюваних модулів – мікросервісів, що набув поширення в середині 2010-х років у зв'язку з розвитком практик гнучкої розробки та технології DevOps. Технологія DevOps (Development і operations) призначена для ефективної організації процесів створення та оновлення програмних продуктів і послуг. Вона заснована на ідеї тісного взаємозв'язку розробки та експлуатації програмного забезпечення. Ця методологія активної взаємодії програмістів-розробників з фахівцями інформаційно-технологічного обслуговування надає можливість взаємної інтеграції їхніх робочих процесів один в одного для забезпечення побудови якісного продукту.

Раніше було показано, що монолітна архітектура передбачає побудову системи як єдине ціле. Недоліком монолітної архітектури є те, що будь які зміни, навіть самі невеликі, потребують перебудови та повторного розгортання всього додатку для адаптації до нових умов функціонування бізнесу. Якщо такі умови змінюються досить часто, то з часом стає складніше зберігати хорошу модульну структуру, тому що зміни логіки одного модуля мають тенденцію впливати на код інших модулів. Крім того, монолітна архітектура не дозволяє масштабувати програми, тому що різні модулі мають конфліктні вимоги до ресурсів. Наприклад, один модуль може реалізовувати логіку обробки зображень з інтенсивним використанням процесору, а інший модуль може бути вимогливим до використання оперативної пам'яті. Однак, оскільки ці модулі розгортаються разом, то доведеться йти на компроміс з вибором апаратного забезпечення. Крім того, якщо потрібно масштабувати тільки один модуль, то разом з тим доводиться примусово масштабувати додаток для другого модуля, навіть якщо це не потрібно.

Для усунення цих незручностей мікросервісна архітектура дозволяє кожен мікросервіс розгортати окремо в наслідок того, що в архітектурі мікросервіса кілька слабко пов'язаних служб працюють разом. Кожен сервіс орієнтований на одну мету і має високий ступінь узгодженості поведінки й даних. Тому, якщо треба змінювати щось в одному з них, то можна розгорнути ці зміни не чіпаючи інших мікросервісів, які можуть продовжувати працювати. Звідси розосереджені завдання веб-проектів, які частіше розробляються як окремі сервіси, котрі можна розгортати та масштабувати окремо. Це стає можливо внаслідок реалізації принципів моделювання (проектування) мікросервісів у мікросервісній архітектурі (рис. 1) [4].

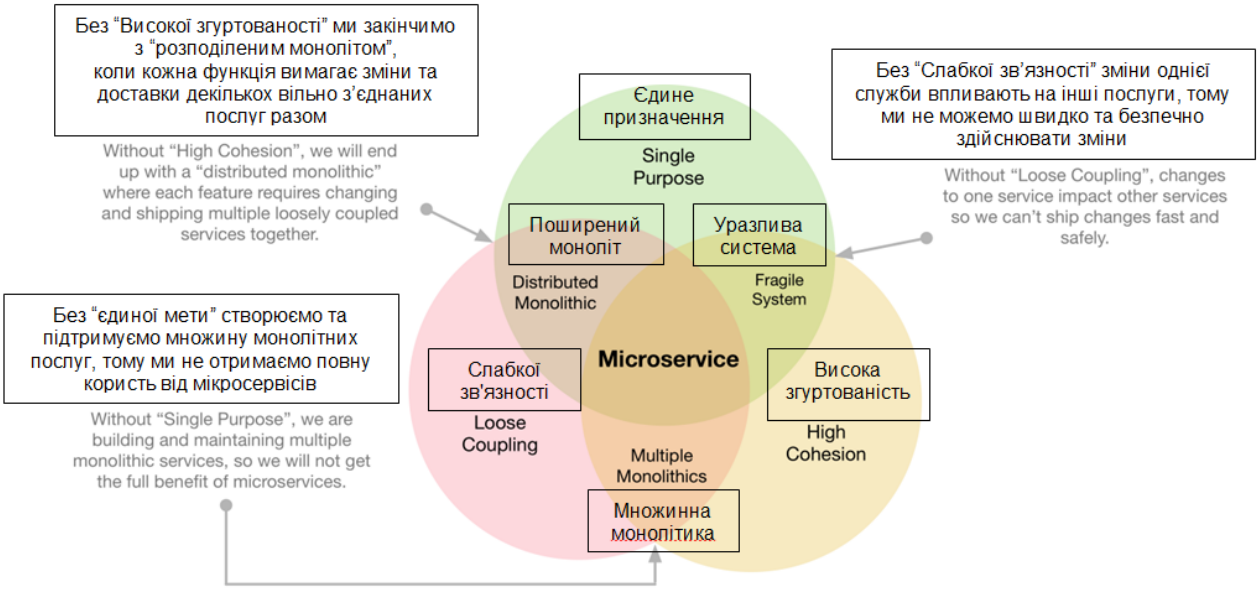

Рис. 1. Три принципи моделювання (проектування) мікросервісів [5]

Єдина мета – кожен сервіс має зосередитися на одній єдиній меті та робити це добре:

1. Слабкий зв'язок означає, що:
   a. Служби мало знають одна про одну;
   b. Зміна однієї послуги не повинна вимагати заміни іншої;
   c. Зв'язок між сервісами повинен відбуватися тільки через загальнодоступні сервісні інтерфейси;
2. Висока згуртованість – кожен сервіс об'єднує всі пов'язані поведінки і дані разом. Якщо нам потрібно створити нову функцію, всі зміни повинні бути локалізовані тільки для одного сервісу.

Важливо розуміти, що мікросервіс це [5]:

це не сервіс, який має невелику кількість рядків коду або виконує "мікро" завдання. Це помилка, яка походить від назви "мікросервіс". Мета мікросервісної архітектури – не створювати якомога більше дрібних сервісів. Послуги можуть бути складними і суттєвими, якщо вони відповідають вищевказаним трьом принципам;

це не послуга, яка постійно створюється з використанням нових технологій. Незважаючи на те, що архітектура мікросервіса дозволяє легше тестувати нові технології, це не є основною метою архітектури мікросервіса. Абсолютно нормально створювати нові сервіси з точно таким технологічним стеком, якщо під час проектування отримується вигода з роз'єднаних сервісів;

це не сервіс, який повинен бути побудований з нуля. Якщо у вас уже є добре спроектований монолітний додаток, то не потрібно створювати кожен новий сервіс з нуля. Можуть бути можливості витягти логіку безпосередньо з монолітного сервісу.

Таким чином, мікросервісну архітектуру додатків раціонально застосовувати коли інші види архітектури стають вузьким місцем: відносно продуктивності процесів; уповільнення розробки нових продукто-послуг; монолітний додаток ускладнює масштабування системи для конкретних завдань або ізоляцію проблем ресурсів для різних типів завдань; заважає випробувати нові технології (однією з основних переваг мікросервісної архітектури є те, що кожен сервіс може бути побудований з використанням різних технологічних стеків та інтегрований з різними технологіями); моноліт – не дозволяє вибрати кращий інструмент для роботи і, що більш важливо, зробити це швидким і безпечним способом.

Сьогодні багато відомих компаній вирішують проблему масштабування за допомогою MSA. Для прикладу застосування технології MSA – стисло розглянемо технологію Netflix

MSA Platform [6]. Компанія Netflix надає послуги потокового відео через Інтернет. Відомо, що якість передачі потокового відео залежить від пропускної здатності каналів мережі MPLS/IP. Ці канали створюють "тунель" між користувачем та сервером. Чим більше таких каналів складаються в "тунель", тим більша ймовірність перевищення 5% вимоги втрати пакетів у каналі потокового відео внаслідок помилок передачі, а це перевищення призводить до зупинки показу відеоконтенту у користувача. Тобто, зрозуміло, що необхідно скорочувати ці "тунелі". Цю проблему компанія Netflix вирішила шляхом розміщення відеоконтенту на серверах компанії Amazon, які розосереджені по всьому світу. Це дозволило суттєво скоротити відстань між сервером та користувачем. Для користування цим сервісом було створено програмне забезпечення, яке вирішувало з одного боку завдання масштабування управління доступом до відео контенту Netflix на найближчому сервері компанії Amazon, а з іншого передачу дозволу для зареєстрованих користувачів на сервері компанії Netflix, що знаходиться в Лос-Гатос, Каліфорнія. Тобто з технічної точки зору компанія Netflix створила бібліотеку відео додатків, які були розосереджені. Бібліотека Netflix доступна для перегляду з більшості сучасних інтернет-браузерів, також існують додатки для мобільних платформ Android, iOS і Windows Phone; приставок Google Chromecast й Apple TV, ігрових консолей Nintendo, PlayStation та Xbox; а також для багатьох моделей телевізорів з функцією Smart TV.

Складність програмного забезпечення Netflix MSA Platform вражає. Воно дозволяє: створювати потрібну конфігурацію серверів; виконувати реєстрацію сервісу та його відкриття; виконувати створення каналів з потрібною якістю передачі потокового відео в мережі MPLS; керувати шлюзами API; виконувати балансування завантаження за допомогою взаємодії між процесами (Inter-process communication, IPC); здійснювати моніторинг послуг у режимі реального часу; забезпечувати фіксацію простою та постачання нуля на час простою; виконувати тестування несправності, застосовувати методи інженерії хаосу для тестування програмного коду тощо. Впровадження технології Netflix MSA Platform є успішним для розробки нових програмних засобів створення мікросервісів. Сьогодні Netflix можна вважати першопрохідцем розвитку мікросервісів, і їхній підхід став об'єктом дослідження для багатьох інших компаній у всьому світі. Серед них Amazon, eBay, Walmart, Twitter, Spotify, Uber та Medium, Sound Cloud, Stripe, PayPal.

Таким чином, MSA – це архітектурний стиль, за яким єдиний додаток будується як сукупність невеличких сервісів, кожен з яких працює у своєму власному процесі та обмінюється даними з іншими. Ці сервіси будуються навколо бізнес-потреб і розгортаються незалежно з використанням зазвичай повністю автоматизованого середовища. Сервіси самі по собі можуть бути написані з використанням різних мов й технологій зберігання даних. Це створює наступні можливості: 1) відсутність обмежень на кількість існуючих сервісів, але кожен сервіс має працювати лише з однією бізнес-задачею; 2) використання стандартизованих протоколів передачі даних (наприклад, HTTP) для обміну інформацією між мікросервісами; 3) для спілкування з іншими мікросервісами кожен сервіс має своє API; 4) всі сервіси можуть бути написані на абсолютно різних мовах програмування та використовувати будь-які бібліотеки; 5) збереження даних децентралізоване, тобто кожен сервіс має свою власну базу даних.

На основі вказаних можливостей можна виділити основні переваги використання MSA: 1) існує досить мала зв'язність між основними компонентами системи та високе зчеплення коду в окремо взятому сервісі; 2) кожен сервіс розгортається незалежно від інших, що надає можливість динамічно вносити зміни та запускати окремо процеси; 3) можна запускати будь-яку кількість сервісів на окремих серверах (масштабування системи); 4) має відмовостійкість, тому що при виведенні з ладу одного сервісу інші можуть вірно працювати; 5) можна застосовувати різні мови програмування та технології; 6) розгалуження групи розробників для створення мікросервісів, які є відповідальними за власний сервіс.

До мінусів MSA можна віднести: 1) відносна складність розробки, прямо пропорційно залежить від кількості обраних мов програмування та фреймворків; 2) витрачаються додаткові ресурси на пересилання повідомлень між сервісами та на їхню серіалізацію та

десеріалізацію; 3) існують проблеми з версіонуванням; 4) відносно складне інтеграційне тестування; 5) складна система моніторингу; 6) сильно розростається кількість можливих збоїв; 7) додає додаткові складності.

Пропозиції щодо впровадження мікросервісної архітектури

Для вирішення питання переходу від монолітної архітектури до мікросервісної спочатку використовувались апаратні засоби віртуалізації такі як KVM, XEN, VMware ESXi тощо, але з часом з'явилися технології віртуалізації рівня операційної системи такі як Docker [7–8] та Kubernetes [9–12].

Відомо, що в апаратній віртуалізації базовий шар – гіпервізор. Цей шар завантажується на сервері та забезпечує взаємодію між апаратним забезпеченням сервера і віртуальними машинами. На відміну від програмної віртуалізації, при апаратній віртуалізації гостьові операційні системи управляються гіпервізором безпосередньо без участі хостової операційної системи. Апаратна віртуалізація набагато ефективніше програмної, тому що дозволяє за допомогою гіпервізора створювати керовані ізольовані гостьові системи. Гостьова система не залежить від архітектури хостової платформи і реалізації платформи віртуалізації.

Технологія Docker відноситься до віртуалізації рівня операційної системи [7–8]. Технологія Docker дозволяє упакувати зліпок стану системи, виконати його розробку і запустити. З появою технології Docker зацікавленість контейнерами вибухово зросла, тому що розгортати додатки виявилося настільки зручно, що технологію почали використовувати буквально всюди. Розглянемо переваги та недоліки технології Docker.

Переваги технології Docker:

1. Застосування технології Docker дозволяє уникнути конфліктів під час адаптації програмного забезпечення між різними додатками. Наприклад, є додаток на PHP (Personal Home Page Tools – "Інструменти для створення персональних веб-сторінок" – скриптової мови загального призначення, яка інтенсивно застосовується для розробки веб-додатків). Додаток на PHP використовує бібліотеку Image Magick для роботи з зображеннями, також цьому додатку потрібні специфічні налаштування php.ini, а сам додаток хоститься за допомогою Apache httpd. Але є проблема: деякі регулярні рутинні задачі реалізовуються запуском Python-скриптів із cron, і бібліотека, яку використовують ці скрипти, конфліктує з версіями бібліотек, що використовуються у цьому додатку. Технологія Docker дозволяє упакувати цей додаток разом із налаштуваннями, бібліотеками та HTTP-сервером в один контейнер, який обслуговує запити на 80-му порті, а рутинні задачі – перенести в інший контейнер. Все разом буде прекрасно працювати, і про конфлікт бібліотек можна буде забути.

2. Особливістю технології Docker для упаковки кожної програми є те, що типова композиція докерізованного додатку має шари ізоляції (рис. 2).

На рисунку 2 наведена типова композиція докерізованного додатку, розгорнутого в AWS. Прямокутниками тут позначені шари ізоляції. Найбільший прямокутник це фізична машина. Далі: операційна система (ОС) фізичної машини, амазонівский віртуалізатор, ОС віртуальної машини, докер-контейнер, ОС контейнера, JVM (Java Virtual Machine), Servlet-контейнер (якщо це веб-додаток), всередині якого вже міститься код вашої програми.

Недоліки технології Docker:

1. JVM – це, віртуальна машина в Java, яка є завжди. Додавання сюди ще додаткового Docker-контейнера, необхідне для упаковки кожної програми. Але, по-перше, часто не дає дуже помітної переваги, тому що JVM сама собою вже непогано ізолює від зовнішнього оточення, по-друге, не обходиться даром. Мова йде про те, що якщо використовуються, наприклад, дискові операції (використання процесора або доступ пам'яті), то технологія Docker додає буквально частки відсотка додаткового часу під час накладних витрат на виконання функції Overhead, але якщо виконується передача по каналам, то виникає

мережева затримка (network latency), яка є важливою характеристикою конструкції та продуктивності телекомунікаційної мережі, а ці затримки є цілком відчутними. Вони не гігантські, але в залежності від того, який у вас додаток, можуть зменшувати якість послуги (рис. 3);

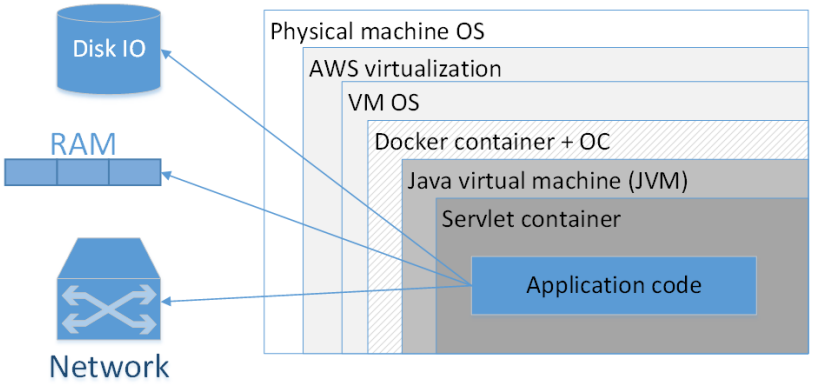

Рис. 2. Типова композиція докерізованного додатку [7]

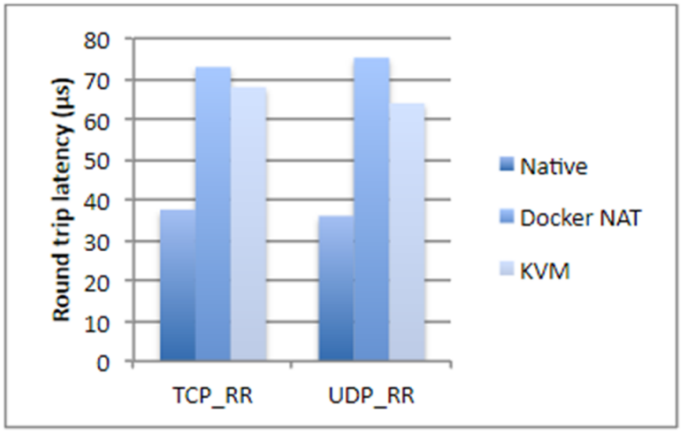

Рис. 3. Внесення затримок технологією Docker

2. Наступний недолік технології Docker – з'їдає додаткове місце на диску, займає частину пам'яті, додає час запуску додатку (start-up time);

3. Технологія Docker помітно ускладнює розбір проблем у мережевих протоколах, не лише ростуть затримки, але й змінюються всі таймінги.

Вказані недоліки впровадження технології Docker є некритичними для більшості систем. Але все ж виникають ситуації, коли пам'яті може не вистачати, і сумарний час старту системи, наприклад, що складається з декількох залежних сервісів, може стати досить великим. У загальному випадку будь-який додаток, чутливий до затримок у мережі в межах до 250–500 мс, краще не докерізувати.

Таким чином, у міру того як розробка додатків переміщується в сторону підходу на основі контейнера, стає важливо управляти ресурсами і контролювати їх. Хмарні сервіси вже дають швидкий доступ до великої кількості віртуальних машин, якими потрібно управляти. Тому без інструменту, який дозволяє запускати контейнери на безлічі хостів, масштабувати і виконувати балансування, вже не обійтися. Таким інструментом є технологія Kubernetes.

Технологія Kubernetes відноситься до віртуалізації рівня операційної системи [9–12]. Розберемося з рішенням, запропонованим Google як технологія Kubernetes. Технологія Kubernetes зараз є провідною платформою надійного планування робочих навантажень для відмовостійких додатків. Платформа Kubernetes, зараз дуже швидко розвивається. Вона призначена для управління контейнерними додатками Docker і пов'язаними з ними компонентами мережі та сховища.

Kubernetes – це кероване середовище, що спрощує розгортання додатків на основі контейнерів і управління ними. Основна увага в ній приділяється робочим навантаженням додатків, а не базових компонентів інфраструктури.

Переваги технології Kubernetes. З переваг даного підходу можна зазначити наступні:

1. Kubernetes як відкрита платформа дозволяє створювати додатки на будь-якій мові програмування, для будь-якої операційної системи, із застосуванням будь-яких бібліотек і служб повідомлень. З платформи Kubernetes можна інтегрувати будь-які засоби безперервної інтеграції та безперервного постачання (CI/CD) для планування і розгортання релізів (випусків) продуктів. Кластер Kubernetes розділяється на два компоненти (рис. 4): 1) головні вузли кластера надають базові служби Kubernetes і оркеструють робочі навантаження додатків; 2) вузли безпосередньо виконують робоче навантаження додатків;

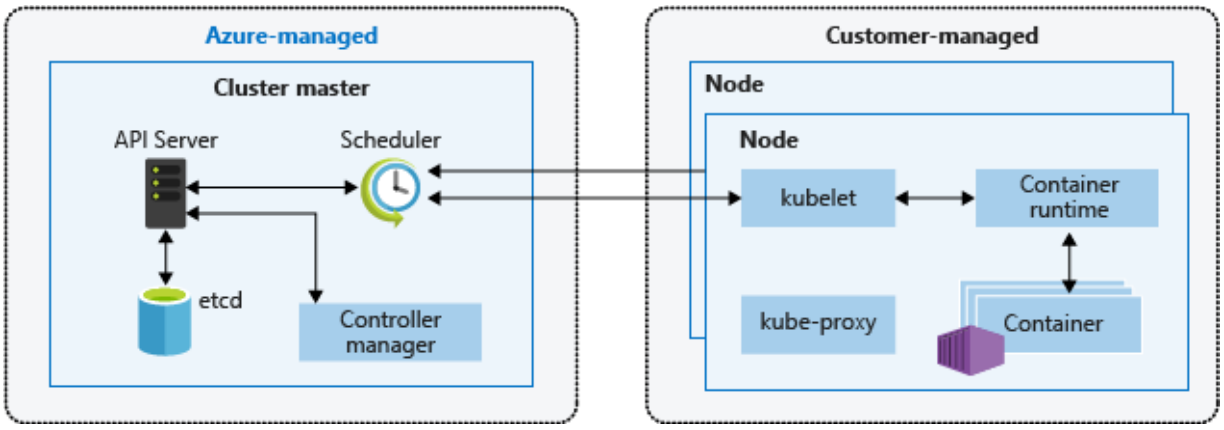

Рис. 4. Компоненти кластера Kubernetes [10]

2. У технології Kubernetes реалізується декларативний підхід до розгортання, підкріплений продуманим набором API-інтерфейсів для операцій управління. Зміст декларативного підходу – це опис того, що потрібно досягти, а не як. Можна створювати і запускати версії сучасних додатків, що масштабуються на базі мікрослужб, при цьому можливості платформи Kubernetes по оркестрації і управлінню використовуються для організації доступності до компонентів програми, що підтримують бізнес-процес. Технологія Kubernetes підтримує програми без відстеження стану і з відстеженням стану мікросервісної архітектури, що дуже зручно при адаптації додатків на базі мікрослужб;

3. Технологія Kubernetes сама себе відновлює в разі збоїв. Щоб запустити новий сервіс, то потрібний запуск контейнера може бути на будь-якій фізичній машині, він може бути будь-куди перенесеним. Система стає повторюваною. Практично це виглядає так, якщо у вас все розгорнулося і працює в тестовому оточенні, ви можете з доброю часткою впевненості сказати, що воно розгорнеться в точно таку ж систему і на процесі створення проекту чи творчого задуму (на продакшені). Нарешті, система починає підтримувати процедуру версіонування. Якщо щось пішло не так, ви можете дістати з розподіленої системи керування версіями Git'a стару конфігурацію і розгорнути все в точності, як було раніше;

4. Технологія Kubernetes є системою оркестрації. Технічне визначення оркестрації – це виконання визначеного робочого процесу: спочатку виконайте A, потім B, потім C. На відміну від цього, Kubernetes містить набір незалежних, компонуючих процесів управління, які постійно спрямовують поточний стан у потрібний стан. Не має значення, як дістатися від

А до С. Також централізований контроль не потрібен. Це призводить до того, що система простіша у використанні та більш потужна, міцна, еластична і розширювана.

Недоліки Kubernetes:

1. Kubernetes – це не традиційна, всеосяжна система PaaS (Platform as a service – платформа як послуга). Дійсно, PaaS розглядається як модель надання хмарних обчислень, при якій споживач отримує доступ до використання інформаційно-технологічних платформ: операційних систем, систем управління базами даних, зв'язного програмного забезпечення, засобів розробки і тестування розміщених у хмарних провайдерах. У свою чергу платформа Kubernetes працює на рівні контейнерів, а не на апаратному рівні, тому вона надає деякі загальноприйнятні функції, загальні для PaaS, такі як розгортання, масштабування, балансування навантаження, ведення журналів та моніторинг. Однак Kubernetes не є монолітним, і ці рішення за замовчуванням є необов'язковими та підключеними. Тому платформа Kubernetes забезпечує будівельні блоки для побудови інших платформ для розробників, але зберігає вибір та гнучкість користувача там, де це важливо;
2. Kubernetes не обмежує типи підтримуваних програм. Технологія Kubernetes має на меті підтримувати надзвичайно різноманітне навантаження. Якщо програма може працювати в контейнері, вона повинна чудово працювати в Kubernetes;
3. Kubernetes не розгортає вихідний код і не створює вашу програму. Поточні робочі процеси інтеграції, доставки та розгортання (CI/CD) визначаються культурою організації та уподобаннями, а також технічними вимогами;
4. Kubernetes не надає послуг на рівні додатків, таких як середнє програмне забезпечення (наприклад, шини повідомлень), рамки обробки даних (наприклад, Spark), бази даних (наприклад, mysql), кеші, а також розподілені файлові системи зберігання даних (наприклад, Ceph) як вбудовані сервіси. Такі компоненти можуть працювати на Kubernetes та/або отримувати доступ до них через додатки, що працюють на Kubernetes через портативні механізми, наприклад, Open Broker;
5. Kubernetes не диктує рішення для реєстрації, моніторингу та оповіщення. Він надає деякі інтеграції як доказ концепції та механізми збору та експорту показників;
6. Kubernetes не надає функції конфігурації мовою/системою (наприклад, jsonnet). Він надає декларативний API, на який можуть бути націлені довільні форми декларативних специфікацій;
7. Kubernetes не забезпечує і не приймає жодної комплексної системи конфігурації, обслуговування, управління або самолікування.

На основі виконаного аналізу можна запропонувати наступні пропозиції щодо впровадження мікросервісної архітектури:

1. Контейнери – це хороший спосіб зібрати та запустити програми. У виробничому середовищі потрібно керувати контейнерами, у яких запущені програми, і забезпечувати відсутність простоїв. Наприклад, якщо контейнер не працює, потрібно запустити інший контейнер. На допомогу приходить Kubernetes. Kubernetes пропонує основу для стійкого запуску розподілених систем. Він піклується про ваші вимоги до масштабування, відмову, схему розгортання тощо. Наприклад, платформа Kubernetes може легко керувати розгортанням каналів для вашої системи;
2. Kubernetes надає:

службу виявлення та балансування навантаження. Kubernetes може відкрити контейнер, використовуючи ім'я DNS або використовуючи власну IP-адресу. Якщо трафік до контейнера великий, Kubernetes може завантажувати баланс та розподіляти мережевий трафік так, щоб розгортання було стабільним;

оркестрацію зберігання. Kubernetes дозволяє автоматично монтувати на свій вибір систему зберігання даних, наприклад, локальні сховища, громадські постачальники хмар тощо;

автоматизовані розгортання та відкату для відновлення роботи. Можна описати потрібний стан для розгорнутих контейнерів за допомогою Kubernetes, і він може змінити фактичний стан на потрібний стан з контрольованою швидкістю. Наприклад, ви можете автоматизувати Kubernetes для створення нових контейнерів для розгортання, видалення існуючих контейнерів та прийняття всіх їхніх ресурсів до нового контейнера;

автоматичну упаковку у контейнер. Kubernetes дозволяє вказати, скільки потрібно процесорів та пам'яті (ОЗП) кожному контейнеру. Якщо в контейнерах вказані запити на ресурси, Kubernetes може приймати кращі рішення щодо управління ресурсами для контейнерів;

самолікування. Kubernetes перезавантажує контейнери, які виходять з ладу, замінює контейнери, вбиває контейнери, які не відповідають визначеним користувачем вимогам, не показує їх клієнтам, поки вони не будуть готові до обслуговування;

таємне та конфігураційне управління. Kubernetes дозволяє зберігати та керувати конфіденційною інформацією, наприклад паролями, маркерами OAuth та ключами ssh. Ви можете розгортати та оновлювати секрети та конфігурацію програми, не відновлюючи зображення контейнерів та не розкриваючи секрети в конфігурації стека.

Висновки

Таким чином, в статті було коротко описано, що в залежності від потреб бізнесу можна застосовувати різноманітні технології, які дозволяють підвищувати ефективність роботи компанії. Головне не забувати, що кожна технологія має своє місце і важливо знати про їхнє існування, але кропітливо підходити до підбору використання тієї чи іншої, якщо ви працює на невеликому підприємстві з відчутно обмеженим бюджетом та невеликою кількістю клієнтів, які відвідують ваш сайт, то можливо зовсім не потрібно вибудовувати велику відмовостійку та швидкомасштабовану систему, а краще зважити ризики й оцінити потенційні витрати на підтримку та експлуатацію великої системи, а також можливі витрати при одній годині простою на місяць/тиждень, існує величезна вірогідність, що для бізнесу буде дешевше "полежати" одну годину, так як, наприклад, вони зосереджені на офлайн бізнесі. Головним висновком є те, що не завжди сучасні та модні технології є ключем до вирішення всіх проблем, а при виборі технологій потрібно оцінити ризики, фінанси та потреби компанії у використанні тієї чи іншої систем.

Список використаної літератури

1. The Computer and the Brain (The Silliman Memorial Lectures Series) 3rd Edition by von Neumann, John (Author), Ray Kurzweil (Foreword), Yale University Press; 3 edition, 2012, 136 pages, ISBN-10: 0300181116, ISBN-13: 978-0300181111.

2. Computer Architecture: A Quantitative Approach 5th Edition by John L. Hennessy (Author), David A. Patterson (Author), Morgan Kaufmann; 5 edition (September 30, 2011), 856 pages, ISBN-10: 012383872X, ISBN-13: 978-8178672663.

3. Microservices vs. Service-Oriented Architecture. by Mark Richards. Publisher: O'Reilly Media, Inc.Release Date: April 2016. ISBN: 9781491975657.

4. *Ivan Zmerzlyi* Microservice architecture – [Електронний ресурс] – 2019 – Режим доступу: https://medium.com/@IvanZmerzlyi/microservice-architecture-f8a382291ff4. Дата доступу: жовтень 2019.

5. *Xiao Ma* Microservice Architecture at Medium – [Електронний ресурс] – 2019 – Режим доступу: https://medium.engineering/microservice-architecture-at-medium-9c33805eb74f. Дата доступу: жовтень 2019.

6. Netflix MSA Platform – MeltingCon – [Електронний ресурс] – 2019 – Режим доступу https://meltingcon.github.io/2018/assets/files/%EC%A0%95%EC%9C%A4%EC%A7%84.pdf. Дата доступу: жовтень 2019.

7. Docker Cookbook/ by Sébastien Goasguen. Copyright © 2016 Sébastien Goasguen. All rights reserved. Printed in the United States of America. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

8. Чи завжди потрібні Docker, мікросервіси та реактивне програмування? – [Електронний ресурс] – 2019 – Режим доступу: https://www.dataart.com.ua/news/chi-zavzhdi-potribni-docker-mikroservisi-ta-reaktivne-programuvannya. Дата доступу: жовтень 2019.

9. Kubernetes: Up and Running, 2nd Edition \ Dive into the Future of Infrastructure. By Brendan Burns, Kelsey Hightower, Joe Beda – Publisher: O'Reilly Media, Release Date: October 2019. Pages: 278.

10. Cloud Native DevOps with Kubernetes – by Justin Domingus, John Arundel. Publisher: O'Reilly Media, Inc. Release Date: March 2019. ISBN: 9781492040750.

11. Offcial Kubernetes documentation https://kubernetes.io/docs/concepts/overview/what-is-kubernetes. Дата доступу: жовтень 2019.

12. Ключевые концепции Kubernetes для службы Azure Kubernetes (AKS) – [Електронний ресурс] – 2019 – Режим доступу: https://docs.microsoft.com/ru-ru/azure/aks/concepts-clusters-workloads. Дата доступу: жовтень 2019.

Автори статті

Ільїн Олег Юрійович – доктор технічних наук, професор, професор кафедри Інженерії програмного забезпечення, Державний університет телекомунікацій, Київ, Україна.

Катков Юрій Ігорович – кандидат технічних наук, доцент, доцент кафедри комп'ютерних наук, Державний університет телекомунікацій, Київ, Україна.

Вергун Дмитро Сергійович – студент, Державний університет телекомунікацій, Київ, Україна.

Шашлов Андрій Вікторович – студент, Державний університет телекомунікацій, Київ, Україна.

Authors of the article

Ilyin Oleh Yuriiovych – doctor of Science (technic), professor, professor of Department of Software Engineering, State University of Telecommunications, Kyiv, Ukraine.

Katkov Yuriy Igorovich – candidate of Science (technic), associate professor, associate professor of Computer Science Department, State University of Telecommunications, Kyiv, Ukraine.

Verhun Dmytro Sergiyovich – student, State University of Telecommunications, Kyiv, Ukraine.

Shashlov Andrii Viktorovich – student, State University of Telecommunications, Kyiv, Ukraine.

Дата надходження в редакцію: 02.11.2019 р. Рецензент: д.т.н., проф. В.В. Вишнівський