УДК 62-503.5

Цитовцева А.С.,
Сопов О.О.

АНАЛІЗ ДОСТУПНОСТІ МІКРОСЕРВІСІВ НА БАЗІ СИСТЕМИ УПРАВЛІННЯ ТА ОРКЕСТРАЦІЇ КОНТЕЙНЕРІВ KUBERNETES

Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

Постановка проблеми в загальному вигляді

Перехід до мікросервісної архітектури вже триває. Однак, як важливий атрибут якості для послуг оператора, доступність залишається проблемою. Доступність - це нефункціональна характеристика, що визначається як обсяг відключення послуги протягом певного періоду [5]. Висока доступність досягається, коли система доступна принаймні в 99,999% випадків. Отже, загальний час простою, дозволений протягом одного року для високодоступних систем, становить близько 5 хвилин [3]. Деякі характеристики мікросервісів та контейнерів, такі як малі та легкі, природно сприяли б підвищенню доступності [1]. Kubernetes забезпечує зцілення своїх керованих мікросервісних програм. Можливість відновлення Kubernetes полягає у перезапуску невдалих контейнерів та заміні або переплануванні контейнерів, коли їхні господарі виходять з ладу. Цілюща здатність також відповідає за відновлення нездорових контейнерів, поки вони знову не будуть готові. Ці функції також би природно покращили доступність послуг, що надаються програмами, розгорнутими Kubernetes. Питання в тому, яка доступність надається цими програмами?

Аналіз останніх досліджень і публікацій

Протягом останнього десятиліття з'явилась загальна тенденція до міграції до хмари [1]. У цьому контексті архітектурний стиль мікропослуг [2] привернув значну увагу. На відміну від монолітного архітектурного стилю, мікросервісна архітектура вирішує проблеми побудови власних хмарних додатків, використовуючи переваги хмари [3]. Незважаючи на те, що цей архітектурний стиль готовий здійснити революцію в IT-галузі, до цього часу він отримував обмежену увагу з боку наукових кіл.

Мета статті

У цій роботі було оцінено програми з мікросервісною архітектурою з точки зору доступності, оскільки наша кінцева мета – забезпечити високу доступність мікросервісів. Як продовження для початкової установки в приватній хмарі та конфігурації Kubernetes за замовчуванням, було досліджено інші архітектури, конфігурації та проведено серію експериментів з Kubernetes, виміряно час відключення для різних сценаріїв відмов. Метою було дати відповідь на такі дослідницькі запитання:

- Який рівень доступності Kubernetes може підтримувати для своїх керованих мікропослуг виключно завдяки своїм властивостям відновлення?
- Який вплив додавання надмірності на доступність можна досягти за допомогою Kubernetes?
- Якої доступності може досягти Kubernetes за найефективнішої конфігурації?
- Як доступність, досягнута за допомогою Kubernetes, порівнюється з існуючими рішеннями?

Експерименти було проведено в конфігурації Kubernetes за замовчуванням, а також у найбільш реагуючій. Для кращого позиціонування та характеристики отриманих результатів було обрано порівняння з існуючим рішенням для управління доступністю, Framework Management Framework (AMF) [1], перевіреною послугою проміжного програмного забезпечення для управління високою доступністю (HA).

Виклад основного матеріалу

1. Розгортання контейнерів у кластері Kubernetes, що працює в загальнодоступній хмарі.

У цьому розділі буде розглянуто кластер Kubernetes, що складається з віртуальних машин, що працюють у загальнодоступній хмарі. Kubernetes працює на всіх віртуальних машинах і створює єдиний вигляд кластера. Одна з віртуальних машин вибрана в якості ведучої, і вона відповідає за управління вузлами. Оскільки ми стурбовані високою доступністю, нам слід розглянути кластер HA, що складається з більш, ніж одного ведучого. Однак така установка все ще є експериментальною та незрілою для Кубернетеса. Таким чином, ми вирішили піти лише з одним майстром і не допустити відмови з боку майстра. Для простоти додаток тут складається лише з одного мікросервісу. Шаблон pod для контейнерної мікросервіси, а також бажана кількість реплік включені до специфікації контролера розгортання, яка розгортається в кластері. Ми обговоримо два способи виставлення послуг у кластерах Kubernetes, що працюють у загальнодоступній хмарі.

Сервіс типу Load Balancer: архітектура для розгортання програм у кластері Kubernetes з використанням служби типу Load Balancer в загальнодоступній хмарі показана на рис. 1. На додаток до IP кластера, послуги типу Load Balancer мають зовнішню IP-адресу, яка автоматично встановлюється як IP-адреса балансира навантаження хмарного провайдера. Використовуючи цю зовнішню IP-адресу, яка є загальнодоступною, можна отримати доступ до подів ззовні кластера.

Ingress: Може бути більше однієї служби, яку потрібно піддавати зовнішньому впливу, і за принципом попереднього методу, для кожного сервіру потрібен один балансир навантаження. З іншого боку, вхідний ресурс Kubernetes може мати декілька сервісів у якості резервних копій та мінімізувати кількість балансувальних навантажень [3]. У кластері Kubernetes, що працює в загальнодоступній хмарі, контролер входу розгортається та виставляється службою типу Load Balancer [3]. Отже, запити на всі послуги, що надсилаються на балансир навантаження хмарного провайдера, отримуються контролером входу та перенаправляються на відповідну службу на основі правил, визначених у ресурсі входу

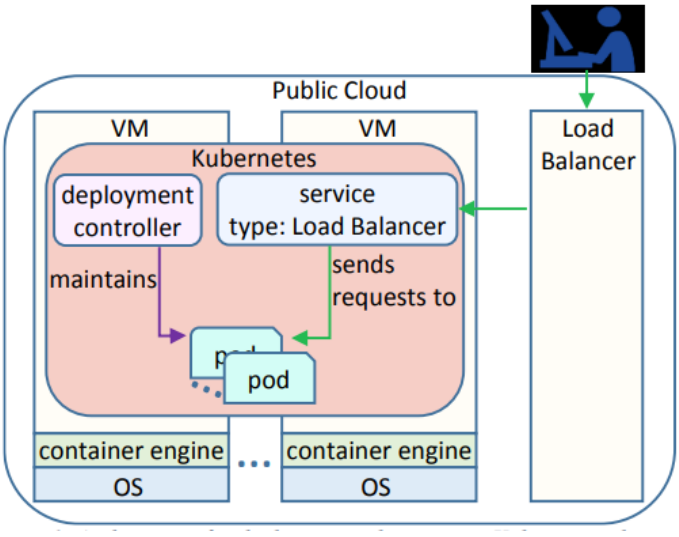

Рис. 1. Архітектура для розгортання програм на кластерах Kubernetes, що працюють у загальнодоступній хмарі

2. Розгортання контейнерів у кластері Kubernetes, що працює у приватній хмарі.

У цьому розділі буде описано налаштування експериментів, розглянуті сценарії відмов, а також показники доступності.

Ми встановили кластер Кубернетеса у приватній хмарі. Цей кластер складається з трьох віртуальних машин, що працюють у хмарі OpenStack. Ubuntu 16.04 - це ОС, що працює на всіх віртуальних машинах. Kubernetes 1.8.2 працює на всіх віртуальних машинах, а контейнерний двигун - Docker 17.09. Протокол мережевого часу (NTP) використовується для синхронізації часу між вузлами. Розгорнутою програмою є VideoLan Client (VLC). У кожному темплейті пода зазначено імедж, на якому встановлено VLC. Після розгортання пода на основі цього імеджа буде створений контейнер програми, який почне потокове передавання з файлу.

Kubernetes пропонує три рівні перевірки працездатності та механізмів для управління доступністю розгорнутих мікросервісів. По-перше, на рівні додатків Kubernetes гарантує, що програмні компоненти, що виконуються всередині контейнера, справні або за допомогою перевірки стану процесу, або заздалегідь визначених проб. В обох випадках, якщо Kubelet виявляє несправність, контейнер перезапускається. По-друге, на рівні поду Kubernetes відстежує відмови підсистеми та реагує відповідно до визначеної політики перезапуску. Нарешті, на рівні вузла, Kubernetes контролює вузли кластера через розподілені демони для виявлення відмов вузлів. Якщо вузол, що розміщує под, виходить з ладу, под перепланується на інший здоровий вузол. Що стосується цих рівнів перевірки працездатності, було зазначено три набори сценаріїв відмов. У першому наборі помилка програми спричинена помилкою процесу контейнера VLC. У другому наборі це пов'язано з відмовою процесу контейнера для подів, а в третьому наборі - через збій вузла. Для кожного набору експериментувалося з різними моделями надмірності, а також з конфігурацією Kubernetes за замовчуванням та найбільш адаптивною. Кожен сценарій повторювався 10 разів, і середнє значення вимірювань показано в Таблицях I - Таблиці V. Усі вимірювання, про які повідомляється в цьому документі, складаються у секундах.

Таблиця I – Експеримент з Кубернетисом – антинадлишкова модель та конфігурація за замовчуванням

Тригер відмови (одиниця: секунди)	Час реакції	Час репарації	Час відновлення	Час простою
Відмова контейнера VLC	0.716	0.472	1.050	1.766
Відмова контейнера пода	0.496	32.570	31.523	32.019
Відмова ноди	38.187	262.542	262.665	300.852

Час реакції: виявлено час між подією відмови, яку ми вводимо, і першою реакцією Кубернетеса, що відображає подію відмови.

Час репарації: Час між першою реакцією Кубернетеса та відновленням пода.

Час відновлення: Час між першою реакцією Kubernetes і часом, коли сервіс знову доступний.

Час простою: тривалість, протягом якої сервіс був недоступний. Він представляє суму часу реакції та часу відновлення

3. Експерименти, результати та аналіз

У цьому розділі представлено архітектури, експерименти, результати та аналіз для відповіді на питання дослідження, яке було поставлено у вступі.

Оцінка дій відновлення за допомогою конфігурації Kubernetes за замовчуванням для підтримки доступності

Рис. 2 показує архітектуру цих експериментів. Модель надмірності в даному випадку не є збитковою [2], а отже, кількість стручків у специфікації контролерів розгортання лише одна.

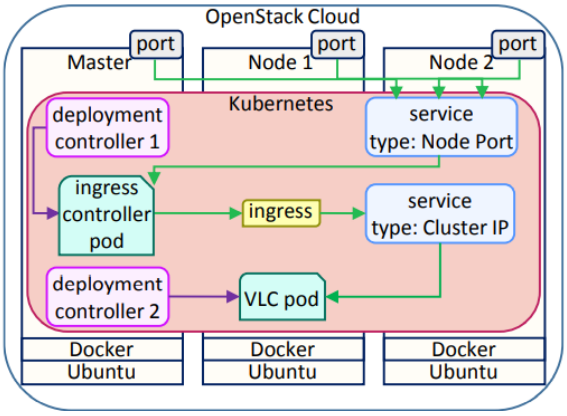

Рис. 2. Конкретна архітектура для розгортання програм за допомогою Kubernetes - модель надмірності без збитковості.

Збої в роботі через збій процесу контейнера VLC.

У цьому сценарії збій моделюється шляхом вимкнення процесу контейнера VLC з ОС. Коли контейнер VLC аварійно завершує роботу, Kubelet виявляє збій і приводить стручок до стану, коли він не отримуватиме нових запитів. У цей час, тобто час реакції, под вилучається зі списку кінцевих точок. Пізніше Kubelet перезапустить контейнер VLC, і відео почнеться з початку файлу. Цей час знаменує час ремонту. Час відновлення - це коли под знову потрапляє до списку кінцевих точок і готовий приймати запити.

Збої в роботі сервісу через збій процесу контейнера Pod.

Коли розгортається под, разом із контейнерами додатків, зазначеними в його шаблоні, створюється один додатковий контейнер, який є контейнером pod. Оскільки контейнер pod є процесом в ОС, можливо, він аварійно завершує роботу. У цьому випадку помилка моделюється шляхом вимкнення процесу контейнера підсистем з ОС. Коли процес контейнера для пода вбивається, Kubelet виявляє, що контейнер для пода відсутній, і це позначає час реакції. Коли новий под створено та запущено його контейнер VLC, відео почне транслюватися з початку файлу, і ми вважаємо под відремонтованим. Після цього Kubelet додасть новий под до списку кінцевих точок, і він буде готовий приймати нові запити, це означає час відновлення.

Збої в роботі служби через збій вузла.

У цьому випадку відмова вузла моделюється командою перезавантаження Linux на віртуальній машині, на якій розміщено под. Як вже згадувалося раніше, Kubelet відповідає за повідомлення звіту про стан вузла ведучим, і саме контролер вузла ведучого виявляє несправність вузла. Коли вузол, що розміщує под, виходить з ладу, він припиняє надсилання оновлень статусу ведучому, і майстер позначить вузол як не готовий після четвертого пропущеного оновлення стану. Цей час - час реакції. Коли вузол позначений як не готовий, підсистему VLC на вузлі планується припинити, а після його завершення буде створено новий. Час ремонту - це час запуску нового струму VLC і потокового передавання відео. Час відновлення - це коли под додається до списку кінцевих точок служби.

Результати та аналіз

Вимірювання та події цього набору експериментів наведені у таблиці I та на рис. 3 відповідно. На рис. 3 відмова контейнера VLC, контейнера pod або вузла, що розміщує Pod1, показана як перша подія. До цієї події IP-адреса Pod1 була в списку кінцевих точок, і сервіс був доступним. Після несправності сервіс стає недоступним. Однак, оскільки Kubernetes ще не виявив помилку, IP-адреса Pod1 залишається у списку кінцевих точок. Він вилучається зі списку кінцевих точок під час реакції.

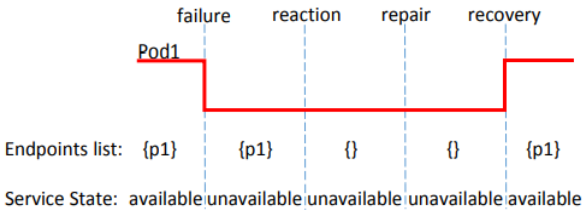

Рис. 3. Аналіз експериментів з Kubernetes з конфігурацією за замовчуванням та моделлю без збитковості – оцінка дій відновлення.

У даній архітектурі видалення IP-адреси Pod1 зі списку кінцевих точок як реакція Kubelet на збій контейнера VLC займає 0,716 секунди, а у випадку відмови контейнера pod - 0,496 секунди. Однак у разі відмови вузла, видалення IP od1 зі списку кінцевих точок, як реакція контролера вузла ведучого вимірювалася як 38,187 секунд. Причина полягає в тому, що за замовчуванням конфігурації Kubernetes, ведучому потрібно щонайменше 30 секунд, щоб виявити збій вузла. Оскільки Kubelet оновлює стан вузла кожні 10 секунд, а майстер дозволяє чотири пропущені оновлення стану, перш ніж позначити вузол як не готовий.

Час відновлення для всіх сценаріїв – це коли новий блок створюється знову і потокове відео починається знову. Як зазначається в таблиці I, час відновлення контейнера VLC або сценарії відмови контейнера пода суттєво відрізняються (0,472 секунди для першого та 32,570 секунд для останнього). Причина полягає в тому, що у випадку відмови контейнера pod, сигнал завершення надсилається до контейнера VLC, і Docker чекає 30 секунд, щоб він закінчився. Процес відновлення не розпочнеться, якщо контейнер VLC не буде припинено. Для сценарію виходу з ладу, як показано в таблиці I, час відновлення значно вищій. Причина полягає в тому, що за замовчуванням конфігурації Kubernetes, у разі відмови вузла, майстер чекає близько 260 секунд, щоб запустити новий под і відновити службу. Через такі високі терміни відновлення відключення обслуговування для сценаріїв відмови контейнера і вузла є значно вищими - 32,019 секунди та 300,52 секунди відповідно.

Висновки

В рамках даної роботи було представлено та порівняно архітектури для розгортання програм на базі мікросервісів у кластерах Kubernetes, розміщених у загальнодоступних хмарах. Проведено експерименти в хмарному середовищі, розглядаючи різні сценарії відмов, конфігурації, моделі резервування щоб оцінити Kubernetes з точки зору доступності програм, що працюють на Kubernetes. Було проаналізовано результати експериментів і виявлено, що дії для відновлення, що забезпечує Kubernetes недосттні для забезпечення доступності, особливо високої доступності. Наприклад, конфігурація Kubernetes за замовчуванням призводить до значного відключення у разі відмови вузла. Kubernetes можна переконфігурувати, щоб уникнути цього значного відключення.

Література

1. *Sam Newman.* Building Microservices: Designing Fine-Grained System. – O'Reilly, 2015. – 251 p.

2. Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. – O'Reilly, 2004. – 694 p.

3. *Pethuru Raj* Chelliah. Service Discovery and API Gateways. – Essentials of Microservices Architecture, 2019

4. *Eric Evans.* Domain-Driven Design: Tackling Complexity in the Heart of Software. – Addison-Wesley, 2003. – 560 p.

5. *Robert Martin*. Clean Code: A Handbook of Agile Software Craftsmanship. – Addison-Wesley, 2008. – 465 p.

Цитовцева А. С., Сопов О. О.

АНАЛІЗ ДОСТУПНОСТІ МІКРОСЕРВІСІВ НА БАЗІ СИСТЕМИ УПРАВЛІННЯ ТА ОРКЕСТРАЦІЇ КОНТЕЙНЕРІВ KUBERNETES

У статті досліджено проблеми оркестрації та проведено експерименти, щоб оцінити доступність, яку надає Kubernetes для керованих мікропослуг. Значну увагу приділено впливу додавання надмірності на доступність програм на базі мікросервісної архітектури та проведено експерименти з конфігурацією Kubernetes за замовчуванням, а також з найефективнішою. Перехід до архітектури мікропослуг триває і зараз. При такому підході система виділяється на менші модулі, які розроблені, розроблені та масштабовані окремо для побудови віртуалізованої функції. Але для постачальників функцій доступність залишається проблемою при переході до розгортання мікросервісів. Kubernetes - це рішення, яке має базу коду з відкритим кодом, воно визначає вибір розгорнутих частин, які разом надають інструменти для побудови, підтримки, масштабування та відновлення контейнерних функцій. Таким чином, Kubernetes приховує складність організації мікросервісів, щоб забезпечити управління їх доступністю. Для початку ми оцінюємо Kubernetes, використовуючи загальну конфігурацію з точки зору доступності в хмарних налаштуваннях. Представлені архітектури для державних та приватних хмар. Ми оцінюємо доступність, яка досягається цілющою силою Кубернету. Порівняльна оцінка була проведена за допомогою Framework Management Framework (AMF), тобто запропонованого механізму, що використовується в проміжному програмному забезпеченні для управління високою доступністю. Результати дослідження демонструють, що в деяких тестах відключення служби для програм, керованих Kubernetes, є значно високим.

Ключові слова: *мікросервіси, контейнери, оркестрація, докер, доступність.*

Tsytovtseva A., Sopov O.

ANALYSIS OF THE AVAILABILITY OF MICROSERVICE BASED ON THE MANAGEMENT AND ORCHESTRATION SYSTEM OF CONTAINERS KUBERNETES

The article explores problems of orchestration and executing test of performance to assess the availability of Kubernetes for supervised services. Considerable attention is paid to the sequence of managing exorbitance, on the availability of programs with on microservice architecture and tests have been conducted with the standard settings of Kubernetes and also with the most efficient. The move to a microservices architecture continues now. In this kind of approach the system is separeted to smaller modules which are designed, developed, and scaled separetely to build virtualized function. But, for function suppliers accessibility remains an issue in the transition to deploying microservices. Kubernetes is a solution which has code base an open source, it defines a selection of deployed parts that together gives instruments for building, supporting, scaling, and recovering containerized functions. In this way, Kubernetes hides the complexity of orchestrating microservices to provide management of their approachability. To start with, we estimate Kubernetes usinggeneral configuration in terms of approachability in cloud settings. It is presented architectures for public and private clouds. We are evaluating the availability achieved by the healing power of Kubernetes. A comparative assessment was executed using the Availability Management Framework (AMF), that is a suggested mechanism used in intermediate software for high availability management. The results of the research demonstrate that in some tests, disabling the service for applications managed by Kubernetes is significantly high.

Keywords*: microservices, containers, orchestration, docker, availability.*