DOI 10.31891/2307-5732-2022-313-5-260-264
УДК 004.056.5:621

МАЄВСЬКИЙ Ярослав
Хмельницький національний університет
ORCID ID: 0000-0002-5732-7093
e-mail: yarchik.mayevskij@gmail.com
ПРАВОРСЬКА Наталія
Хмельницький національний університет
ORCID ID: 0000-0001-6001-3311
e-mail: margana2000007@gmail.com

ПІДВИЩЕНЯ ЕФЕКТИВНОСТІ АВТОМАТИЗАЦІЇ МАСШТАБУВАННЯ МІКРОСЕРВІСІВ У СИСТЕМІ КЕРУВАННЯ КОНТЕЙНЕРИЗОВАНИМИ ЗАСТОСУНКАМИ KUBERNETES

Автоматичне масштабування контейнерів в системі Kubernetes відіграє важливу роль у опрацюванні вхідних запитів користувачів веб-застосунків. В цій статті проаналізовано етапи масштабування контейнерів, їхня ініціалізація і подальший запуск програмного забезпечення. Для досягнення низької затримки запитів користувача у випадку динамічного навантаження проводився аналіз процесу автоматичного масштабування контейнерів та факторів, які впливатимуть на процес масштабування. Маючи повне розуміння процесів та механізмів, за правилами яких відбувається масштабування, стало можливим розробка методу та стратегії для прибирання перепон, що сповільнюють сам процес автомасштабування і при цьому збереження необхідних властивостей від існуючого масштабування. Пришвидшення масштабування контейнерів, яке напряму буде впливати на швидкість веб-сервісів стає можливим саме через позбавлення затримки в автоматичному масштабуванні контейнерів.

З отриманих результатів дослідження сформовано метод оптимізації автоматичного масштабування контейнеризованих застосунків за рахунок позбавлення затримки під час холодного старту. Така затримка проявляється у випадку автомасштабуванням мікросервіса, де Kubernetes, як очікується, горизонтально масштабує контейнери шляхом створення додаткових реплік до необхідної кількості, щоб опрацювати необхідний трафік ззовні. Затримка, спричинена автомасштабувальником, впливає на час опрацювання запитів користувача веб-сервісу.

Ключові слова: мікросервіси, контейнери, автомасштабувальник, автоматичне масштабування контейнерів, Kubernetes, затримка, холодний старт.

MAYEVSKIJ Yaroslav, PRAVORSKA Natalya
Khmelnitskyi national university

IMPROVING THE EFFICIENCY OF AUTOMATION THE SCALING OF MICROSERVICES IN THE KUBERNETES CONTAINERIZED APPLICATION MANAGEMENT SYSTEM

Automatic container scaling in Kubernetes plays an important role in handling incoming requests from web application users. This article analyzes the stages of container scaling, their initialization and subsequent software launch. In order to achieve low latency of user requests in the case of dynamic load, the analysis of the process of automatic scaling of containers and the factors that will affect the scaling process was carried out. Having a full understanding of the processes and mechanisms by which scaling takes place, it became possible to develop a method and strategy for cleaning obstacles that slow down the autoscaling process itself and at the same time preserve the necessary properties of the existing scaling. Acceleration of scaling of containers, which will directly affect the speed of web services, becomes possible precisely because of the elimination of the delay in automatic scaling of containers.

The work considered scaling optimization using not only container pre-creation networks, but also the use of container images, which enable the sharing of linked libraries in memory and the extension of Kubernetes' declarative configuration management approach for parallel creation of multiple container instances.

Based on the obtained research results, a method for optimizing the automatic scaling of containerized applications by eliminating the delay during a cold start has been developed. This latency manifests itself in the case of microservice autoscaling, where Kubernetes is expected to scale containers horizontally by creating additional replicas to the required number to handle the required traffic from the outside. The delay caused by the autoscaler affects the processing time of the user's web service requests.

Keywords: microservices, containers, autoscaler, autoscaling of containers, Kubernetes, latency, cold start.

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

У контексті Kubernetes холодний старт – це затримка у процесі масштабування контейнеризованих застосунків, яка спричинена ініціалізацією контейнера з необхідним сервісом. Така затримка проявляється у випадку автомасштабування мікросервіса, де Kubernetes, як очікується, горизонтально масштабує контейнеризовану програму шляхом створення додаткових реплік потрібного контейнера до необхідної кількості, щоб опрацювати необхідний розмір трафіка ззовні.

Затримка автоматичного масштабування визначається багатьма факторами на різних етапах. В Kubernetes процес масштабування складається з кілька взаємопов'язаних фаз. Починається процес з того, що платформа визначає пік трафіку а також наступне планування набіру контейнерів аж до ініціалізації контейнера і виконання коду програмного забезпечення, після чого ми можемо констатувати успішну відповідь на запит

Щоб досягти низької затримки запитів користувача у випадку динамічного навантаження потрібно проаналізувати процес автоматичного масштабування контейнерів і визначити ключові фактори, які впливають на цей процес. Під час проведення дослідження детально вивчається проблема холодного запуску і аналізується процес масштабування. Тільки з повним розумінням процесів і механізмів масштабування є змога запропонувати метод або стратегії для усунення проблем, які сповільнюють процес автоматичного масштабування і в той же момент залишивши всі потрібні властивості від існуючого масштабування.

Позбавлення затримки в автоматичному масштабуванні контейнерів дозволяє пришвидшити масштабування контейнерів, що на пряму впливає на швидкодію веб-сервісів.

Аналіз останніх джерел

Дослідження засноване на офіційній документації платформи Kubernetes, яке детально описує етапи масштабування контейнеризованих застосунків а також описує в деталях горизонтальний і вертикальний автомасштабувальники. Для порівняння звичайних горизонтальних і вертикальних масштабувальників з вбудованим фреймворком від Kubernetes було проаналізована офіційна документація KNtive. Також в роботі авторами було розглянуто оптимізацію масштабування за рахунок використання нейронних мереж для попереднього створення контейнерів, використання образів контейнерів, які дозволяють спільно використовувати зв'язані бібліотеки в пам'яті і розширення декларативного підходу керування конфігурацією Kubernetes для паралельного створення кількох екземплярів контейнера. Ця проблема має різні варіанти и в різних контекстах і умовах. Наприклад, у безсерверній парадигмі проблема холодного запуску чітко визначається як час, витрачений на підготовку середовища виконання коду під час першого виклику функції. Порівняно з мікросервісами, безсерверна архітектура є більш модульним архітектурним стилем і має поставлені задачі щодо економії ресурсів, коли немає активного трафіку користувачів.

Виклад основного матеріалу

Холодний запуск у автоматичному масштабуванні контейнеризованих застосунків – це подія, яка описує момент значного збільшення навантаження на веб-застосунок, але в противагу програмному забезпеченні не достатньо ресурсів для опрацювання цих запитів. Тому, платформа керування контейнеризованими застосунками реагує на цю подію і розпочинає процес запуску додаткових контейнерів, які будуть опрацьовувати новий трафік. На цей процес витрачається багато часу, і якщо інші контейнери не звільнять свої ресурси – користувач буде очікувати час запуску додаткових контейнерів. Як результат – запити користувача довго опрацьовуються, або користувач взагалі не отримує відповіді на свої запити.

Процес холодного старту зображений на рисунку 1.

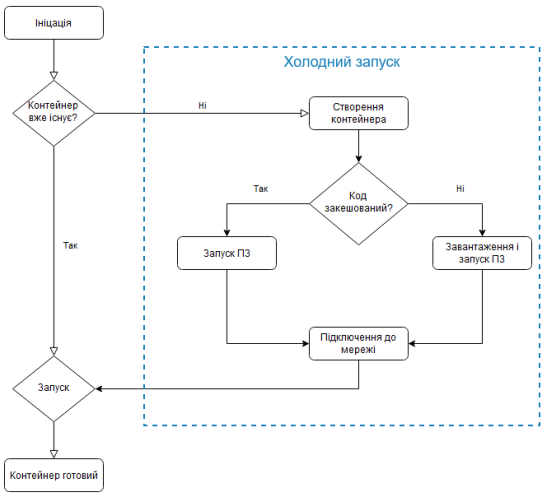

Рис. 1. Централізована система управління

Зазвичай, цей процес починається з того, що платформа перевіряє наявність середовища для запуску програмного забезпечення. Якщо таке існує – виклик буде опрацьований існуючими екземплярами – такий процес називається теплим стартом. В іншому випадку розпочинається процес холодного старту, де немає підготовлених контейнерів для обробки запита. Відповідно починається процес запуску контейнера, для подальшої роботи потребують код програмного забезпечення, який може бути закешований локально або завантажений. Після чого відбувається підключення мережі.

Загалом, для проходження усіх етапів холодного запуску може знадобитися кілька секунд – в залежності від програмного забезпечення або конфігурації контейнерів. Створений контейнер використовується багаторазово для обслуговування майбутніх запитів.

Підводячи підсумок, можна сказати, що типовий холодний запуск передбачає три етапи створення: створення контейнера, завантаження коду програмного забезпечення та розгортання мережі. Створення контейнера та розгортання мережі сприяють секундній затримці холодного запуску, тоді як час завантаження функціонального коду залежить від способів кешування коду.

Контейнери запуску програмного забезпечення

Для початку, проводиться порівняння трьох доступних середовищ для запуску контейнерів, а саме: Docker (v19.03.15), container (v1.4.4), CRI-O (v1.20.3). Вони відрізняються реалізацією Kubernetes Container Runtime Interface (CRI).

Для Docker потрібен окремий процес dockershim, щоб підтримувати разом Docker і kubelet, а container містить ту саму функціональність dockershim, що й плагін CRI, у своєму коді. CRI-O реалізує CRI за дизайном і може безпосередньо взаємодіяти з kubelet без додаткових витрат. Щоб порівняти їх продуктивність, вимірюється час запуску контейнера під час виконання різних контейнерів. Час запуску контейнера — це тривалість від того моменту, як kubelet отримав контейнеру щойно створений контейнер, призначений йому.

На рисунку 2 показана діаграма, на якій CRI-O має найкращу продуктивність з точки зору затримки.

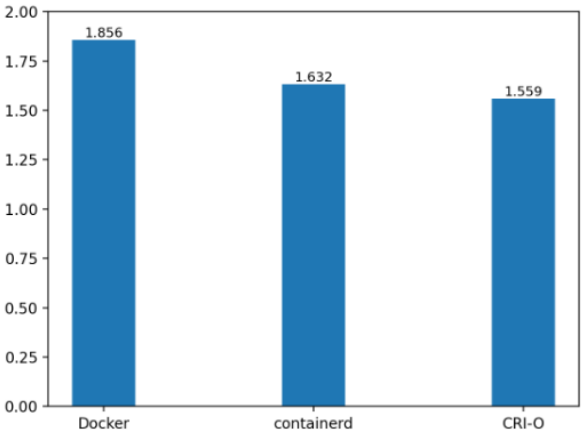

Рис. 2. Діаграма затраченого часу на запуск контейнера в різних умовах

Налаштування мережі

Плагіни мережевого інтерфейсу контейнера (CNI) використовуються в Kubernetes для підтримки роботи в кластерній мережі. CNI плагіни відповідають за підключення мережі до контейнера під час його ініціалізації. Такі плагіни відрізняються за собою механізмами реалізації, протоколами і мережевою моделлю. Порівняємо 5 поширених рішень, а саме: Cilium, Flannel, Calico, Weave Net, Kube-router. Діаграма затраченого часу в секундах на запуск контейнера з різними плагінами CNI представлена на рисунку 3.

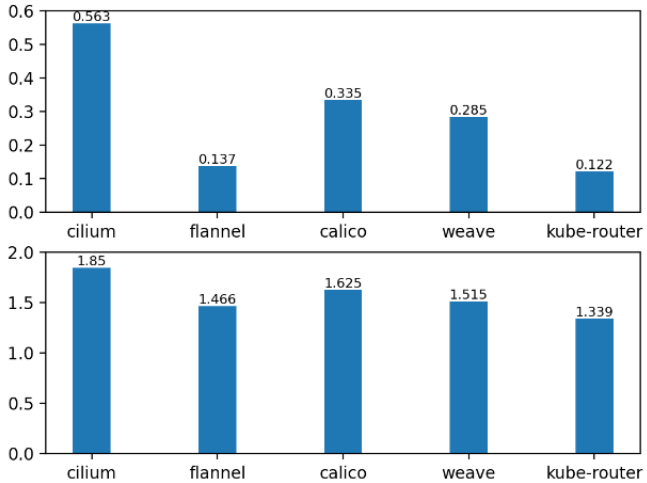

Рис. 3. Діаграма затраченого часу в секундах на запуск контейнера з різними плагінами CNI

Згідно з діаграмою, Flannel і kube-router витрачають найменше часу, тоді як Cillium відносно повільніший за інших. Однак, загальна різниця між плагінами не значна. У результаті можна зробити висновок, що різні CNI плагіни можуть збільшити швидкодію і зменшити час запуску контейнера. Але варто розуміти, що різні плагіни мають інші параметри як час на запуск, значною мірою вони відрізняються в пропускній здатності вводу-ввиводу запитів, а не часу на запуск.

Розмір кластера

Розмір кластера є однією з гіпотетичною проблем у швидкодії запуску контейнерів. Але, планувальник Kubernetes має механізм, який аналізує дані найменшого за розміром вузла в контейнері. Однак, незалежно від розміру кластера, планування модуля працює на шкалі часу в мілісекунди; таким чином, вплив розміру кластера на продуктивність масштабування модуля має бути незначним. Н рисунку 4 представлена діаграма, яка відображує затрачений час на запуск контейнера з різними за розміром кластерами.

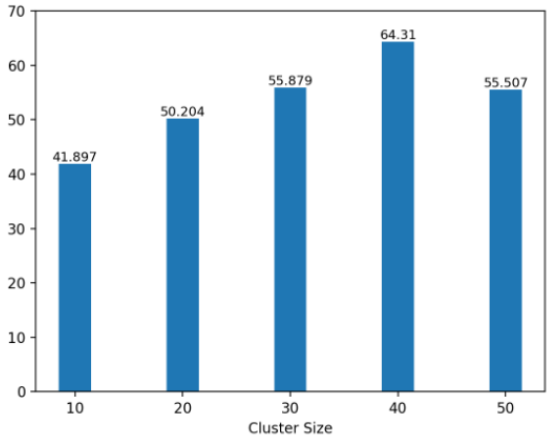

Рис. 4. Діаграма затраченого часу в мікросекундах на запуск контейнера з різними розмірами кластера

Мова програмування

Важливим фактором у запуску контейнерів є програмне забезпечення, а саме мова якою воно написане. Таким чином, ми порівнюємо час на запуск мінімального програмного забезпечення в умовах платформи Kubernete з контейнером Docker. Для прикладу, було реалізоване мінімальне програмне забезпечення на шістьох різних мовах, яке виводить в консоль інформацію, що воно запустилось. Вибрані мови: Go 1.16, Java 11.00, C# 3.1, JavaScript 14.17, PHP 7.3, Python 3.9. Результати запусків наведені на рисунку 5.

Результат дослідження показав, що мова програмування теж впливає на час запуску контейнерів.

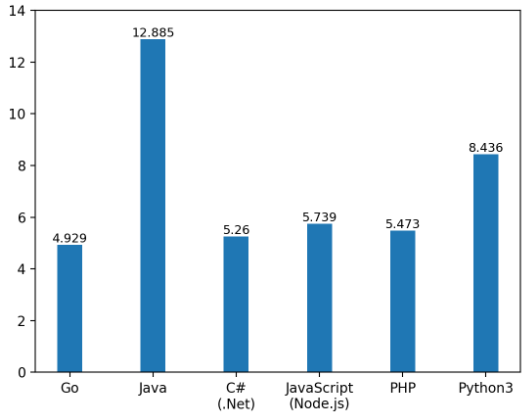

Рис. 5. Діаграма використаного часу для запуску мінімального ПЗ

З діаграми добре видно, що більший час було затрачено для запуску програмного забезпечення написаного мовою Java, а лідером швидкості з запуску ПЗ, стало написане мовою Go.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі

Проведені в дослідженні експерименти надали деяку корисну інформацію, яка стане в пригоді при розробці методів, направлених на покращення автоматичного масштабування контейнерів. У проведеному дослідженні було проведено аналіз та порівняння різних факторів, що впливають на ефективність масштабування контейнеризованих застосунків в системі керування контейнеризованими застосунками. Було проаналізовано вплив інструментів контейнеризації на час запуску сервісів, в результаті чого, отримані дані, які вказують, що CRI-O є найшвидшим інструментом контейнеризації. Також детально проаналізовано вплив плагінів мережевого інтерфейсу контейнера, і доведено, що Flannel і kube-router витрачають найменше часу, тоді як Cillium, який використовується з коробки – відносно повільніший за інших. Однак, загальна різниця між плагінами не значна. Також корисною інформацією є те, що розмір кластера не має впливу на час запуску контейнера, а мови програмування значно впливають на час запуску програмного забезпечення в контейнерах. Всі отримані дані будуть використані в наступній розробці як самого методу,

так і стратегії для прибирання перепон, що сповільнюють сам процес автомасштабування і при цьому збереження необхідних властивостей від існуючого масштабування.

Література:

1. D. Mao, Z. Hao, F. Wang, and H. Li, State-of-the-practice Autoscalers: A case study in Shandong Province, China, Sustainability, vol. 10, no. 2, p. 3149, 2018.
2. B. Bershad, R. P. Draves, and A. Forin. Using microbenchmarks to evaluate system performance. In [1992] Proceedings Third Workshop on Workstation Operating Systems, pages 148–153. IEEE, 1992
3. J. Dieltjens, E. Heydari Beni, E. Truyen, B. Lagaisse – Reducing cold start during elastic scaling, KU Leuven, Belgium, 2019.
4. Kubernetes Documentation – [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/home/

Надійшла/Paper received : 27.09.2022 р. Надрукована/Printed :01.11.2022 р.