УДК 621.301
doi:10.18720/SPBPU/2/id24-557

Долматов Роман Александрович [1],
аспирант;
Сараджишвили Сергей Эрикович [2],
доцент, канд. техн. наук, доцент

ОПТИМИЗАЦИЯ МАСШТАБИРУЕМОСТИ И ОТКАЗОУСТОЙЧИВОСТИ МИКРОСЕРВИСОВ С ПРИМЕНЕНИЕМ KUBERNETES

[1,2] Россия, Санкт-Петербург,
Санкт-Петербургский политехнический университет Петра Великого,
[1] d_roman.kst@mail.ru, [2] ssaradg@yandex.ru

Аннотация. В работе исследуются методы оптимизации масштабируемости и отказоустойчивости микросервисов. Описывается процесс настройки автоматического масштабирования приложения на основе пользовательских метрик, собранных с помощью Prometheus. Для демонстрации предложена практическая реализация, включающая развертывание контейнеров приложения и нагрузки. Применение этих технологий позволяют обеспечивать динамическое масштабирования приложений от текущей нагрузки, что способствует повышению отказоустойчивости системы.

Ключевые слова: Kubernetes, микросервисы, масштабируемость, отказоустойчивость, автоскейлинг, Prometheus, docker.

Roman A. Dolmatov [1],
Postgraduate Student;
Sergey E. Saradgishvili [2],
Associate Professor, Candidate of Technical Sciences

OPTIMIZATION OF SCALABILITY AND FAULT TOLERANCE OF MICROSERVICES WITH USING KUBERNETES

[1,2] Peter the Great St. Petersburg Polytechnic University, St. Petersburg, Russia,
[1] d_roman.kst@mail.ru, [2] ssaradg@yandex.ru

Abstract. The paper examines methods for optimizing the scalability and fault tolerance of microservices. Describes the process of configuring the automatic scaling of the application based on user metrics collected using Prometheus. A practical implementation is proposed for demonstration, including the deployment of application containers and loads. The use of these technologies allows for dynamic scaling of applications from the current load, which helps to increase the fault tolerance of the system.

Keywords: Kubernetes, microservices, scalability, fault tolerance, auto scaling, Prometheus, docker.

Введение

Микросервисная архитектура позволяет создавать гибкие и масштабируемые приложения, где каждая функциональная часть приложения работе в отдельном сервисе. Однако для поддержания высокой доступности и производительности важно правильно настроить масштабируемость и отказоустойчивости. Kubernetes является мощным инструментом для управления контейнизированными приложениями, который предоставляет возможность для автоматического масштабирования и мониторинга сервисов.

1. Постановка задачи и описание предметной области

– Настройка среды: подготовка Kubernetes кластера с использованием оператора и адаптера, а также Redis без Sentinel.

– Создание и развертывания контейнеров: разработка образов для приложения и инструмента нагрузки, сборка и загрузка.

– Настройка мониторинга метрик: конфигурация адаптера Prometheus для сбора метрик.

– Визуализация метрик: настройка дашборда в Grafana для мониторинга состояния и производительности.

– Тестирование автоскейлинга: запуск теста нагрузки на приложение с помощью инструмента нагрузки и анализ результатов через интерфейс kubectl и Grafana.

В сфере разработки программного обеспечения масштабирование делится на категории.

Горизонтальное масштабирование – изменение числа копий (реплик) приложения. Речь идет либо о числе Pod'ов определенного приложения, либо о числе узлов в кластере, на которых размещаются приложения.

Вертикальное масштабирование – изменение количества ресурсов отдельно взятого экземпляра. В случае с приложением это означает изменение запросов и/или лимитов на ресурсы для его контейнеров. Если речь об узлах кластера, это обычно подразумевает изменение количества доступных ресурсов процессора и памяти.

В системах, которые нуждаются в динамическом масштабировании (в том числе испытывающих частые и существенные колебания нагрузки), по возможности следует отдавать предпочтение горизонтальному масштабированию. Вертикальное масштабирование ограничено самым производительным сервером, который вам доступен. К тому же такое «вертикальное» повышение емкости требует перезапуска приложения. Даже в виртуальных окружениях, поддерживающих динамическое масштабирование, Pod'ы приходится перезапускать, так как запросы и лимиты на ресурсы в настоящее время нельзя обновлять на лету. Для сравнения, при горизонтальном масштабировании экземпляры приложениях не нуждаются в перезапуске, а динамическое увеличение емкости происходит за счет добавления реплик.

Вопрос автомасштабирования особенно важен для сервис ориентированных систем. Одно из преимуществ разделения приложения на отдельные компоненты – возможность их независимого масштабирования. Отделение веб-приложений от реляционных баз данных и их независимое масштабирование стало общепринятой практикой. В микросервисных архитектурах можно пойти еще дальше. Например, у веб-сайта предприятия может быть два сервиса, один из которых обслуживает его Интернет-магазин, а другой отвечает за статьи в блоге. Во время рекламных акций Интернет-магазин можно масштабировать, а сервис блога можно оставить без изменений, так как на него эти акции не влияют. Имея возможность масштабировать сервисы независимо друг от друга, вы можете более эффективно задействовать инфраструктуру, которую используют приложения. Но в то же время повышаются накладные расходы, связанные с масштабированием множества отдельных приложений. Автоматизация этого процесса становится очень важной. На определенном этапе без нее уже не обойтись. Автомасштабирование хорошо подходит для небольших, легковесных приложений с крошечными образами и коротким временем запуска. Если скачивание образа на заданный узел и запуск соответствующего контейнера происходит быстро, приложение может оперативно реагировать на события масштабирования. Это существенно облегчает изменение емкости. А вот приложения с образами, занимающими гигабайт, и скриптами запуска, время выполнения которых исчисляется минутами. куда менее приспособлены к реагированию на изменения нагрузки.

2. Моделирование системы

Horizontal Pod Autoscaler (HPA) – это самое распространенное средство автомасштабирования для платформ на основе Kubernetes. Его поддержка встроена в Kuberetes в виде одноименного ресурса и контроллера, входящего в состав kube-controller-manager, где используется потребление ресурсов процессора или памяти в качестве метрик для автомасштабирования своих приложений.

В данном случае, чтобы у HPA был доступ к метрикам Pod'ов, понадобится Kubernetes Metrics Server. Это сервер, который извлекает из kubelet информацию об использовании процессора и памяти контейнерами кластера и делает ее доступной посредством API-интерфейса метрик в ресурсах PodMetrics. Metrics Server задействует слой агрегации в API-интерфейсе Kubernetes. Запросы ресурсов в группе API и версия будут направляться этому серверу. На рис. 1 показано, как данные компоненты выполняют эту функцию. Prometheus Adapter настраивается для сбора определенных пользовательских метрик из базы данных Prometheus.Это включает в себя определение, какие метрики следует собирать, и какие запросы к Prometheus нужно делать для получения этих метрик. Horizontal Pod Autoscaler (HPA) использует собранные пользовательские метрики для

принятия решений о масштабировании приложения. На основе установленных правил и пороговых значений HPA определяет, нужно ли увеличить или уменьшить количество реплик приложения. Если метрики превышают установленные пороги (например, количество запросов в секунду превышает определенное значение), HPA автоматически увеличивает количество реплик приложения. Это позволяет распределить нагрузку на большее количество экземпляров приложения, обеспечивая требуемую производительность и отзывчивость.

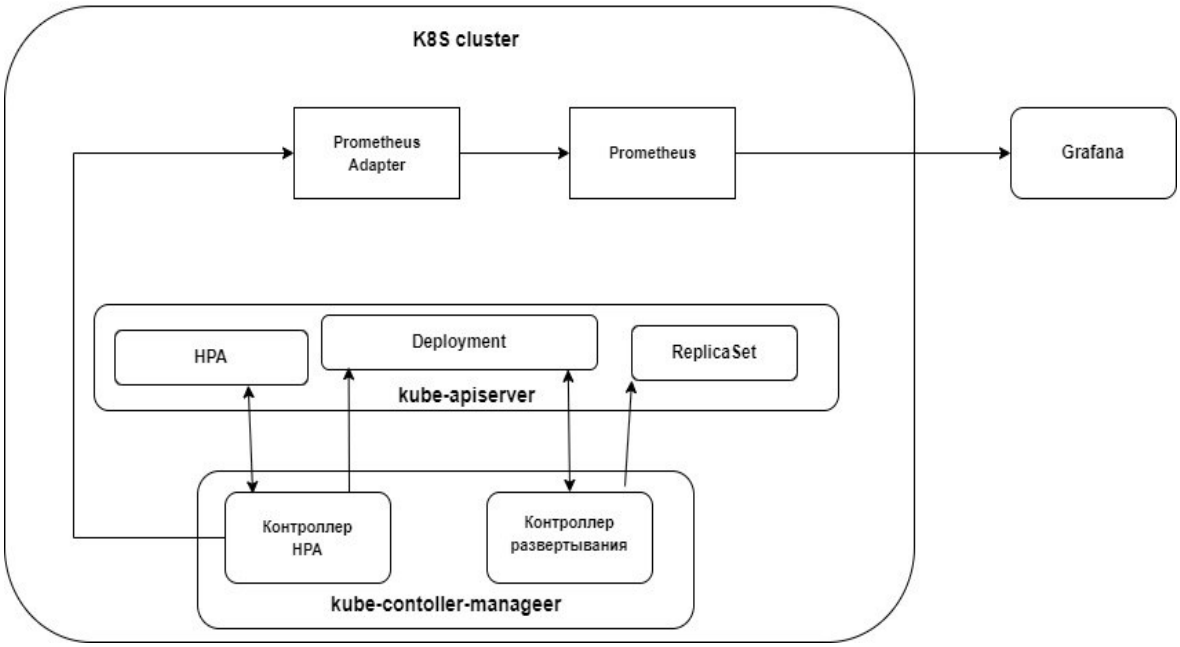

Рис. 1. Архитектурная схема

Сервер собирает метрики потребления ресурсов контейнерами платформы. Он получает эти данные от агента kubelet, запущенного на каждом узле кластера, и делает их доступными для клиентов, которым они нужны. По умолчанию контроллер HPA каждые 15 секунд обращается к API-серверу Kubernetes за информацией об использовании ресурсов. AP1-сервер передает эти запросы серверу метрик, который выдает соответствующие данные. Контроллер HPA следит за ресурсами типа HorizontalPodAutoscaler и на основе заданной в них конфигурации определяет, является ли адекватным число реплик приложения. В листинге продемонстрировано, как это происходит. Приложение чаще всего имеет вид ресурса Deployment, когда контроллер HPA решает, что число реплик нужно отрегулировать, он обновляет соответствующий ресурс Deployment с помощью API-сервера. В ответ контроллер развертываний обновляет ReplicaSet, что приводит к изменению количества Pod'ов. Не все приложения могут масштабироваться горизонтально. Для приложения, которое неспособно разделять нагрузку между несколькими экземплярами, горизонтальное масштабирование не имеет смысла. Это относится к некоторым приложениям, хранящим свое состояние, и

приложениям с выбором лидера. В этих случаях стоит подумать о вертикальном автомасштабировании. Масштабирование будет ограничено размером кластера. Когда приложение расширяется, емкость рабочих узлов кластера может исчерпаться. Чтобы избежать этой проблемы, можно заранее выделять достаточное количество ресурсов (отправляя администраторам платформы оповещения, чтобы те увеличивали емкость вручную) или использовать автомасштабирование кластера. Потребление процессора и памяти может быть неподходящей метрикой для принятия решений о масштабировании. Не следует основывать автомасштабирование приложений на метрике, которая не всегда изменяется пропорционально нагрузке на приложение. Потребление ресурсов процессора – самая распространенная метрика для автомасштабирования. Но, если с повышением нагрузки на определенное приложение количество потребляемого им процессорного времени не меняется существенно, а вместо этого происходит прямо пропорциональное увеличение занятой им памяти, откажитесь от этой метрики.

3. Реализация

Прежде чем развертывать приложение на Kubernetes, необходимо создать Docker-образы для демонстрационного микросервисного приложения и контейнера для нагрузочного тестирования. После создания образов их необходимо загрузить в приватный Docker-реестр для последующего использования. Для успешного запуска демонстрации потребуется установленный Kubernetes кластер, настроенный с помощью инструмента Kubespray. Также необходимо установить Prometheus Operator и адаптер для сбора метрик, а также Redis Server без sentinel. Сбор пользовательских метрик необходимо настроить Prometheus Adapter. Этот шаг включает добавление конфигурации в файл values.yml и обновление адаптера с помощью Helm. Для визуализации метрик масштабирования необходимо создать дашборд в Grafana. Этот шаг позволяет наглядно отслеживать изменения количества реплик приложения в зависимости от нагрузки. Для сбора метрик из приложения необходимо создать объект Service Monitor. Этот шаг определяет, какие метрики собирать с приложения и какие правила использовать для их сбора. После подготовки среды и настройки мониторинга можно развернуть демо-приложение в Kubernetes кластере с использованием YAML-файла. Для проверки работы масштабирования необходимо запустить тест нагрузки на демонстрационное приложение с использованием инструмента нагрузочного тестирования (например, Vegeta). В завершение можно наблюдать за результатами масштабирования через команду или через визуализацию в Grafana, как на рис. 2.

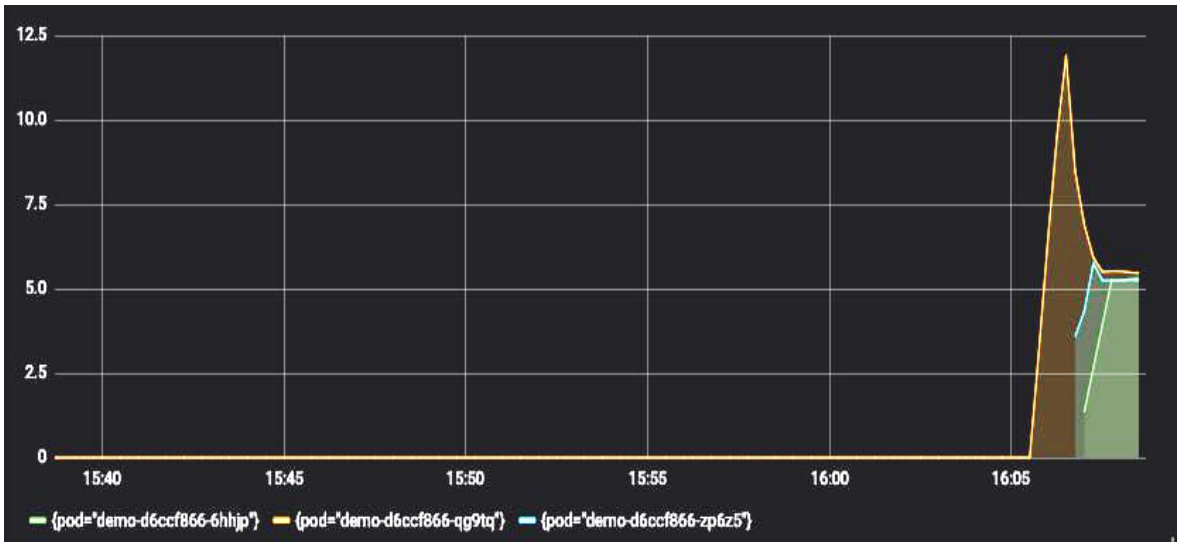

Рис. 2. Пример дашборда с автоскейлингом

Заключение

В работе был реализован процесс оптимизации и отказоустойчивости микросервисов. Изучены разные виды масштабирования, которые позволяют потреблять ресурсы процессора и памяти вне зависимости от требований.

Список литературы

1. Маркелов А. А. Введение в технологию контейнеров и Kubernetes. – М.: ДМК Пресс, 2019. – 194 с.

2. Документация по Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/ru/docs/home/ (дата обращения: 25.05.2024).

3. With Kubernetes, the U.S. Department of Defense is enabling DevSecOps on F-16s and battleships [Electronic resource] // CNCF: website. – URL: https://www.cncf.io/case-study/dod/ (access date: 25.05.2024).

4. CNCF Survey 2019: Deployments are getting larger as cloud native adoption becomes mainstream [Electronic resource] // CNCF: website. – URL: https://www.cncf.io/wp-content/uploads/2020/08/CNCF_Survey_Report.pdf (access date: 25.05.2024).

5. Goodin D. Tesla cloud resources are hacked to run cryptocurrencymining malware [Electronic resource] // Ars Technica: website. – 20 February 2018. – URL: https://arstechnica.com/information-technology/2018/02/tesla-cloud-resources-are-hacked-to-run-cryptocurrency-mining-malware/ (access date: 25.05.2024).