DOI: 10.18137/RNU.V9187.23.02.P.183 УДК 004.05

Д.В. Кокотов

МЕТОДИКА ОРГАНИЗАЦИИ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ СОВРЕМЕННОГО WEB-ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ СЕРВИСНОЙ СЕТИ ISTIO

Аннотация. По мере роста популярности сервисов в Интернете разработчикам все больше внимания необходимо уделять архитектуре приложения и используемым технологиям внутри системы. В данной статье сформулированы ключевые свойства современного web-приложения, необходимые для стабильной работы системы, безопасной передачи информации, а также даны гарантии высокого уровня предоставляемого сервиса. Рекомендована архитектура, позволяющая сформировать фундамент для достижения необходимых качеств системы.

Рассмотрены некоторые способы физического размещения микросервисов на серверных компьютерах и предложено использовать контейнеры Docker как инструмент для эффективного запуска и масштабирования микросервисов. Но в процессе развития системы и повышения ее сложности требуется инструмент для управления множеством контейнеров, в связи с чем рекомендована платформа Kubernetes и описаны ее основные возможности.

Также рассмотрены основные принципы подхода «сервисная сеть» и возможности его эффективной реализации в проекте Istio. Сервисная сеть Istio предоставляет дополнительные возможности для управления трафиком и безопасной передачи информации внутри платформы Kubernetes.

Приведена общая схема архитектуры приложения на платформе Kubernetes с использованием сервисной сети Istio. Описаны ключевые достоинства и недостатки полученной архитектуры, примеры успешного использования выбранных технологий в промышленной среде.

Ключевые слова: микросервисная архитектура, контейнеры Docker, платформа Kubernetes, сервисная сеть Istio.

D.V. Kokotov

METHOD FOR ORGANIZING THE MICROSERVICE ARCHITECTURE OF A MODERN WEB APPLICATION USING THE ISTIO SERVICE NETWORK

Abstract. As the popularity of services on the Internet grows, developers need to pay more and more attention to the architecture of the application and the technologies used within the system. This paper provides the key features of a modern web application, which are necessary for the stable operation of the system, the secure transfer of information, as well as gives guarantees of a high level of the service provided. An architecture is recommended that allows forming the foundation to achieve the necessary qualities of the system.

The paper considers some ways of physical placement of microservices on server computers. As a result, it is proposed to use Docker containers as a tool for efficient deploing and scaling of microservices. But as the system evolves and becomes more complex, you will need a tool to manage multiple containers. To solve this problem, the Kubernetes platform is recommended and its main features are described.

Further, the paper discusses the basic principles of the Service Mesh approach and its effective implementation in the Istio project. The Istio service mesh provides additional options for traffic management and secure communication within the Kubernetes platform.

The last section of the paper is devoted a general diagram of the application architecture on the Kubernetes platform using the Istio service mesh. The key advantages and disadvantages of the resulting architecture are described as well as the examples of successful use of selected technologies in an industrial environment.

Keywords: microservice architecture, Docker container, Kubernetes platform, Istio service mesh.

Кокотов Дмитрий Валерьевич
главный программист, ООО «Цифровые привычки», Санкт-Петербург. Сфера научных интересов: информационные технологии, программирование. ORCID: 0009-0004-1058-7594. Электронный адрес: kokotovdv@gmail.com

Введение

Количество сервисов, предоставляемых компаниями в Интернете, с каждым днем увеличивается; всё больше пользователей выбирают Интернет в качестве способа получения информации, товара или услуги быстрее и дешевле. Рост популярности сервисов приводит к увеличению нагрузки на все модули системы; повышаются требования к безопасности приложений, возникает потребность в инструментах для мониторинга и анализа надежности работы сервисов. Сложность системы существенно возрастает, поэтому очень важно правильно спроектировать архитектуру приложения и выбрать подходящие инструменты для решения возникающих проблем.

Свойства системы и их реализация

Для сохранения и развития качества предоставляемых услуг необходимо обеспечить следующие важные свойства системы.

Масштабируемость (англ. scalability) – возможность независимо увеличивать количество системных ресурсов, предоставляемых отдельным компонентам приложения, а также изменять количество отдельных компонентов системы для увеличения их пропускной способности.

Безопасность данных (англ. data security) – защита цифровой информации от несанкционированного доступа, модификации или уничтожения, в том числе при взаимодействии различных модулей системы между собой.

Высокая доступность (англ. high availability) – возможность непрерывно использовать приложение в течение какого-либо промежутка времени. Например, доступность 99,999 % («пять девяток») гарантирует, что время простоя системы в год будет не более 5,26 минут. (Расчет по формуле: Доступность = (Д – П) / Д · 100 %, где Д – время ожидаемой доступности системы, П – время простоя системы; (360 · 24 · 60 – 5,24) / 360 · 24 · 60 · 100 % = 99,999 % [1]).

Балансировка нагрузки (англ. loadbalancing) – автоматическое распределение нагрузки между отдельными компонентами системы, выполняющими одну задачу.

Отказоустойчивость (англ. fault tolerance) – способность системы продолжать функционировать при отказе одного или нескольких ее компонентов.

Первым шагом в процессе реализации описанных выше свойств системы является микросервисная архитектура приложения. *Микросервисная архитектура* (Рисунок 1) – это подход к проектированию, при котором система строится как набор небольших сервисов (приложений). Каждый сервис работает в собственном процессе и взаимодействует с другими сервисами через сеть. Приложения работают независимо и построены вокруг бизнес-потребностей [2; 3], например, формируют заказы покупателей, отвечают за хранение или доставку товаров и др.

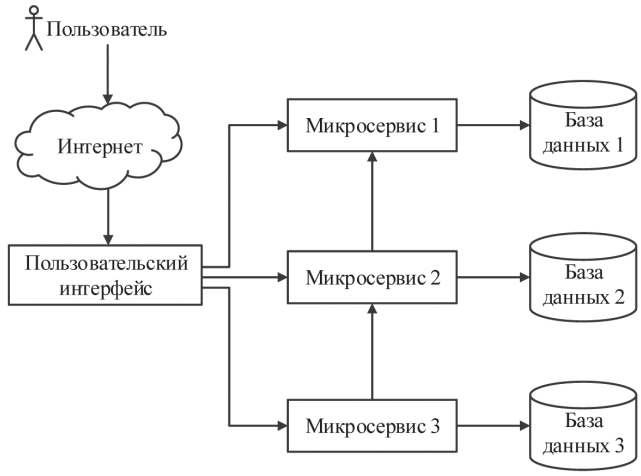

Рисунок 1. Микросервисная архитектура
Источник: составлено автором.

Способы развертывания микросервисов

При традиционном подходе микросервисы запускаются на одном физическом сервере и используются общие системные ресурсы. При этом отсутствует возможность определить границы выделяемых ресурсов для каждого приложения. Для решения этой проблемы можно запустить микросервисы на отдельных физических серверах. Но такой подход не решит проблему полностью, так как приложение не будет использовать все ресурсы каждого сервера.

Помимо распределения системных ресурсов, может потребоваться выполнить *горизонтальное масштабирование*, то есть добавить новые экземпляры (англ. instance) приложений, которые работают параллельно и принимают часть нагрузки на себя. Для этого также потребуются дополнительные физические серверы. В результате финансовые затраты на сопровождение системы существенно повышаются.

Альтернативным способом решения проблем масштабирования является виртуализация. Виртуальная машина – система, эмулирующая реальный компьютер. На виртуальную машину можно установить операционную систему и приложение, которое нужно масштабировать [4]. На одном физическом сервере можно запустить несколько виртуальных машин, то есть повысить эффективность использования ресурсов физического сервера.

Кроме того, виртуализация позволяет изолировать приложения друг от друга, что обеспечивает более *высокую доступность* системы, поскольку приложения можно добавлять или обновлять независимо. Минусом использования виртуальных машин является избыточное количество операционных систем; каждая операционная система, установленная на виртуальную машину, требует дополнительных системных ресурсов.

Технология, которая решает проблему избыточного количества операционных систем, – контейнеризация. Контейнер – процесс операционной системы, работающий в изолированном окружении со своей файловой системой. Контейнеры похожи на виртуальные машины, но используют общую операционную систему и потребляют меньше ресурсов.

Одной из популярных платформ, реализующей контейнеризацию, является *Docker*. Docker позволяет создавать контейнеры и управлять их жизненным циклом [5]. Микросервис запускается внутри контейнера Docker со всеми необходимым зависимостями и библиотеками. Контейнер можно быстро запустить или остановить, создать на другом физическом сервере и др. Использование контейнеров существенно увеличивает скорость разработки, тестирования и запуска микросервисов. Во многих проектах банковской и производственной сфер большинство сервисов упаковываются в контейнеры Docker, что существенно оптимизирует процессы сопровождения системы.

Платформа Kubernetes

По мере развития системы и повышения ее сложности количество контейнеров и связей между ними быстро увеличивается. Появляется потребность в инструменте, позволяющем управлять множеством контейнеров, – *балансировать нагрузку* (англ. load balancing) между контейнерами и осуществлять общий мониторинг работы системы. Для решения этой задачи рекомендуется использовать Kubernetes.

Kubernetes – сложная система с открытым исходным кодом, которая автоматизирует развертывание, масштабирование и менеджмент контейнерных сервисов [6, 7]. Она существенно ускоряет разработку, тестирование и обновление приложений. Kubernetes вводит ряд абстракций для описания инфраструктуры системы (Рисунок 2).

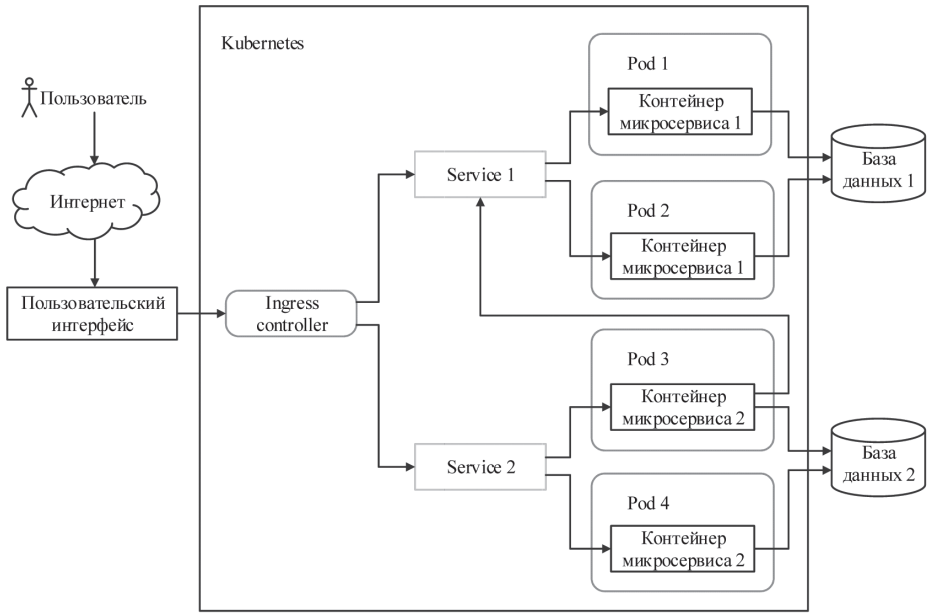

Рисунок 2. Микросервисная архитектура на платформе Kubernetes
Источник: составлено автором.

Ключевые компоненты Kubernetes

• *Pod* – абстрактный объект Kubernetes, который предназначен для запуска одного экземпляра конкретного приложения (микросервиса). Приложение запускается в контейнере Docker внутри Pod'а. Помимо основного контейнера с приложением, Pod может содержать вспомогательные контейнеры, обеспечивающие дополнительную функциональность;

• *Service* – абстракция, которая описывает логический набор Pod'ов и политику доступа к ним, маршрутизирует запросы к конкретному Pod'у, а также выполняет балансировку нагрузки между Pod'ми;

• *Ingress controller* – специальный компонент, который маршрутизирует внешний входящий трафик между Service'ми внутри Kubernetes.

Kubernetes не содержит самостоятельной реализации Ingress controller. На сегодняшний день на рынке существует более двадцати реализаций Ingress controller от различных компаний. Каждый проект имеет свои преимущества и недостатки, однако для получения ключевых возможностей, рассмотренных далее, рекомендуется использовать паттерн «сервисная сеть» (англ. service mesh).

Сервисная сеть Istio

Сервисная сеть – это подход, при котором к каждому микросервису внутри системы добавляется прокси-сервис, который перехватывает весь трафик приложения и предоставляет дополнительные возможности для управления системой [8].

Популярной реализацией паттерна «сервисная сеть», являет проект *Istio*. Istio расширяет Kubernetes, создавая дополнительный уровень абстракции над сетью, перехватывая весь входящий и исходящий трафик внутри кластера (Рисунок 3).

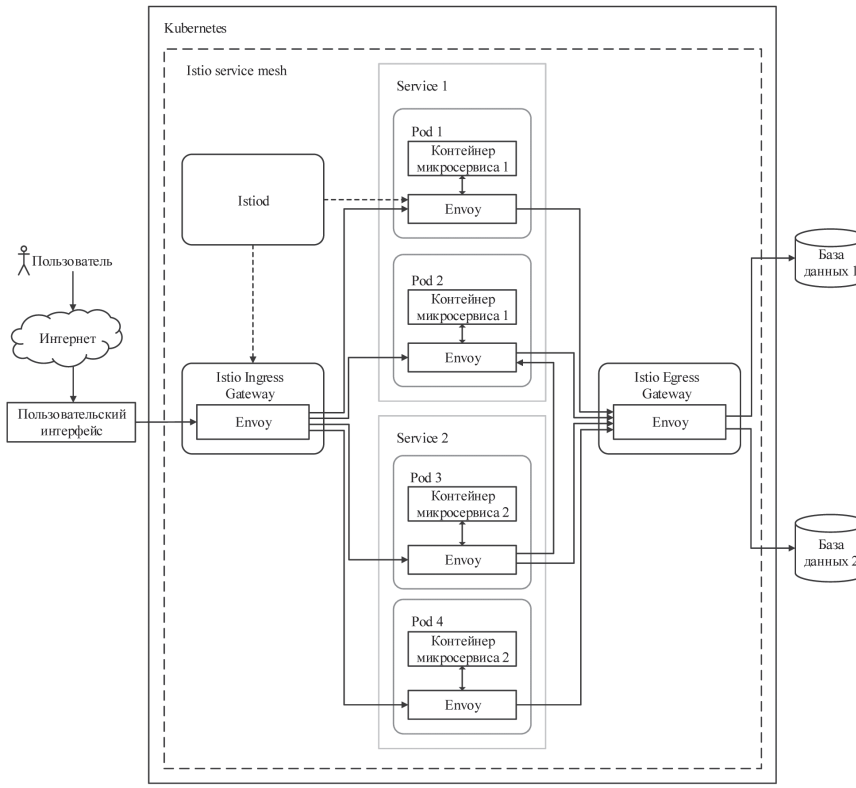

Рисунок 3. Микросервисная архитектура на платформе Kubernetes с использованием сервисной сети Istio
Источник: составлено автором.

Istio использует расширенную версию мощного сервисного прокси Envoy. Envoy – высокопроизводительный прокси с открытым исходным кодом, разработанный на C++ для контроля и передачи входящего и исходящего трафика сервисов внутри системы [9]. Istio разворачивает Envoy как sidecar-контейнер для каждого сервиса – дополнительный контейнер внутри Pod'а, логически дополняющий сервисы многочисленными возможностями.

Для обеспечения маршрутизации входящего и исходящего трафика Istio содержит компоненты Ingress Gateway и Egress Gateway – специальные Pod'ы, которые также запускают внутри себя контейнер с прокси-Envoy и обеспечивают контроль перемещения и безопасность трафика в кластере Kubernetes [10].

Помимо компонентов, отвечающих за перемещение данных (dataplane), Istio содержит компоненты для общего конфигурирования системы – Istiod (controlplane). Istiod преобразует высокоуровневые правила управления трафиком в специализированные конфигурации для Envoy и распространяет их для sidecar-контейнеров динамически во время работы системы. Istiod также отвечает за аутентификацию, авторизацию и контроль сертификатов [11].

Основные возможности сервисной сети Istio и решаемые ею задачи:

• управление трафиком (англ. traffic control) – детальное управление трафиком с помощью расширенных правил маршрутизации по протоколам HTTP, gRPC, WebSocketи TCP;

• динамическое обнаружение сервисов (англ. dynamic service discovery) – процесс формирования списка сервисов в пределах Kubernetes. Сервисы в Kubernetes получают IP-адреса динамически. Envoy позволяет связывать сервисы между собой без дополнительного программирования маршрутизации на уровне приложения;

• безопасность данных (англ. data security) – возможность использовать mTLS (mutual TLS) для запросов между сервисами. MTLS – криптографический протокол взаимной (двусторонней) аутентификации клиента и сервиса, который является преемником протокола SSL;

• тайм-ауты (англ. timeouts) – время, в течение которого Envoy должен ожидать ответа от сервиса, гарантируя, что сервис не зависает в ожидании на неопределенный срок, а вызов завершится успехом или неудачей в течение предсказуемого периода времени;

• повторные попытки (англ. retries) – автоматические повторные попытки подключения к сервису, если предыдущий вызов не удался;

• автоматические выключатели (англ. circuit breakers) – механизм, обнаруживающий сбои в работе приложения и не позволяющий выполнить действие, которое приводит к ошибкам, до тех пор, пока проблема не будет устранена;

• внесение неисправностей (англ. faultin jection) – метод тестирования, в рамках которого искусственно генерируются ошибки при вызове сервисов и производится проверка устойчивой работы системы;

• канареечное развертывание (англ. canary deployment) – методика обновления сервисов, при которой небольшая часть пользователей перенаправляется на новую версию сервиса, что позволяет производить тестирование в рабочей среде и при обнаружении проблем быстро восстанавливать работоспособность приложения, возвращая пользователей на стабильную версию сервиса;

• трассировка и мониторинг (англ. tracing and monitoring) – возможность отслеживать и анализировать всю цепочку вызовов сервисов, используя данную информацию для решения возможных проблем в работе системы [12].

Заключение

В результате сервисная сеть Istio существенно повышает *безопасность* и *отказоустойчивость* работы системы, а также предоставляет эффективные инструменты для *мониторинга* входящего и исходящего трафика. Istio отлично решает задачи маршрутизации запросов внутри кластера Kubernetes, однако если нужно активно управлять сервисами за пределами Kubernetes, то необходимо рассмотреть другие технологии. Также к недостаткам Istio можно отнести необходимость в большом количестве дополнительных контейнеров Envoy, что потребует увеличения количества системных ресурсов.

Таким образом, если система имеет высокие требования к безопасности и отказоустойчивости, если есть высокая нагрузка на сервисы и нет острой нехватки системных ресурсов, то рекомендуется использовать платформу Kubernetes в комбинации с сервисной сетью Istio. Данная архитектура успешно используется в сложных системах банков для расчета финансовых показателей и получения эффективных результатов. Приложения работают стабильно, и клиенты довольны качеством предоставляемых услуг.

Литература

1. *Atchison L.* Architecting for Sale: High Availability for Your Growing Applications. 1st edition. O'Reilly Media, Inc., 2016, 230 p. ISBN 1491943394.
2. *Newman S.* Building Microservices. 2nd edition. O'Reilly Media, Inc., 2021, 615 p. ISBN 9781492034025.
3. *Lewis J., Fowler M.* Microservices // martinFowler.com. 2014. 25 March. URL: https://martinfowler.com/articles/microservices.html (дата обращения: 07.01.2023).
4. OS-level_virtualization // Wikipedia. The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/OS-level_virtualization (дата обращения 08.01.2023).
5. Docker documentation // Docker docs. Guides. URL: https://docs.docker.com/get-started/overview/ (дата обращения: 08.01.2023).
6. *Poulton N.* The Kubernetes Book. Independently published, 2022. 309 p. ISBN 979-8402153776.
7. Kubernetes documentation // Kubernetes. URL: https://kubernetes.io/docs/home/ (дата обращения: 10.01.2023).
8. *Richardson R., Pearlman K.* Service Mess to Service Mesh // Cloud Native Computing Foundation. Blog. 2020. 14 February. URL: https://www.cncf.io/blog/2020/02/14/service-mess-to-service-mesh/ (дата обращения: 12.01.2023).
9. Envoy documentation // Envvoy. Docs. URL: https://www.envoyproxy.io/docs (дата обращения: 11.01.2023).
10. *Chase N.* Your App Deserves More than Kubernetes Ingress: Kubernetes Ingress vs. Istio Gateway // Mirantis. 2021. 12 August. URL: https://www.mirantis.com/blog/your-app-deserves-more-than-kubernetes-ingress-kubernetes-ingress-vs-istio-gateway-webinar/ (дата обращения: 12.01.2023).
11. Istio documentation // Istio. Documentation. URL: https://istio.io/latest/docs/ (дата обращения: 12.01.2023).
12. *Posta C., Maloku R.* Istio in action. Shelter Island : Manning Publications, 2022. 480 p. ISBN 1617295825.

References

1. Atchison L. (2016) *Architecting for Sale: High Availability for Your Growing Applications*. 1st edition. O'Reilly Media, Inc. 230 p. ISBN 1491943394.
2. Newman S. (2021) *Building Microservices.* 2nd edition. O'Reilly Media, Inc. 615 p. ISBN 9781492034025.
3. Lewis J., Fowler M. (2014) Microservices. *martinFowler.com*. 25 March. URL: https://martinfowler.com/articles/microservices.html (accessed 07.01.2023).
4. OS-level_virtualization. *Wikipedia. The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/OS-level_virtualization (accessed 08.01.2023).
5. Docker documentation. *Docker docs. Guides.* URL: https://docs.docker.com/get-started/overview/ (accessed 08.01.2023).
6. Poulton N. (2022) *The Kubernetes Book*. Independently published. 309 p. ISBN 979-8402153776.
7. Kubernetes documentation. *Kubernetes.* URL: https://kubernetes.io/docs/home/ (accessed 10.01.2023).
8. Richardson R., Pearlman K. (2020) Service Mess to Service Mesh. *Cloud Native Computing Foundation. Blog*. 14 February. URL: https://www.cncf.io/blog/2020/02/14/service-mess-to-service-mesh/ (accessed 12.01.2023).
9. Envoy documentation. *Envvoy. Docs.* URL: https://www.envoyproxy.io/docs (accessed 11.01.2023).
10. Chase N. (2021) Your App Deserves More than Kubernetes Ingress: Kubernetes Ingress vs. Istio Gateway. *Mirantis.* 12 August. URL: https://www.mirantis.com/blog/your-app-deserves-more-than-kubernetes-ingress-kubernetes-ingress-vs-istio-gateway-webinar/ (accessed 12.01.2023).
11. Istio documentation. *Istio. Documentation*. URL: https://istio.io/latest/docs/ (accessed 12.01.2023).
12. Posta C., Maloku R. (2022) *Istio in action*. Shelter Island : Manning Publications. 480 p. ISBN 1617295825.