DOI 10.37882/2223–2966.2023.04–2.36

МЕТОДОЛОГИЯ ВНЕДРЕНИЯ БЕССЕРВЕРНЫХ ТЕХНОЛОГИЙ В СИСТЕМУ С МИКРОСЕРВИСНОЙ АРХИТЕКТУРОЙ

METHODOLOGY FOR IMPLEMENTING SERVERLESS TECHNOLOGY IN A SYSTEM WITH A MICROSERVICE ARCHITECTURE

S. Shalgueva
T. Leontieva

Summary. Serverless technology is slowly gaining popularity, attracting auto-scaling capabilities, flexible configurations, and most importantly the pay-for-performance model. This paper explores the feasibility of migrating an existing microservices system to a serverless approach using the Knative framework. A methodology is developed to convert conventional microservices running in Kubernetes to serverless mode with support for modes of installing a new network layer or configuring an existing one. A system performance comparison between regular microservices and those converted to serverless mode is given. The results of performance measurements of the new system, including the costs of "cold start" depending on the workload of services and the programming languages in which these services are developed, are presented.

Keywords: cloud computing, microservices, serverless computing, self-hosted frameworks, knative.

Шалгуева София Леонидовна
Санкт-Петербургский политехнический университет Петра Великого
shalgueva.sl@edu.spbstu.ru
Леонтьева Татьяна Владимировна
Кандидат технических наук, доцент, Санкт-Петербургский политехнический университет Петра Великого
leontyeva@ics2.ecd.spbstu.ru

Аннотация. Бессерверные технологии постепенно набирают популярность, привлекая возможностями автомасштабирования, гибкостью конфигураций, и главное моделью оплаты «плати только за время работы». В данной работе рассмотрена возможность перевода существующей системы микросервисов на бессерверный подход с использованием фреймворка Knative. Разработана методология перевода обычных микросервисов, работающих в Kubernetes, в бессерверный режим с поддержкой режимов установки нового сетевого слоя или конфигурации уже имеющегося. Приводится сравнение характеристик системы обычных микросервисов и переведённых в бессерверный. Представлены результаты замеров производительности новой системы, в том числе и затраты на «холодный старт в зависимости от загруженности сервисов и языков программирования, на которых эти сервисы разработаны.

Ключевые слова: облачные вычисления, микросервисы, бессерверные вычисления, self-hosted фреймворки, knative.

Введение

В современном мире компьютерных наук трудно назвать сферу, в которой не использовались бы облачные технологии. На данный момент облачные вычисления даже признают одной из лучших вычислительных парадигм [1]. Такая популярность обусловлена удобностью технологии и для клиентов, и для разработчиков.

Однако с появлением облачных вычислений перед разработчиками возникли дополнительные задачи. При разработке продукта необходимо дополнительно думать о балансировке нагрузки, возможности масштабирования, безопасности, мониторинга. Решение данных проблем привело к возникновению парадигмы бессерверных вычислений (serverless computing) [2]. При использовании бессерверных технологий заботы по управлению нагрузкой, выделением ресурсов, масштабированием ложатся на облачного провайдера, а пользователь может сосредоточиться на разработке. Бессерверные вычисления также привлекают моделью оплаты "плати только за время использования". Однако разработка приложений в соответствии с новой парадигмой может оказаться непростой из-за фундаментальных различий в архитектурах. Например, бессерверные экземпляры являются эфемерными, поэтому все данные приложения должны храниться во внешнем хранилище. Кроме того, бессерверные приложения трудно поддаются отладке и приносят новые сценарии отказов.

На данный момент запуск сервисов в бессерверном режиме возможен в рамках подходов: Function-as-a-Service (FaaS), Container-as-a-Service (CaaS) и с использованием Self-hosted фреймворков.

Function-as-a-Service (FaaS) [3] и Container-as-a-Service (CaaS) [4] — решения, предоставляемые облачными провайдерами, которые позволяют пользователям управлять функциональными возможностями приложений без необходимости создания и поддерж-

ки собственной инфраструктуры. Клиентам достаточно лишь загрузить исполняемый код в поддерживаемом сервисом формате. Оплата будет взиматься за ресурсы (vCPU и RAM), которые будет потреблять исполняемый код во время работы.

Self-hosted бессерверные фреймворки используют другой подход. Это больше не отдельный сервис, предоставляемый облачным провайдером, это дополнительный функционал к существующим PaaS и IaaS решениям [5]. В отличие от технологий FaaS и CaaS, подобные фреймворки могут быть развернуты непосредственно на виртуальной машине, в том числе и рядом с существующими системами. Это позволяет избежать привязки к определённому облачному провайдеру и предоставляет больше гибкости в конфигурациях по сравнению с публичным облаком. Однако создание собственных эффективных функций требует большого опыта и глубокого понимания фреймворков платформы и характеристик, влияющих на производительность [6]. Также важной особенностью являются возможные затраты на размещение фреймворка в кластере и условия холодного старта serverless приложений.

Данная работа посвящена исследованию возможности применения бессерверных технологий в существующей системе с микросервисной архитектурой и разработке методологии внедрения одного из фреймворков с целью сокращения потребления ресурсов. В работе приводятся результаты тестирования получившейся системы и анализ итоговых результатов. В качестве используемого фреймворка выбран Knative.

Обзор литературы

Бессерверные вычисления набирают популярность, так как позволяют пользователям сосредоточиться на написании бизнес-логики, а не сосредотачиваться на управлении серверами. Балдини и др. [7] подчеркнули это в обзоре сервисов от различных облачных провайдеров, предоставляющих возможность работы с бессерверными технологиями. В своем исследовании авторы описали примеры использования таких приложений и обозначили возможные сложности работы с бессерверными вычислениями. Подобные работы были проведены Линном и др. [8], Ли и др [9], Шилакером и др. [10] и Ллойдом и др. [11]. Однако данные исследования в основном сосредоточены на сравнении различных FaaS и CaaS сервисов между собой.

Моханти и др. [12] в своей работе проводят комплексное сравнение характеристик популярных фреймворков для бессерверных вычислений. Авторы проводят оценку производительности для таких фреймворков как Fission, Kubeless и OpenFaaS. Критикос и др. [13] тоже провели сравнение характеристик бессерверных фреймворков, но выбрали других представителей: IronFunctions, Sparta, Fn. Аналогичные исследования представили Паладе и др [14] и Ли и др. [15]. В этих исследованиях к обзору уже известных фреймворков добавляется новый фреймворк от Google — Knative. Авторы сходятся во мнении, что Knative является перспективным фреймворком с хорошими характеристиками производительности и возможностью автомасштабирования в ноль экземпляров.

Knative — это корпоративное решение с открытым исходным кодом для создания бессерверных и event-driven приложений [16]. Knative был разработан компанией Google в 2018 году, а теперь поддерживается как open-source проект. Несмотря на то, что Knative достаточно молодой фреймворк, его активно применяют в разных сферах: машинном обучении [17, 18], интернете вещей [19, 20], биоинформатике [21], решении задач мониторинга [22] и других.

Ванг и др. в [19] также рассматривают возможность применения Knative в системе микросервисов, проводят анализ как количество и тип микросервисов влияют на производительность. Их результаты показывают, что бессерверный подход в итоге действительно помогает оптимизировать расход ресурсов по отношению к традиционному подходу облачных вычислений. Каллас и др. [23] в своём исследовании представили новый фреймворк для перевода микросервисных приложений, написанных на Python, в бессерверный режим. Разработанный фреймворк управляет жизненным циклом новых сервисов и работает на основе технологии Knative.

В основном исследования, связанные с Knative, разбирают вопросы эффективности использования бессерверного подхода в рамках одного приложения или набора тестов. Применимость Knative в системах связанных микросервисов ещё мало изучена.

Обзор Knative

В качестве self-hosted фреймворка для работы с бессерверными приложениями был выбран Knative [24], потому что он отлично интегрирован с Kubernetes и поддерживает “scale-to-zero”, то есть возможность автомасштабирования в 0 реплик сервиса, а значит сервис полностью перестаёт потреблять ресурсы. Также фреймворк умеет ограничивать ресурсы, потребляемые функциями. Knative бесплатен для использования, распространяется по открытой лицензии Apache 2.0 и имеет активное сообщество и постоянные релизы.

Knative состоит из двух независимых компонент, которые могут использоваться как вместе, так и раз-

Таблица 1. Описание типов микросервисов.

Тип сервиса	Потребление CPU в пассивном режиме, cores	Потребление CPU в активном режиме, cores	Потребление RAM, MiB	Критичность затрат на холодный старт	Время работы сервиса	Количество сервисов данного типа
A	0.1	0.5–1.5	300–500	Средний	10%	7
B	0.1	0.2–0.5	36–100	Высокий	10%	2
C	0.1	1	900	Высокий	90%	2
D	0.05	0.2	64	Низкий	1%	1

дельно. Компоненты Serving предназначены для управления бессерверными приложениями, в том числе процессами деплоя и автомасштабирования. Компоненты Eventing предназначены для разработки и управления системами приложений, построенных в событийно-ориентированной архитектуре. Все компоненты Knative определяются через набор пользовательских определений ресурсов (CRD), каждое из которых отражает подмножество функциональных требований для приложения Knative.

Основными ресурсами Knative Serving являются сервисы — они управляют жизненным циклом приложения, контролируют создание других объектов и направляют трафик.

Основными ресурсами Knative Eventing являются источники событий, брокеры и триггеры. Источники событий создают сообщения на основе заданных фильтров. Брокер предоставляет канал для передачи событий между источником событий и потребителем. Триггер предоставляет механизм подписки на события из определенных источников событий.

Потребителем событий может выступать любой ресурс, который удовлетворяет одному из интерфейсов Kubernetes Addressable или Callable.

Описание существующей системы

Исходная система состоит из 12 микросервисов. Для обеспечения связи между микросервисами используется собственный service-mesh, базирующийся на Envoy. Сервисы взаимодействуют друг с другом посредством REST запросов, а также через websockets и gRPC. Все сервисы поддерживают режим масштабирования в несколько экземпляров, работающих параллельно. В системе присутствуют приложения, написанные на языках Go, Java Spring и Java Quarkus. Деплой сервисов системы в Kubernetes происходит с помощью инструмента Helm. Конфигурации для деплоя описаны в YAML файлах, базовыми сущностями являются Deployment и Service [25].

В системе можно выделить базовые типы микросервисов, представленные в таблице 1.

При работе с бессерверными технологиями необходимо учитывать затраты на “холодный старт” приложения. Холодный старт — время, затрачиваемое системой на создание нового контейнера при первом обращении к функционалу. Проблема холодного старта не всегда является критичной для бессерверного направления, так как для некоторых приложений задержка запуска на несколько секунд не представляет проблем.

Методология внедрения

Алгоритм внедрения бессерверных технологий в существующую систему микросервисов выглядит следующим образом:

1. Установка компонент Knative в кластер при необходимости
2. Адаптация существующих микросервисов к бессерверному режиму
3. Деплой обновлённых микросервисов в кластер

Установка компонент Knative в кластер при необходимости

Компоненты Knative необходимо установить заранее или при первом запуске. Компоненты Serving и Eventing не зависят друг от друга и могут быть установлены в разное время. Все объекты Knative имеют кластерную область видимости, то есть после установки доступ к ним возможен из любых неймспейсов кластера.

- компонента Serving нужна для включения возможности автомасштабирования экземпляров сервиса

Таблица 2. Описание различий между Deployment и Knative Service.

Секция изменений	Deployment	Knative Service	Комментарий
Заголовок	kind: Deployment apiVersion: apps/v1	kind: Service apiVersion: serving.knative.dev/v1	Вместо Deployment используется новый объект Service
Секция spec	replicas: X	-	Секция с количеством реплик сервиса отсутствует, потому что цель использования Knative — автомасштабирование в зависимости от нагрузки
Секция nodeSelector	nodeSelector:…	-	Отсутствует возможность запуска экземпляра на определённом узле, Knative сам выбирает в зависимости от доступных ресурсов
Секция affinity	affinity.podAntiAffinity: …	-	
Пробы	- livenessProbe - readinessProbe - startupProbe	- livenessProbe - readinessProbe	Отсутствует возможность задавать startupProbe, при старте Knative ориентируется на встроенные метрики для определения состояния сервиса

- компонента Eventing позволяет объединять бессерверные приложения в последовательности вызовов

Адаптация существующих микросервисов к бессерверному режиму

Knative Services являются ресурсом Kubernetes и принадлежат к группе serving.knative.dev. Спецификация этого объекта практически полностью совпадает со спецификацией ресурса Deployment, за исключением некоторых полей. Различия между спецификациями отражены в таблице 2.

Если внести все указанные изменения в исходный Deployment.yaml, получим готовую к деплою конфигурацию исходного сервиса в бессерверном формате. Также ресурс Knative Service управляет всеми своими ревизиями и трафиком через них, а значит надобность в ресурсе Service отпадает и его можно удалить из проекта.

При переводе сервиса в бессерверный режим необходимо учитывать следующие особенности:

- Автомасштабирование запускается только через HTTP запросы. Взаимодействие по протоколам websockets и gRPC будет осуществляться только пока сервис работает после http вызова.
- Контейнер не имеет состояния, поэтому все данные должны быть воспроизводимыми или сохраняться во внешнем хранилище.
- Приложение может использовать только тома Secret и ConfigMap.

Деплой обновлённых микросервисов в кластер

Развертывание обновлённой системы в кластере Kubernetes возможно любым способом, отличий от обычной системы микросервисов нет. Для корректной работы Knative Serving необходимо дополнительно сконфигурировать сетевой слой. Сетевой слой отвечает за маршрутизацию запросов.

Каждый сервис необходимо зарегистрировать в сетевом слое, чтобы envoy обновил свои таблицы маршрутизации. При использовании сетевого слоя, предоставляемого командой Knative это будет сделано автоматически. При использовании собственного service-mesh нужно отправить запрос регистрации необходимых маршрутов трафика на API control-plane. Чтобы сетевой слой мог работать с Knative, он должен базироваться на envoy-proxy сервисе.

Результаты перевода

Исследование проводилось в кластере Kubernetes. Ресурсы кластера 16 CPU и 32 ГБ RAM. Версия Kubernetes v1.23.10. Заранее была проведена установка ресурсов Serving и Eventing.

В таблице 3 представлены результаты замеров потребляемых ресурсов до перевода всех микросерви-

Таблица 3. Сравнение потребления ресурсов кластером.

	Потребление RAM, ГБ	Потребление CPU в режиме без нагрузки	Потребление CPU в режиме нагрузки
До перевода на Knative	4.1	1.1	8
После перевода на Knative	4.2	1.1	11

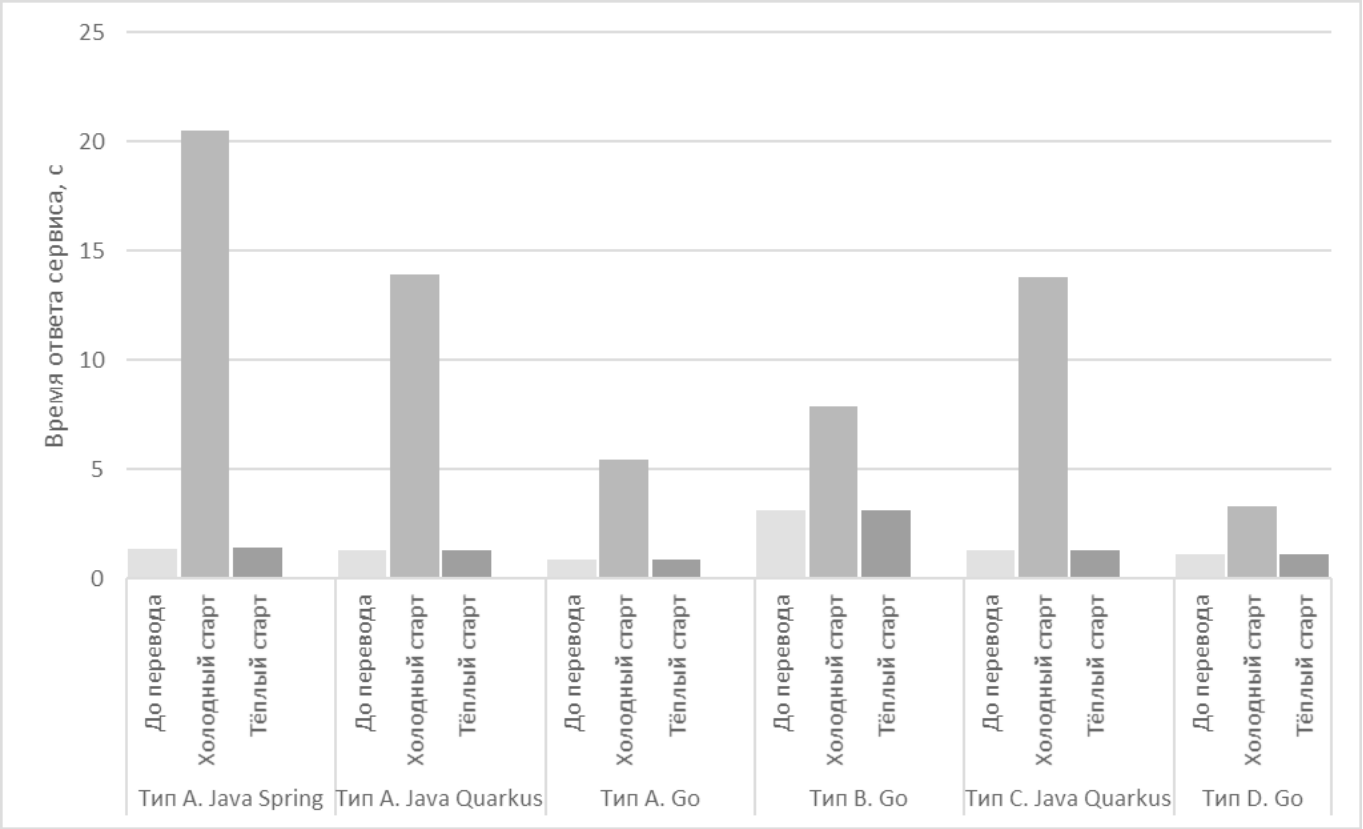

Рис. 1. Время ответа сервиса в зависимости от режима тестирования

сов в бессерверный режим и после. Замеры CPU проводились в двух режимах, при отсутствии нагрузки и при нагрузке 10 запросов в секунду к каждому сервису с открытым API.

При переходе на использование Knative необходимо учитывать, что сам фреймворк использует часть ресурсов кластера. Документация предполагает, что для устойчивой работы Knative необходимо 2 CPU и 4 ГБ RAM. Во время проведения исследования фреймворк занял 1 CPU и 2 ГБ RAM.

За счёт этой надбавки случился проигрыш по RAM после перехода на Knative. К тому же у каждого Knative Service при старте запускаются два контейнера: один с экземпляром приложения, а второй с прокси сервисом, необходимый для корректной работы автомасштабирования. Этот прокси контейнер занимает порядка 50 МБ и вносит свой вклад в общую потребляемую память.

Достигнуть выигрыша по CPU тоже не удалось. В режиме простоя потребление CPU примерно одинаковое, но в обычной системе это из-за постоянно работающих инстансов приложения, а в бессерверном режиме это из-за затрат на ресурсы самого Knative. В этом режиме активных экземпляров приложений нет.

В режиме повышенной нагрузки были отмечены следующие моменты:

- для некоторых сервисов автоматически были созданы дополнительные экземпляры, часть запросов обрабатывалась параллельно. Knative предоставляет возможность регулировать по-

Таблица 4. Замеры времени выполнения запросов при нагрузочном тестировании.

Тип сервиса	Допустимое время ответа, с.	Язык, на котором написан сервис	Минимальное время ответа в бессерверном режиме, с.	Максимальное время ответа в бессерверном режиме, с.
Тип A	10	Java Spring	1.3	21.4
		Java Quarkus	1.3	14.1
		Go	0.8	6.0
Тип B	3	Go	2.4	8.9
Тип C	1	Java Quarkus	1.3	14.4
Тип D	10	Go	0.8	3.3

явление дополнительных экземпляров, задавать ограничение на количество реплик или выбирать по какому принципу будет проводиться автобалансировка.

- для сервисов типа C не были заданы нужные ограничения по времени, поэтому Knative принудительно завершал выполнение задачи и уничтожал экземпляр. После дополнительного конфигурирования параметров времени выполнения проблема была решена.

На рис. 1 представлены графики среднего времени ответа микросервисов до перевода и после. Для каждого типа указан язык программирования, на котором разработаны сервисы. Для приложений в бессерверном режиме проводилось несколько замеров времени: в режиме холодного старта и в режиме, когда экземпляр приложения уже был запущен.

Как видно из представленных графиков, холодный старт ощутимо влияет на время обращения к сервису. Особенно критично это для сервисов, разработанных с использованием Java Spring, так как им для старта требуется несколько минут. В ходе эксперимента мы сталкивались с тем, что сервис не успевал полностью подняться и обработать запрос, а Knative уже был готов завершить работу экземпляра по таймауту. Однако в случае тёплого старта, бессерверные сервисы обрабатывают запросы в течение того же времени, что и в обычном режиме.

Тестирование

Переведённую на использование Knative систему необходимо было протестировать. Было проведено несколько видов тестирования: интеграционное для проверки отсутствия регрессии функционала и нагрузочное для проверки эффективности работы системы под нагрузкой.

Исследование проводилось в кластере Kubernetes. Ресурсы кластера 16 CPU и 32 ГБ RAM. Версия Kubernetes v1.23.10. Перед каждым видом тестирования из кластера удалялись все ресурсы и заново устанавливалась нужная версия системы. Запросы к кластеру отправлялись с рабочей машины с процессором CPU AMD Ryzen 7 PRO 4750U with Radeon Graphics, 8 CPU, 32 ГБ RAM.

В качестве интеграционных тестов использовался существующий набор тестов, которые используются для проверки работоспособности исходной системы. Все обращения к сервисам проходят через service-mesh, поэтому не понадобилось вносить изменений в существующие методы.

Был выполнен полный набор тестов. Прохождение всех тестов заняло 1 час 54 минуты, что на 10 минут дольше среднего времени выполнения обычных тестов. Небольшое увеличение времени выполнения связано с тем, что тесты выполняются для каждого сервиса по очереди, а значит возникала ситуация “холодных стартов”. Все тесты были пройдены без ошибок.

Во время тестирования производительности проводилось два исследования: способность системы работать под нагрузкой длительное время без деградации и способность обрабатывать входящие запросы за заданное время. Во время тестирования не было выявлено отказов каких-либо сервисов, но во временные рамки уложились не все запросы (таблица 4) .

Из таблицы 4 видно, что холодный старт достаточно сильно ограничивает возможность применения бес-

серверных технологий. Но для сервисов типа A и D, написанных на Go и Quarkus возможные задержки укладываются в допустимые лимиты.

Выводы

Нами была разработана и проверена методология перехода с обычной микросервисной модели на использование бессерверных технологий. Новая система была протестирована на сохранение функциональности, на способность выдерживать нагрузки и оставаться работоспособной — все тесты были пройдены с положительным результатом.

Однако эксперименты показали, что использование бессерверных технологий не всегда выгодно. Так при высокой нагрузке на сервисы все преимущества бессерверных систем нивелируются и системы могут потреблять даже больше ресурсов. Для некоторых сервисов критично время ответа, что не может быть удовлетворено средствами Knative из коробки.

В дальнейших планах определить микросервисы с какими характеристиками оптимально переводить в бессерверный режим, разработать инструменты автоматического перевода с микросервисного подхода в бессерверный.

ЛИТЕРАТУРА

1. Alam T. Cloud Computing and its role in the Information Technology // IAIC Transactions on Sustainable Digital Innovation (ITSDI). 2020. Т. 1. № 2. С. 108–115.
2. Jonas E. et al. Cloud programming simplified: A berkeley view on serverless computing // arXiv preprint arXiv:1902.03383. 2019.
3. Shafiei H., Khonsari A., Mousavi P. Serverless computing: a survey of opportunities, challenges, and applications // ACM Computing Surveys. 2022. Т. 54. № 11s. С. 1–32.
4. Hussein M.K., Mousa M.H., Alqarni M.A. A placement architecture for a container as a service (CaaS) in a cloud environment // Journal of Cloud Computing. 2019. Т. 8. № 1. С. 1–15.
5. Decker J., Kasprzak P., Kunkel J.M. Performance Evaluation of Open-Source Serverless Platforms for Kubernetes // Algorithms. 2022. Т. 15. № 7. С. 234.
6. Li J. et al. Analyzing Open-Source Serverless Platforms: Characteristics and Performance // arXiv preprint arXiv:2106.03601. 2021.
7. Baldini I. et al. Serverless computing: Current trends and open problems // Research advances in cloud computing. Springer, Singapore, 2017. С. 1–20.
8. Lynn T. et al. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms // 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2017. С. 162–169.
9. Lee H., Satyam K., Fox G. Evaluation of production serverless computing environments // 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE, 2018. С. 442–450.
10. Shillaker S. A provider-friendly serverless framework for latency-critical applications // 12th Eurosys Doctoral Workshop. 2018. С. 71.
11. Lloyd W. et al. Serverless computing: An investigation of factors influencing microservice performance // 2018 IEEE international conference on cloud engineering (IC2E). IEEE, 2018. С. 159–169.
12. Mohanty S.K. et al. An Evaluation of Open Source Serverless Computing Frameworks // CloudCom. 2018. Т. 2018. С. 115–120.
13. Kritikos K., Skrzypek P. A review of serverless frameworks // 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). IEEE, 2018. С. 161–168.
14. Palade A., Kazmi A., Clarke S. An evaluation of open source serverless computing frameworks support at the edge // 2019 IEEE World Congress on Services (SERVICES). IEEE, 2019. Т. 2642. С. 206–211.
15. Li J. et al. Analyzing open-source serverless platforms: Characteristics and performance // arXiv preprint arXiv:2106.03601. 2021.
16. Chester J. Knative in Action. NY: Simon and Schuster, 2021. 272 С.
17. Cox C. et al. Serverless inferencing on Kubernetes // arXiv preprint arXiv:2007.07366. 2020.
18. Bac T.P., Tran M.N., Kim Y.H. Serverless computing approach for deploying machine learning applications in edge layer // 2022 International Conference on Information Networking (ICOIN). IEEE, 2022. С. 396–401.
19. Wang I., Liri E., Ramakrishnan K.K. Supporting iot applications with serverless edge clouds // 2020 IEEE 9th International Conference on Cloud Networking (CloudNet). IEEE, 2020. С. 1–4.
20. Wang I.C. et al. Towards a Proactive Lightweight Serverless Edge Cloud for Internet-of-Things Applications // 2021 IEEE International Conference on Networking, Architecture and Storage (NAS). IEEE, 2021. С. 1–4.
21. Grzesik P. et al. Serverless computing in omics data analysis and integration // Briefings in bioinformatics. 2022. Т. 23. № 1. С. 349.
22. Klaise J. et al. Monitoring and explainability of models in production // arXiv preprint arXiv:2007.06299. 2020.
23. Kallas K. et al. Executing Microservice Applications on Serverless, Correctly // Proceedings of the ACM on Programming Languages. 2023. Т. 7. № . POPL. С. 367–395.
24. Knative — Documentation: [сайт] URL: https://knative.dev (дата обращения: 21.01.2023).
25. Kubernetes — Documentation: [сайт] URL: https://kubernetes.io/docs/home/ (дата обращения: 21.01.2023) .

© Шалгуева София Леонидовна (shalgueva.sl@edu.spbstu.ru), Леонтьева Татьяна Владимировна (leontyeva@ics2.ecd.spbstu.ru).
Журнал «Современная наука: актуальные проблемы теории и практики»