Программные системы и вычислительные методы
Правильная ссылка на статью:
Рогов Д.В., Алпатов А.Н. Интеллектуальная инфраструктура автоматизированного управления и интероперабельности микросервисов в облачных средах // Программные системы и вычислительные методы. 2025. № 2. DOI: 10.7256/2454-0714.2025.2.74760 EDN: WIHQAD URL: https://nbpublish.com/library_read_article.php?id=74760

Интеллектуальная инфраструктура автоматизированного управления и интероперабельности микросервисов в облачных средах

Рогов Дмитрий Вадимович

студент; кафедра Инструментального и Прикладного Программного Обеспечения; Федеральное государственное бюджетное образовательное учреждение высшего образования "МИРЭА - Российский технологический университет"

123007, Россия, г. Москва, Хорошевский р-н, Хорошёвское шоссе, д. 60

1664286@gmail.com

Алпатов Алексей Николаевич

кандидат технических наук

доцент; кафедра Инструментального и Прикладного Программного Обеспечения; Федеральное государственное бюджетное образовательное учреждение высшего образования "МИРЭА - Российский технологический университет"

111033, Россия, г. Москва, р-н Лефортово, Танковый проезд, д. 4А, кв. 24

aleksej01-91@mail.ru

Статья из рубрики "Операционные системы "

DOI:

10.7256/2454-0714.2025.2.74760

EDN:

WIHQAD

Дата направления статьи в редакцию:

09-06-2025

Дата публикации:

19-06-2025

Аннотация: В условиях стремительного роста масштабов и сложности информационных

систем, вопросы эффективной интеграции и сопровождения микросервисных архитектур становятся всё более актуальными. Одной из ключевых проблем является обеспечение интероперабельности программных компонентов, что предполагает возможность надёжного обмена данными и совместного использования информации между различными сервисами, реализованными с использованием разнородных технологий, протоколов и форматов данных. В данной работе предметом исследования выступает формализация и построение интеллектуальной системы, обеспечивающей интероперабельность микросервисных компонентов в облачной инфраструктуре. Предложен формализованный подход, основанный на графовых, категориальных и алгебраических моделях, позволяющий строго описывать маршруты передачи данных, условия совместимости интерфейсов и процедуру автоматизированного согласования форматов взаимодействия. Введена операция согласования интерфейсов, обеспечивающая выявление необходимости использования адаптеров и преобразователей для интеграции различных сервисов. Особое внимание уделяется задаче построения универсального интерфейса, через который возможна маршрутизация любых потоков данных, что значительно упрощает процедуру масштабирования и доработки микросервисной системы. Разработанная архитектура системы охватывает этапы создания, публикации и развертывания контейнерных микросервисов, автоматическую проверку маршрутов передачи данных, а также динамическое управление состоянием сервисов на основе прогнозирования нагрузки с помощью моделей искусственного интеллекта. Применение предложенной методики позволяет существенно повысить гибкость, надёжность и масштабируемость инфраструктуры, снизить эксплуатационные затраты, а также автоматизировать процессы поддержки и интеграции новых компонентов. Предложенное решение основывается на формализованном подходе к обеспечению интероперабельности микросервисных компонентов в облачной инфраструктуре. В качестве основы используется графовая и категорная модель, позволяющая строго определить маршруты передачи данных и процедуры согласования интерфейсов между различными сервисами. Для унификации взаимодействия и повышения гибкости системы введена операция согласования интерфейсов, а также реализована возможность автоматизированного выявления необходимости применения адаптеров и преобразователей данных. Разработанный алгоритм интеллектуального прогнозирования нагрузки на сервисы позволяет динамически управлять состоянием компонентов и оперативно адаптировать инфраструктуру к изменяющимся условиям эксплуатации.

Ключевые слова:

Микросервисная архитектура, Интероперабельность, Контейнеризация, Облачная инфраструктура, Согласование интерфейсов, Категорная модель, Графовая модель, Автоматизация деплоя, Адаптеры, Интеллектуальное управление

Введение

В последние годы наблюдается стремительный рост масштабов и сложности информационных систем, что обусловливает высокие требования к их сопровождению, масштабируемости и интеграции [1]. Одной из ключевых тенденций современного этапа развития отрасли становится переход к микросервисной архитектуре, которая позволяет создавать гибкие, легко масштабируемые и модульные программные продукты. Микросервисы, функционируя в единой инфраструктуре, обеспечивают независимость

разработки, развертывания и обновления отдельных компонентов, что значительно повышает эффективность работы команд сопровождения и эксплуатации.

В то же время внедрение микросервисных систем сопряжено с необходимостью обработки больших объёмов разнородных данных, поступающих от различных источников. Для преобразования такой информации компании выстраивают ETL процессы. Данный подход позволяет преобразовать данные и сохранить в целевом хранилище. Но такие системы также требуют для своей работы набор сервисов, который выполнит преобразование необходимой информации. В условиях возрастающей сложности и масштабности систем для ИТ-компаний становится крайне актуальной задача оптимизации затрат на построение и сопровождение микросервисной инфраструктуры, а также повышения эффективности процессов поддержки и интеграции.

Одним из перспективных подходов к решению данной проблемы выступает использование систем интероперабельности программных компонентов [2], которые обеспечивают автоматизацию процессов создания, развертывания и управления микросервисами, а также позволяют реализовать графические интерфейсы взаимодействия между ними. Такие системы обладают большим функционалом по созданию, развертыванию и управлению микросервисами, а также позволяют реализовать графические интерфейсы взаимодействия между ними, что способствует снижению эксплуатационных расходов и повышению устойчивости современных информационных систем.

На практике для этих целей нередко используются промышленные платформы, такие как SAS RTDM [3] и аналогичные решения. Однако проведённый анализ показывает, что существующие системы зачастую не обеспечивают необходимой гибкости при интеграции современных технологий, не поддерживают автоматическое интеллектуальное управление ресурсами и не способны оперативно реагировать на динамические изменения во входных потоках данных. Это существенно ограничивает возможности развития, масштабирования и эксплуатации информационных систем, а также приводит к росту эксплуатационных затрат.

В связи с этим актуальной задачей является разработка архитектуры интеллектуальной системы, способной обеспечить высокую степень интероперабельности программных компонентов, автоматическое управление состоянием сервисов и адаптацию к изменяющимся условиям эксплуатации.

Научная новизна работы заключается в разработке метода и обеспечения интероперабельности программных компонентов, а также применение моделей искусственного интеллекта для прогнозирования поступающей на сервисы нагрузки с целью управления состоянием программных компонент. Предметом исследования являются методы обеспечения интероперабельности программных компонентов, а также интеллектуальные методы прогнозирования нагрузки, поступающей на программные компоненты.

Постановка задачи обеспечения интероперабельности программных компонентов

Прежде чем приступим к рассмотрению способа организации системы интеграции для микросервисной архитектуры, дадим определение данному понятию и формализируем данную задачу.

Под интерперабельностью в современных программных системах понимают, как способность двух или более информационных систем или их компонентов к обмену

информацией и использованию полученных данных в совместной работе [4]. Вместе с тем, эволюция цифровых технологий и усложнение архитектур привели к существенному расширению содержания данного термина. Если изначально интероперабельность рассматривалась преимущественно в техническом аспекте, а именно как синтаксическая и протокольная совместимость, то современный подход требует учитывать семантические, организационные, законодательные и даже политические аспекты взаимодействия систем, что отражено в ряде стандартов [5]. Так например, в российском стандарте ГОСТ Р 55062–2012 выделяются технический, семантический и организационный уровни. Технический уровень связан с форматами, протоколами и средствами передачи информации, семантический связан с единым толкованием и трактовкой данных, а организационный связан с согласованием бизнес-целей, регламентов и правовых условий взаимодействия. Таким образом, интероперабельность на современном этапе рассматривается как многогранное понятие, охватывающее широкий спектр аспектов, без которых невозможно построение масштабируемых и взаимосвязанных информационных систем, в том числе, основанных на микросервисной архитектуре.

Микросервисное взаимодействие может быть описано, через граф

$G = (V, E, \mathcal{T}, \mathcal{P})\#(1)$

где $V = (s_1, s_2, \ldots, s_n)$ - множество вершин (микросервисов),

$E \subseteq V \times V \times \mathcal{T}$- множество рёбер, каждое из которых связано с типом взаимодействия,

τ - множество типов взаимодействия (REST, gRPC, SOAP и др.),

$\mathcal{P}: E \to [0,1]$- функция, задающая вероятность передачи сообщения по ребру (стоимость, надёжность канала и т.д.).

Стоит понимать, что в современных системах данные могут проходить через цепочки сервисов и преобразований, и необходимо учитывать, какие маршруты совместимы, где есть ограничения, а также наличие новых возможностей для интеграции. Для этого расширим модель, путём внедрения категорной структуры [6]. На основе графа (1) строится категория $\mathcal{C}_G$, где каждому микросервису ставится в соответствие объект $\mathrm{Obj}(\mathcal{C}_G) = V$, а каждому ребру $(s_i, s_j, t_{ij}) \in E$ сопоставляется элементарный морфизм $f_{ij}^{(t_{ij})}: s_i \to s_j$, который моделирует непосредственное взаимодействие между сервисами. Если существуют морфизмы (отображения) $f: s_i \to s_j$ и $g: s_j \to s_k$, то их композиция $g \circ f: s_i \to s_k$позволяет описать сложные маршруту передачи данных или комбинированную обработку. Для каждого микросервиса $s_i \in V$также определён тождественный морфизм $\mathrm{id}_{s_i}: s_i \to s_i$, соответствующий отсутствию какого-либо преобразования. Таким образом, любой путь в графе, то есть любая последовательность рёбер, индуцирует составной морфизм в категории. Это позволяет строго и формально описывать сложные сценарии взаимодействия между микросервисами. При этом, поскольку каждое ребро содержит тип взаимодействия t_{ij}, данный атрибут выступает аннотацией соответствующего морфизма, что даёт возможность учитывать специфику используемых протоколов, форматов данных или других технических особенностей на уровне категорной модели. Также такой подход принципиально важен для анализа совместимости, поскольку он позволяет моделировать не только прямые взаимодействия, но и цепочки передачи данных, а также обосновывать необходимость построения адаптеров и промежуточных преобразователей. Рассмотрим пример, пусть имеется крупная компания, у которой есть распределённая

система, где каждый микросервис выполняет отдельную бизнес-функцию. Так сервис A отвечает за авторизацию пользователей, принимая на вход запросы по REST, отдавая JSON данные. Сервис B реализует механизмы антифрода, то есть проверку транзакций, при это он принимает JSON по REST, отдаёт структурированные события в формате Protobuf по gRPC. Сервис C реализует биллинг, получая события по gRPC (Protobuf), преобразуя и записывая платёжные операции. Сервис D реализует хранилище данных принимая отчёты в формате Avro по HTTP POST. Формируется цепочка действий, когда пользователь аутентифицируется через Сервис A, далее его транзакция отправляется на проверку в Сервис B, а после проверки результат отправляется в Сервис C. Для построения отчёта данные из сервиса C агрегируются и отправляются в сервис D. Цепочка взаимодействий $A \to B \to C \to D$ возможна только при наличии совместимых форматов/протоколов на всех промежуточных этапах, либо адаптеров между ними. На рисунке 1 показана категорная модель маршрута передачи данных для данного примера.

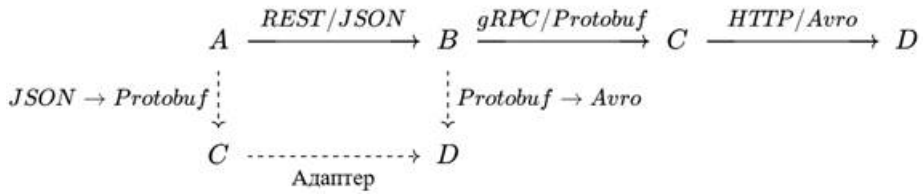

Рисунок 1 - Категорная диаграмма, моделирующая маршрут передачи данных между микросервисами

Расширение модели за счёт внедрения категорий позволяет выявлять такие ситуации, когда если хотя бы на одном участке нет преобразователя, то путь (композиция морфизмов) не существует, и такой маршрут автоматически исключается из множества допустимых интеграций, что позволяет автоматически контролировать процесс разрешимости маршрутов интеграции, даже в случае их динамического изменения. Для этого определим базовую операцию согласования структуры данных и интерфейсов.

Пусть каждому микросервису $s_i \in V$ сопоставлено множество поддерживаемых им типов данных D_i множество реализованных интерфейсов I_i. Тогда общее множество всех интерфейсов в системе может быть определено по формуле

$$\mathcal{I} = \bigcup_{i=1}^{n} I_i, \#(2)$$

Введём бинарную операцию (согласован или нет) согласования интерфейсов, которая может быть определена, как

$$\otimes: \mathcal{I} \times \mathcal{I} \to \mathcal{I} \cup \{\emptyset\}, \#(3)$$

Тогда для любых двух интерфейсов $I_i, I_j \in \mathcal{I}$ может быть определена операция

$$I_i \otimes I_j = \begin{cases} I_k, \text{если } \exists \mathcal{C}_{ci} : I_i \to I_j \\ \emptyset, \text{если } \nexists \mathcal{C}_{ci}, \end{cases} \quad (4)$$

где I_k - наименьший общий интерфейс,

C_{ij} - набор преобразований (конвертеров), обеспечивающий совместимость.

Определим ряд важных свойств операций согласования. Прежде всего стоит понимать, что не для всех пар интерфейсов, то есть на $(\mathcal{I}, \otimes)$ существует частично определённая алгебраическая структура, аналогичная полугруппе с нулевым элементом $(\emptyset)$. Это

означает, что если хотя бы на одном этапе согласования возникает несовместимость (то есть результатом является $(\varnothing)$, то при любой дальнейшей композиции результат также будет $(\varnothing)$. Для задачи обеспечения интероперабельности это будет означать, что если последовательное согласование интерфейсов в цепочке взаимодействия не удаётся хотя бы между одной парой компонентов, то передача данных по этому маршруту невозможна. Также операция согласования может обладать ассоциативностью, если если промежуточные интерфейсы существуют, а также если для цепочек интерфейсов результат не зависит от порядка выполнения преобразований, то есть конвертеры транзитивны

$$(I_i \otimes I_j) \otimes I_k = I_i \otimes (I_j \otimes I_k). \#(5)$$

Формула 5 фактически указывает на наличие в системе такого интерфейса, к которому, при наличии конвертеров между интерфейсами I_i и I_j а также при наличии конвертера между I_j и I_i, всегда можно согласовать любой другой интерфейс. В рамках данной работы данный интерфейс будет называться универсальным. Рассмотрим пример, пусть сервисы в микросервисной системе могут использовать разные форматы данных (JSON, XML, Protobuf), и между некоторыми парами форматов есть адаптеры. Если при этом для любого формата реализован переход к JSON и обратно (и эти переходы ассоциативны и транзитивны), то JSON будет являться универсальным интерфейсом для данной микросервисной системы. Тогда можно согласовать между собой любые два сервиса через промежуточный формат, не заботясь о прямых адаптерах между всеми возможными парами форматов, что упрощает интеграцию системы.

В условиях микросервисной архитектуры микросервисы обмениваются информацией, что можно описать как

$$m_{ij}^t : s_i \to s_j, \quad \forall (s_i, s_j) \in E \ \#(6)$$

где $m_{ij}^t \in \mathrm{M_{ij}}$ - отображение сообщения $\mathrm{M_{ij}}$ из s_i в s_j.

Тогда каждый сервис может быть представлен как функция обработки поступающих сообщений

$$m_{ij}^t \to m_{jk}^{t+1}, \quad \forall s_k \in V, (s_j, s_k) \in E \#(7).$$

Способ организации интероперабельности программных компонентов

Формализованная выше модель интероперабельности, основанная на графовых, категориальных и алгебраических структурах, позволяет строго описывать и анализировать совместимость компонентов, а также выявлять ключевые требования к организации инфраструктуры. Полученные выше результаты прямо указывают на необходимость согласования интерфейсов сервисов на всех этапах передачи данных, а также обеспечения гибкости и расширяемости инфраструктуры, с целью оперативной адаптации к изменяющимся условиям эксплуатации и динамике потоков данных. Поэтому стоит более подробно рассмотреть механизм построения микросервисов и управления инфраструктурой.

Рассматривая построение микросервисов в облачной инфраструктуре, необходимо учитывать, что в большинстве платформ используются заранее собранные образы сервисов, хранящиеся в отдельном репозитории. Каждый сервис развёртывается на основе заранее подготовленного контейнерного образа, в который включаются все необходимые зависимости, конфигурации и описания интерфейсов. Контейнеры не

являются полноценными операционными системами, настроенными для общего использования. Скорее, это минимальные среды для запуска приложений [7]. Данные представлены в виде слоев, накладываемых друг на друга, что показано на рисунке 2. Такие «сборки слоев» хранятся в централизованном реестре контейнерных образов и интерфейсов (репозитории), а сборка таких образов для однотипных систем происходит по одинаковым конфигурационным файлам, что позволяет однозначно воспроизводить среду исполнения.

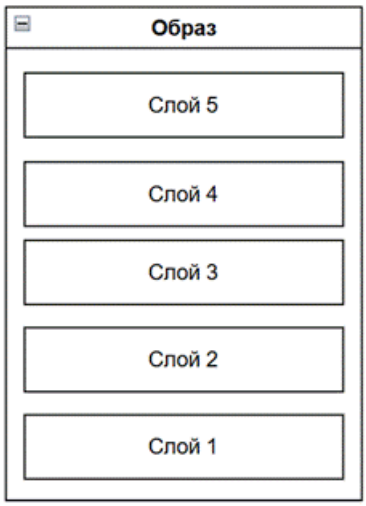

Рисунок 2 – Представление образа контейнера

Сформулированная ранее алгебраическая модель согласования интерфейсов реализуется в виде процедуры автоматической проверки маршрутов передачи данных. Пусть каждому контейнерному образу C_i сопоставлено множество поддерживаемых им интерфейсов $I(C_i)$. Тогда при развёртывании системы выполняется проверка

$$\forall (C_i, C_j) \in \text{DeploymentPlan}: \exists\, \mathcal{C}_{ij}: I(C_i) \to I(C_j) \quad (8)$$

где C_i, C_j - контейнерные образы (сервисы),

$I(C_i)$ - множество поддерживаемых интерфейсов образа C_i,

$\mathcal{C}_{ij}$ - морфизм, обеспечивающий согласование интерфейсов между C_i и C_j.

Если хотя бы для одной пары (C_i, C_j) не существует $\mathcal{C}_{ij}$, то маршрут деплоя считается неразрешимым и отклоняется системой.

Категорная модель (диаграмма передачи данных) непосредственно используется для автоматизированной проверки маршрутов передачи данных в момент развёртывания. Практически это реализуется за счёт описания контейнерного образа, в котором задаются формальные характеристики поддерживаемых интерфейсов, после чего категорные морфизмы служат алгоритмической основой для проверки того, что в цепочке взаимодействий все необходимые конвертеры и адаптеры существуют. Если такой морфизм отсутствует, автоматизированная система отказывает в развёртывании соответствующего маршрута.

Для сборки образов предполагается использовать программное обеспечение Docker. Оно позволяет производить такую сбору удалённо. Для таких операций используется встроенная утилита «демона», способная выполнить сборку за короткое время. Далее

для развертывания необходимо непосредственно в облачной инфраструктуре указать необходимый репозиторий образов, в котором находится слепок созданного сервиса и выполнить развертывание данного образа непосредственно в облачной инфраструктуре. В качестве такой инфраструктуры стоит использовать кластер Kubernrtes. Он является основой крупнейших облачных инфраструктур [8]. Алгебраическая операция согласования интерфейсов реализуется как программный сервис, который получает на вход описания интерфейсов каждого контейнера и, используя матрицу совместимости, автоматически определяет необходимость в адаптерах. Это позволяет системе Kubernetes перед развёртыванием проверять совместимость интерфейсов через специализированный API.

Развёртывание целесообразно проводить путём взаимодействия с целевым API облачной инфраструктуры. Такое взаимодействие будет наиболее быстрым, по сравнению с вариантами ручного развёртывания и развёртывания в режиме, когда все команды выполняются в пользовательских утилитах автоматически.

На рисунке 3 представлена диаграмма взаимодействия компонент системы непосредственно между собой. Данные типы взаимодействия выбраны с учётом особенностей построения и сопровождения подобных систем и описанной выше архитектурой.

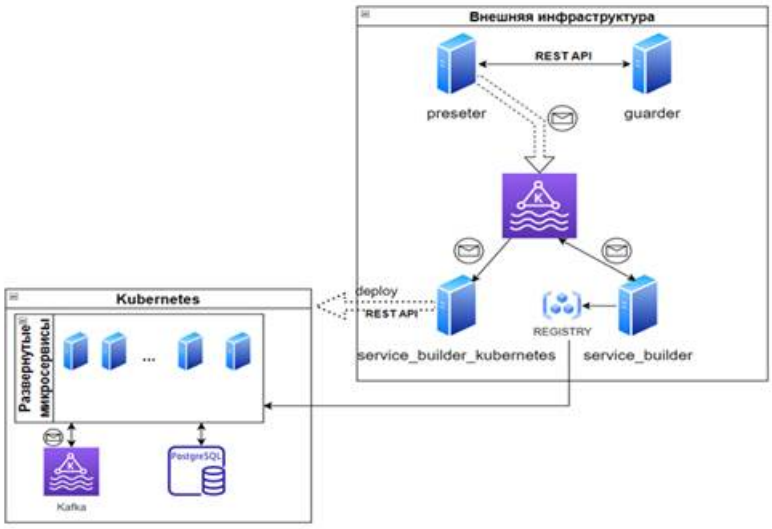

Рисунок 3 - Схема взаимодействия компонентов системы

Разработанный выше способ организации интероперабельности компонентов обеспечивает не только автоматизированную проверку совместимости и согласования интерфейсов между микросервисами, но и служит основой для оперативного управления и мониторинга инфраструктуры в условиях изменяющейся нагрузки и динамического окружения. Для обеспечения надёжного функционирования и оперативной адаптации к изменениям в условиях эксплуатации требуется постоянный мониторинг состояния сервисов и прогнозирование будущих нагрузок. Такой подход позволит не просто своевременно реагировать на возможные проблемы, но и заблаговременно перераспределять ресурсы между микросервисами, минимизируя время простоя и повышая общую эффективность работы системы. Далее рассмотрим метод прогнозирования нагрузки на компоненты инфраструктуры.

Метод прогнозирования нагрузки

Стоит рассмотреть метод прогнозирования нагрузки на компоненты системы

интероперабельности программных компонентов. Ключевой потенциал систем анализа данных кроется в извлечении полезных выводов из данных. Цель данного процесса состоит в том, чтобы ответить на интересные (а иногда и трудные) вопросы, используя технологии анализа данных, и достичь более полного понимания конкретной предметной области [9]. Рассмотрение такого метода стоит начать с описания алгоритма сбора метрик сервисов и алгоритма прогнозирования нагрузки.

Рассматривая алгоритм сбора метрик программных компонентов, стоит обратить внимание на то, что он должен быть адаптирован для работы с различным количеством микросервисов. Таким образом, разработанный алгоритм сбора метрик выполняет периодический сбор показателей каждого микросервиса. Разработанная блок-схема его работы представлена на рисунке 4.

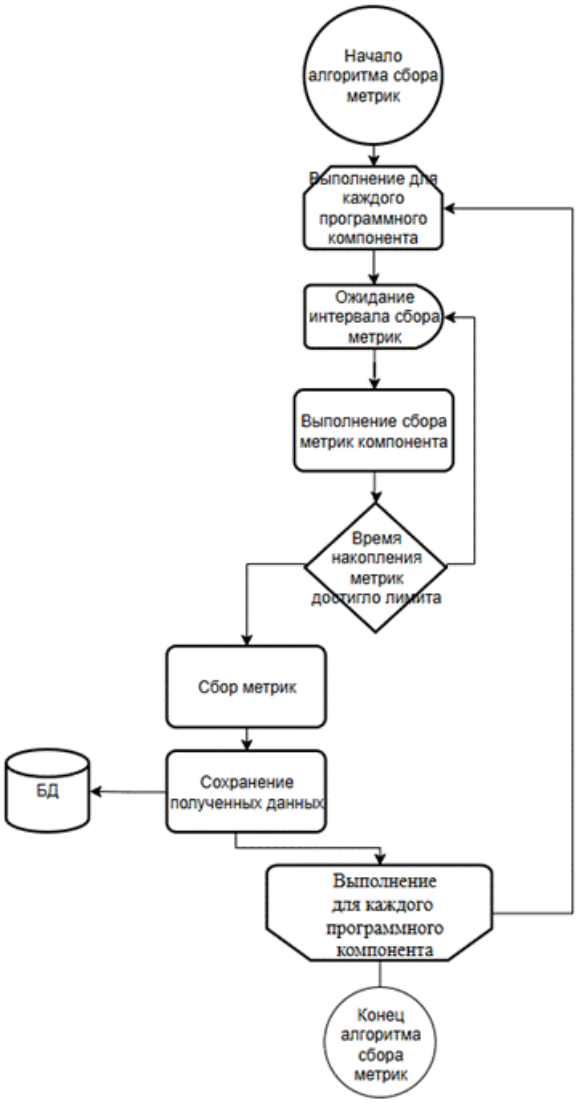

Рисунок 4 – Блок-схема алгоритма сбора метрик

Стоит сказать, что программные компоненты представляют из себя микросервисы, развёрнутые в облачной инфраструктуре и сопровождаемые самой системой. Поэтому для прогнозирования можно использовать следующий алгоритм, представленный на рисунке 5. Такой алгоритм обладает возможностью работы для множества развёрнутых и контролируемых системой интероперабельности программных компонентов.

Рисунок 5 – Блок-схема алгоритма прогнозирования

Из представленного описания алгоритма прогнозирования можно выбрать программные средства для его осуществления. Для реализации такого сервиса наилучшим образом подойдет язык программирования Python. Он содержит все необходимые библиотеки для работы моделей искусственного интеллекта, а также позволяет выполнять преобразование поступаемой информации в агрегаты.

Рассмотрим сам сервис запуска моделей. Его детальная архитектура представлена на рисунке 3. Можно увидеть, что такой сервис состоят из нескольких внутренних компонентов, каждый из которых отвечает за свою часть.

Следует детально рассмотреть работу компонента выполнения моделей. Он отвечает за запуск и функционирование нейронных сетей. Такой модуль выполняет загрузку нейронной сети из базы данных или внутреннего хранилища и запускает её с полученными на вход данными. В результате работы моделей компонент получает результат прогнозирования, который в дальнейшем может быть обработан.

Пример кода выгрузки модели из базы данных показан на листинге 1. Представленная функция организует загрузку нейронной сети из базы данных, а также загружает

созданный заранее весы для слоёв сети.

Листинг 1 -

```
def load_model_from_db(model_name):

global model, tokenizer, model_params

try:

conn = psycopg2.connect(**DB_CONFIG)

cursor = conn.cursor()

# Загружаем модель

cursor.execute("SELECT model_data FROM ai_models WHERE model_name = '"+ model_name +"' AND model_active = True")

model_data = cursor.fetchone()[0]

model = load_model(io.BytesIO(model_data))

# Загружаем параметры

cursor.execute("SELECT params FROM ai_model_params WHERE model_name = '"+ model_name +"'")

model_params = cursor.fetchone()[0]

cursor.close()

conn.close()

print("Модель загружена из БД")

except Exception as e:

print(f"Ошибка загрузки модели из БД: {str(e)}")

raise e
```

Выгрузка нейронной сети выполняется стандартными методами доступа до хранилища данных. Стоит заметить, что сами нейронные сети, а также их параметры хранятся в реляционной базе данных, что позволяет хранить множество моделей в структурированном виде.

Библиотеки языка программирования Python позволяют выполнить инициализацию самой модели в несколько строк кода. Это особенно удобно при разработке сервиса, в котором необходимо выполнять загрузку моделей.

Стоит рассмотреть и процесс работы самой модели. Функция генерации прогнозирования представлена на листинге 2.

Листинг 2

```
def predict():

global prediction_buffer
```

```
# Получаем данные из POST запроса
data = request.get_json()
# Проверяем обязательные поля
if not all(k in data for k in ['cpu', 'ram', 'timestamp']):
return jsonify({'error': 'Missing required parameters (cpu, ram, timestamp)'}), 400
try:
cpu = float(data['cpu'])
ram = float(data['ram'])
timestamp = data['timestamp']
combined_load = 0.6 * cpu + 0.4 * ram
# Добавляем в буфер
prediction_buffer.append(combined_load)
# Если буфер заполнен, делаем предсказание
if len(prediction_buffer) >= SEQ_LENGTH:
sequence = prediction_buffer[-SEQ_LENGTH:]
sequence_scaled = scaler.transform(np.array(sequence).reshape(-1, 1))
X = sequence_scaled.reshape(1, SEQ_LENGTH, 1)
predicted_load_scaled = model.predict(X)
predicted_load = scaler.inverse_transform(predicted_load_scaled)[0][0]
if len(prediction_buffer) > SEQ_LENGTH * 2:
prediction_buffer = prediction_buffer[-SEQ_LENGTH:]
return jsonify({
'timestamp': timestamp,
'current_load': combined_load,
'predicted_load': float(predicted_load),
'message': 'Success'
})
else:
return jsonify({
'timestamp': timestamp,
```

'current_load': combined_load,

'predicted_load': None,

'message': f'Collecting data, need {SEQ_LENGTH - len(prediction_buffer)} more samples'

})

except Exception as e:

return jsonify({'error': str(e)}), 500

Видно, что данная функция получает объект с данными и на основании него выполняет запуск модели и получение результатов прогнозирования для полученных данных – метрик, собранных при работе сервисов.

Затем результат прогнозирования проходит интерпретацию – корректировку результата и перевод её в значения необходимости корректировки, а затем отправляется в виде сообщений в нужный топик шины.

Стоит также обсудить и само построение модели. В предлагаемой системе наилучшим образом подходит использование LSTM нейронных сетей. Они позволяют проводить ретроспективный анализ работы компонентов с возможностью отслеживания периодов однообразности, поступающей на развёрнутые системой интероперабельности программные компоненты нагрузки. Построение модели происходит путём проведения её сборки и обучения в Jupiter ноутбуках. Это веб-приложение с открытым исходным кодом, которое позволяет создавать и обмениваться документами, содержащими живой код, уравнения, визуализацию и разметку [10].

Рассматривая процесс построения и обучения нейронной сети, нужно обратить внимание, что существующие библиотеки для Python, такие как tensorflow и sklearn позволяют выполнять построение любых моделей искусственного интеллекта с применением минимального количества кода. Также подразумевается, что система интеллектуального управления нагрузкой будет использовать различные нейронные сети для анализа большого количества метрик, для обеспечения возможности наиболее точного прогнозирования нагрузки.

Для примера спроектирована и разработана модель, позволяющая прогнозировать необходимость изменения состояния программных компонентов в части изменения выделенных квот CPU и памяти. Пример части кода по построению и обучению такой модели приведён на листинге 3.

Листинг 3

def build_lstm_model(input_shape, received_data):

model = Sequential([

LSTM(64, return_sequences=True, input_shape=input_shape),

Dropout(0.2),

LSTM(32, return_sequences=False),

Dropout(0.2),

```
Dense(16, activation='relu'),
Dense(1)
])
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mse')
return model
SEQ_LENGTH = 60
TEST_SIZE = 0.2
BATCH_SIZE = 32
EPOCHS = 100
data, scaler = load_and_preprocess_data(received_data)
train_size = int(len(data) * (1 - TEST_SIZE))
train_data = data['Load_scaled'].values[:train_size]
test_data = data['Load_scaled'].values[train_size:]
X_train, y_train = create_sequences(train_data, SEQ_LENGTH)
X_test, y_test = create_sequences(test_data, SEQ_LENGTH)
model = build_lstm_model((X_train.shape[1], X_train.shape[2]))
print(model.summary())
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
history = model.fit(
X_train, y_train,
batch_size=BATCH_SIZE,
epochs=EPOCHS,
validation_data=(X_test, y_test),
callbacks=[early_stopping],
verbose=1
)
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
train_predict = scaler.inverse_transform(train_predict)
```

```
y_train = scaler.inverse_transform([y_train])

test_predict = scaler.inverse_transform(test_predict)

y_test = scaler.inverse_transform([y_test])

train_rmse = np.sqrt(mean_squared_error(y_train[0], train_predict[:,0]))

test_rmse = np.sqrt(mean_squared_error(y_test[0], test_predict[:,0]))

print(f'Train RMSE: {train_rmse:.2f}')

print(f'Test RMSE: {test_rmse:.2f}')

plt.figure(figsize=(15, 6))

plt.plot(data['Timestamp'], data['Load'], label='Actual Load')

train_predict_plot = np.empty_like(data['Load'])

train_predict_plot[:] = np.nan

train_predict_plot[SEQ_LENGTH:SEQ_LENGTH+len(train_predict)] = train_predict[:,0]

test_predict_plot = np.empty_like(data['Load'])

test_predict_plot[:] = np.nan

test_predict_plot[train_size+SEQ_LENGTH:train_size+SEQ_LENGTH+len(test_predict)] =
test_predict[:,0]

plt.plot(data['Timestamp'], train_predict_plot, label='Train Predict')

plt.plot(data['Timestamp'], test_predict_plot, label='Test Predict')

plt.legend()

plt.title('CPU and RAM Load Prediction')

plt.xlabel('Time')

plt.ylabel('Combined Load')

plt.show()
```

Таким образом, предложенный подход организации интероперабельности и модель прогнозирования нагрузки обеспечивают возможность не только оперативного контроля текущего состояния инфраструктуры, но и её заблаговременного адаптивного изменения. Однако для практической реализации этих преимуществ необходима систематическая и надёжная процедура сбора, хранения и обработки текущих данных о состоянии микросервисов, в том числе в части для возможности дальнейшего ретроспективного анализа работы сетей и возможности корректировки используемых нейронных сетей.

Организация мониторинга, прогнозирования и управления состоянием компонентов облачной инфраструктуры

Как было отмечено выше для реализации возможности управления состоянием

программных компонентов необходимо обеспечить сбор метрик с создаваемых системой интероперабельности программных компонентов. Это можно реализовать с использованием ПО Promrtheus. Получаемые метрики стоит сохранять в хранилище, которое имеет возможность компактного хранения больших объёмов данных.

Также стоит позаботиться о вопросе транспортировки собранных метрик до хранилища. Наиболее простой способ передачи данных – центральная шина. Такой подход позволит унифицировать передачу различных метрик в общих сообщениях. Также использование шины позволит гарантировать доставку метрик до хранилища и сохранение их за счёт использования надёжных брокеров сообщений, таких как Apache Kafka.

Для прогнозирования нагрузки необходимо использовать отдельный сервис, который позволяет загрузить нейронную сеть и её веса и произвести прогнозирование. Стоит предусмотреть возможность постоянной работы некоторых сетей, путём хранения их в краткосрочной памяти непосредственно внутри сервиса для ускорения работы сервиса по прогнозированию нагрузки.

Результаты работы такого сервиса необходимо сохранять и валидировать. Например, не должно быть ситуации прогнозирования необходимости уменьшения ресурсов меньше минимального значения для таких сервисов. Стоит также применять механизмы авторизации и аутентификации при работе такого сервиса для исключения возможности несанкционированного управления состоянием программных компонент. Полученные же прогнозные значения нагрузки от нейросетевой модели используются системой оркестрации Kubernetes для автоматического масштабирования сервисов. Так, при превышении прогнозного порога по нагрузке, Kubernetes автоматически увеличивает количество реплик сервиса, что предварительно описывается через правила горизонтального масштабирования (Horizontal Pod Autoscaler), опирающиеся на результаты работы нейросетевых моделей.

Стоит более детально рассмотреть предлагаемую архитектуру сервисов получения метрик, прогнозирования и управления облачной инфраструктурой. Такая архитектура приведена на рисунках 6 и 7.

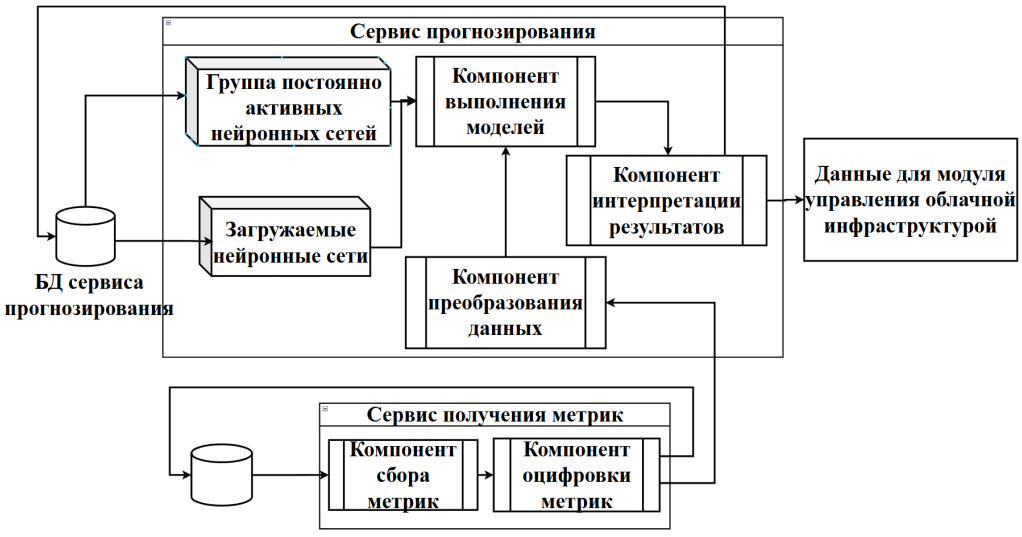

Рисунок 6 – Архитектура сервисов прогнозирования и получения метрик системы

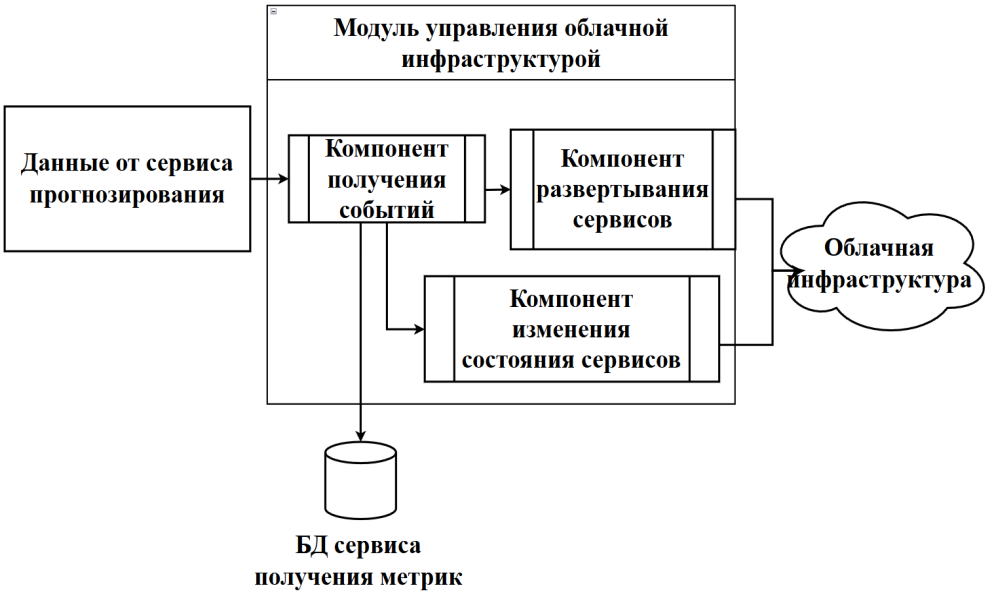

Рисунок 7 – Архитектура модуля управления облачной инфраструктурой

Можно видеть, что каждый сервис состоит из нескольких активных компонентов, совместно выполняющих работу. Также можно видеть, что каждый сервис взаимодействует со своей базой данных, при этом межсервисное взаимодействие должно осуществляться при помощи общей шины для обеспечения надёжности работы сервисов.

Заключение

Предложенное решение основывается на формализованном подходе к обеспечению интероперабельности микросервисных компонентов в облачной инфраструктуре. В качестве основы используется графовая и категорная модель, позволяющая строго определить маршруты передачи данных и процедуры согласования интерфейсов между различными сервисами. Для унификации взаимодействия и повышения гибкости системы введена операция согласования интерфейсов, а также реализована возможность автоматизированного выявления необходимости применения адаптеров и преобразователей данных. Такой подход обеспечивает формальное структурирование интеграционных процессов и способствует масштабируемости архитектуры. Контейнерная организация сервисов и централизованное управление образами позволяют автоматизировать этапы создания, публикации и развёртывания компонентов, обеспечивая воспроизводимость и высокую надёжность инфраструктуры. Особое внимание уделяется автоматической проверке совместимости интерфейсов на этапе деплоя, что исключает невозможные или неразрешимые маршруты передачи данных. Разработанный алгоритм интеллектуального прогнозирования нагрузки на сервисы позволяет динамически управлять состоянием компонентов и оперативно адаптировать инфраструктуру к изменяющимся условиям эксплуатации.

Рассмотренное решение предоставляет заявленный функционал, но имеет и возможности дальнейшей модернизации. Стоит рассмотреть возможность более гибкого управления и интерпретации метрик, которые анализирует система. Также стоит рассмотреть возможность ускорения обработки данных и работы моделей искусственного интеллекта за счет применения технологий, обеспечивающих максимальное быстродействие.

Таким образом, итоговое решение представляет собой интеграцию формальных методов моделирования, автоматизации процессов управления микросервисами и

интеллектуального прогнозирования, что обеспечивает устойчивое функционирование и эффективную поддержку современных распределённых информационных систем.

Библиография

1. Paul A., Kelvin L., Brown K. Optimizing IT Growth: Strategies for Building and Scaling Robust Infrastructure Systems // Ladoke Akintola University of Technology. 2024. Т. 17. URL: https://www.researchgate.net/publication/377447014.
2. Макаренко С. И., Олейников А. Я., Черницкая Т. Е. Модели интероперабельности информационных систем // Системы управления, связи и безопасности. 2019. № 4. DOI: 10.24411/2410-9916-2019-10408 EDN: NACKGD.
3. Дьяков О. А., Солянов К. С. Формирование кредитного конвейера банка на основе систем бизнес-аналитики // Стратегии бизнеса. 2016. № 7 (27). С. 7-12. EDN: WWJUUB.
4. ISO/IEC 2382:2015 Information technology. Vocabulary. – 2015. – Текст: электронный. URL: https://www.iso.org/ru/standard/63598.html
5. Михневич С. Ю., Тежар А. А. Эволюция понятия интероперабельности открытых информационных систем // Цифровая трансформация. 2023. Т. 29. № 2. С. 60-66. URL: https://doi.org/10.35596/1729-7648-2023-29-2-60-66. EDN: RKSXQW.
6. Ковалев С. П. Теоретико-категорный подход к проектированию программных систем // Фундаментальная и прикладная математика. 2014. Т. 19. № 3. С. 111-170.
7. Сейерс Э. Х., Милл А. Docker на практике / Э. Х. Сейерс, А. Милл ; перевод с английского Д. А. Беликов. Москва : ДМК Пресс, 2020. 516 с. ISBN 978-5-97060-772-5. Текст : электронный // Лань : электронно-библиотечная система. URL: https://e.lanbook.com/book/131719 (дата обращения: 04.06.2025). Режим доступа: для авториз. пользователей.
8. Маркелов А. А. Введение в технологию контейнеров и Kubernetes / А. А. Маркелов. Москва : ДМК Пресс, 2019. 194 с. ISBN 978-5-97060-775-6. Текст : электронный // Лань : электронно-библиотечная система. URL: https://e.lanbook.com/book/131702 (дата обращения: 04.06.2025). Режим доступа: для авториз. пользователей.
9. Бонцанини М. Анализ социальных медиа на Python. Извлекайте и анализируйте данные из всех уголков социальной паутины на Python / М. Бонцанини ; перевод с английского А. В. Логунова. Москва : ДМК Пресс, 2018. 288 с. ISBN 978-5-97060-574-5. Текст : электронный // Лань : электронно-библиотечная система. URL: https://e.lanbook.com/book/108129 (дата обращения: 09.06.2025). Режим доступа: для авториз. пользователей.
10. Груздев А. В. Предварительная подготовка данных в Python / А. В. Груздев. Москва : ДМК Пресс, 2023. Том 1 : Инструменты и валидация. 2023. 816 с. ISBN 978-5-93700-156-6. Текст : электронный // Лань : электронно-библиотечная система. URL: https://e.lanbook.com/book/314945 (дата обращения: 09.06.2025). Режим доступа: для авториз. пользователей.

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Представленная статья на тему «Интеллектуальная инфраструктура автоматизированного управления и интероперабельности микросервисов в облачных средах» и посвящена актуальному вопросу разработки архитектуры интеллектуальной системы, способной обеспечить высокую степень интероперабельности программных компонентов, автоматическое управление состоянием сервисов и адаптацию к изменяющимся

условиям эксплуатации

Одним из перспективных подходов к решению данной проблемы выступает использование систем интероперабельности программных компонентов, которые обеспечивают автоматизацию процессов создания, развертывания и управления микросервисами, а также позволяют реализовать графические интерфейсы взаимодействия между ними. Такие системы обладают большим функционалом по созданию, развертыванию и управлению микросервисами, а также позволяют реализовать графические интерфейсы взаимодействия между ними, что способствует снижению эксплуатационных расходов и повышению устойчивости современных информационных систем.

В условиях возрастающей сложности и масштабности систем для ИТ-компаний становится крайне актуальной задача оптимизации затрат на построение и сопровождение микросервисной инфраструктуры, а также повышения эффективности процессов поддержки и интеграции. Авторами дано определение понятию «интероперабельности программных компонентов».

Авторами рассматрен алгоритм сбора метрик программных компонентов. Разработанный алгоритм сбора метрик выполняет периодический сбор показателей каждого микросервиса. Разработанная блок-схема его работы представлена в графическом виде.

Список литературы представлен российскими и зарубежными источниками по теме исследования. Стиль и язык изложения материала является достаточно доступным для широкого круга читателей. Практическая значимость статьи четко обоснована. Статья по объему соответствует рекомендуемому объему от 12 000 знаков.

Статья достаточно структурирована - в наличии введение, заключение, внутреннее членение основной части (постановка задачи обеспечения интероперабельности программных компонентов, способ организации интероперабельности программных компонентов, метод прогнозирования нагрузки, организация мониторинга, прогнозирования и управления состоянием компонентов облачной инфраструктуры).

Авторами в заключительной части статьи обозначено, что предложенное решение основывается на формализованном подходе к обеспечению интероперабельности микросервисных компонентов в облачной инфраструктуре. В качестве основы используется графовая и категорная модель, позволяющая строго определить маршруты передачи данных и процедуры согласования интерфейсов между различными сервисами. Для унификации взаимодействия и повышения гибкости системы введена операция согласования интерфейсов, а также реализована возможность автоматизированного выявления необходимости применения адаптеров и преобразователей данных.

К недостаткам можно отнести следующие моменты: из содержания статьи не прослеживается научная новизна. Отсутствует четкое выделение предмета исследования.

Рекомендуется четко обозначить научную новизну исследования, сформулировать предмет. Также будет целесообразным добавить о перспективах дальнейшего исследования.

Статья «Интеллектуальная инфраструктура автоматизированного управления и интероперабельности микросервисов в облачных средах» требует доработки по указанным выше замечаниям. После внесения поправок рекомендуется к повторному рассмотрению редакцией рецензируемого научного журнала.

Результаты процедуры повторного рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.

Со списком рецензентов издательства можно ознакомиться здесь.

Статья посвящена актуальной проблеме обеспечения интероперабельности микросервисов в облачных средах с использованием интеллектуальных методов управления и прогнозирования нагрузки. Автор исследует методы формализации взаимодействия микросервисов, а также применение моделей искусственного интеллекта для оптимизации ресурсов и повышения устойчивости информационных систем.

В работе применён комплексный подход, сочетающий теоретическое моделирование и практическую реализацию. Для формализации интероперабельности использованы графовые и категорные модели, что позволяет строго описывать маршруты передачи данных и проверять совместимость интерфейсов. Алгебраические структуры, такие как полугруппы с нулевым элементом, применяются для анализа согласованности интерфейсов. Практическая часть включает разработку алгоритмов сбора метрик, прогнозирования нагрузки на основе LSTM-сетей и интеграцию с Kubernetes для автоматического масштабирования. Методология отличается высокой строгостью и воспроизводимостью.

Тема статьи исключительно актуальна в условиях роста сложности и масштабов информационных систем. Переход к микросервисной архитектуре требует решения задач интеграции, управления ресурсами и адаптации к динамическим изменениям нагрузки. Предложенные автором методы отвечают на эти вызовы, что делает исследование востребованным как в академической среде, так и среди ИТ-специалистов.

Научная новизна работы заключается в следующих аспектах:

1. Разработка категорной модели для описания взаимодействия микросервисов, включая анализ совместимости интерфейсов и автоматическое исключение недопустимых маршрутов.
2. Введение операции согласования интерфейсов с использованием универсального интерфейса, что упрощает интеграцию разнородных компонентов.
3. Применение LSTM-сетей для прогнозирования нагрузки и динамического управления ресурсами в облачной инфраструктуре.

Эти решения представляют значительный вклад в область распределённых систем и машинного обучения.

Статья написана ясным и логичным языком, с соблюдением академических норм. Структура работы хорошо продумана: от постановки проблемы и теоретического обоснования до практической реализации и визуализации результатов. Использование графиков, блок-схем и листингов кода enhances понимание предложенных методов. Библиография включает актуальные источники, что подчёркивает глубину проработки темы.

Автор предлагает комплексное решение, сочетающее формальные методы моделирования, автоматизацию управления микросервисами и интеллектуальное прогнозирование. Результаты работы имеют высокую практическую значимость, так как позволяют снизить эксплуатационные затраты и повысить устойчивость информационных систем. Статья демонстрирует завершённость исследования, но также указывает на направления для дальнейшей модернизации, такие как оптимизация обработки данных.

Статья будет полезна исследователям в области распределённых систем, разработчикам облачных решений и специалистам по машинному обучению. Практические аспекты, такие как интеграция с Kubernetes и примеры кода на Python, делают материал ценным для ИТ-инженеров. Читательская аудитория получит не только теоретические знания, но и готовые инструменты для внедрения предложенных методов.

Статья соответствует высоким академическим стандартам, обладает значительной

научной и практической ценностью. Рекомендую её к публикации в ТОП-5 статей месяца издательства. Работа заслуживает положительной оценки благодаря оригинальности, глубине проработки и чёткому изложению. Учитывая актуальность темы и качество представленного материала, статья вызовет интерес у широкого круга специалистов и внесёт вклад в развитие области.