|  |  |
| --- | --- |
| **­** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА**

**«Управление данными в микросервисных архитектурах»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Студент гр. ИУК4-82Б | |  |  | ( | Сафронов Н.С. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Руководитель | |  |  | ( | Гагарин Ю.Е. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

Оценка руководителя \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_

30-50 (дата)

Оценка защиты \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_

30-50 (дата)

Оценка работы \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка по пятибалльной шкале)

Комиссия: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_Гагарин Ю.Е.\_\_)

(подпись) (Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_Белов Ю.С.\_\_\_\_)

(подпись) (Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_Амеличева К.А.\_\_)

(подпись) (Ф.И.О.)

Калуга, 2024

# *Калужский филиал федерального государственного бюджетного образовательного учреждения высшего образования*

***«Московский государственный технический университет имени Н.Э. Баумана*** *(национальный исследовательский университет)»* ***(КФ МГТУ им. Н.Э. Баумана)***

УТВЕРЖДАЮ

Заведующий кафедрой **\_\_ИУК4\_\_\_**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Гагарин Ю.Е.)

« 09 » февраля 2023 г.

**ЗАДАНИЕ**

**на НАУЧНО-ИССЛЕДОВАТЕЛЬСКУЮ РАБОТУ (НИР)**

За время выполнения НИР студенту необходимо:

1. Выполнить анализ информации по заданной теме, выявить наиболее рациональные варианты решения поставленной задачи; построить математическую модель исследуемого процесса, выполнить анализ объекта исследования посредством принятой математической модели; сформулировать выводы, адекватные полученным результатам.

в том числе:

***– ознакомиться с возникающими в микросервисных архитектурах проблемах при работе с данными; рассмотреть основные механизмы для решения конкретных проблем, выявить их преимущества и недостатки.***

1. Подготовить тезисы доклада и оформить их в соответствии с установленными требованиями; выступить с докладом на научной конференции.

Дата выдачи задания « 09 » февраля 2023 г.

|  |  |  |
| --- | --- | --- |
| Руководитель НИР |  | Гагарин Ю.Е. |
|  |  |  |
| Задание получил студент гр.ИУК4-82Б |  | Сафронов Н.С. |

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc163771997)

[1. Микросервисные архитектуры 5](#_Toc163771998)

[2. Проблемы управления данными в микросервисных архитектурах 7](#_Toc163771999)

[2.1. Согласованность данных 7](#_Toc163772000)

[2.2. Дублирование данных 8](#_Toc163772001)

[2.3. Доступ к данным 8](#_Toc163772002)

[2.4. Хранение данных 9](#_Toc163772003)

[3. Механизмы управления данными в микросервисных архитектурах 10](#_Toc163772004)

[3.1. Отдельные и общие базы данных 10](#_Toc163772005)

[3.2. API для доступа к данным 11](#_Toc163772006)

[3.3. Событийно-ориентированное взаимодействие 12](#_Toc163772007)

[3.4. Шаблон проектирования Saga 14](#_Toc163772008)

[3.5. Шаблон проектирования CQRS 16](#_Toc163772009)

[ЗАКЛЮЧЕНИЕ 17](#_Toc163772010)

[СПИСОК ЛИТЕРАТУРЫ 18](#_Toc163772011)

# 

# ВВЕДЕНИЕ

**Актуальность данной темы** заключается в том, что микросервисные архитектуры становятся все более популярными для разработки программного обеспечения. Этот подход предоставляет множество преимуществ, таких как увеличение гибкости, масштабируемости и скорости разработки, но одновременно создает сложности в управлении данными. Управление данными в микросервисных архитектурах имеет свои собственные вызовы, включая обеспечение согласованности данных, управление схемами данных, обеспечение безопасности и конфиденциальности данных, а также обеспечение производительности и масштабируемости.

**Целью работы** является исследование существующих механизмов управления данными в микросервисных архитектурах и решаемых ими задач.

Для решения поставленной цели ставятся следующие **задачи**:

* + - * Изучить основные понятия, связанные с микросервисными архитектурами.
      * Выделить основные проблемы, возникающие при работе с данными в микросервисных архитектурах.
      * Рассмотреть механизмы управления данными, призванные решить эти проблемы.

# 1. Микросервисные архитектуры

Микросервисная архитектура представляет собой подход к разработке программного обеспечения, основанный на создании набора небольших, автономных и легко масштабируемых сервисов, каждый из которых отвечает за выполнение конкретной функциональности. Архитектура микросервисов характеризуется набором маленьких, независимых сервисов, которые слабо связаны между собой и взаимодействуют друг с другом с использованием API. Каждый сервис отвечает за выполнение определенной бизнес-функции и может быть разработан, развернут и масштабирован независимо. В отличие от традиционных монолитных архитектур, где весь функционал приложения сосредоточен в едином монолите, микросервисы являются отдельными компонентами, взаимодействующими между собой через сетевые запросы.

Основные характеристики микросервисов включают:

1. *Независимость сервисов*. Каждый микросервис может быть разработан, развернут и масштабирован независимо от других. Это позволяет командам разработки работать независимо друг от друга и быстро вносить изменения в свой сервис без необходимости изменений в других частях приложения.

2. *Автономность и самостоятельность*. Микросервисы являются автономными компонентами, которые могут функционировать независимо от состояния других сервисов. Это означает, что каждый сервис имеет свою собственную базу данных, свои собственные зависимости и свой собственный процесс развертывания.

3. *Гибкость и масштабируемость.* Микросервисы позволяют гибко масштабировать отдельные компоненты системы в зависимости от нагрузки. Это позволяет эффективно использовать ресурсы и обеспечивать высокую производительность приложения даже в условиях изменяющейся нагрузки.

4. *Легкость развертывания и обновления.* Благодаря автономности каждый микросервис может быть легко развернут и обновлен независимо от других. Это позволяет быстро внедрять новые функции, исправлять ошибки и улучшать производительность приложения без необходимости переразворачивания всего приложения целиком.

В архитектуре микросервисов данные распределены по нескольким сервисам, и каждый сервис может иметь свою базу данных или хранилище данных. Это распределение данных может создавать вызовы в управлении согласованностью данных, избыточностью данных, доступом к данным и их хранением.

Одной из основных проблем в управлении данными в архитектуре микросервисов является поддержание согласованности данных между сервисами [11]. Поскольку каждый сервис имеет свою собственную базу данных, важно обеспечить синхронизацию и согласованность данных в каждой базе данных. Кроме того, управление данными между сервисами может быть сложным, поскольку сервисы могут использовать разные модели данных или даже разные технологии баз данных. Более того, поскольку микросервисы могут быть независимо развернуты, они могут размещаться в разных местах, что дополнительно усложняет управление данными.

Для решения этих проблем в архитектуре микросервисов возникли различные шаблоны управления данными. Эти шаблоны разработаны для обеспечения согласованности данных, облегчения обмена данными между сервисами и упрощения управления данными между сервисами.

# Проблемы управления данными в микросервисных архитектурах

# 2.1. Согласованность данных

В случае использования отдельных баз данных каждый из сервисов управляет своей собственной базой данных. Отсутствие централизованной точки доступа к данным усложняет обеспечение их целостности, поскольку изменения в одном сервисе могут требовать соответствующих обновлений в других сервисах.

В отличие от традиционных монолитных приложений, где целостность данных часто обеспечивается за счет ACID-транзакций в пределах одной базы данных, в микросервисных архитектурах достижение аналогичного уровня транзакционной целостности представляет собой более сложную задачу. Это обусловлено физическим и логическим разделением баз данных, что усложняет синхронизацию и согласование данных между различными сервисами.

В распределенных системах широко используется репликация данных между сервисами для повышения производительности и обеспечения отказоустойчивости. Однако такой подход также вносит дополнительные трудности в поддержание целостности данных, особенно в контексте синхронизации и управления конфликтами при обновлении реплицированных данных. Сетевые вызовы между сервисами, осуществляемые через протоколы типа HTTP/HTTPS, gRPC или AMQP, могут подвергаться различным сетевым задержкам, зависящим от различных факторов, таких как физическое расстояние и качество сети.

Сетевые задержки могут существенно увеличить время ответа сервисов, особенно в случае операций, требующих последовательных взаимодействий между несколькими сервисами. Такие задержки также повышают вероятность сбоев и ошибок, что требует реализации механизмов повторных попыток и балансировки нагрузки.

Поддержание консистентности данных между сервисами в условиях сетевых задержек представляет собой сложную задачу. Обновление данных в одном сервисе может привести к работе других сервисов с устаревшими данными. Реализация распределенных транзакций также усложняется из-за увеличения времени ожидания подтверждений от участвующих сервисов.

# 2.2. Дублирование данных

Дублирование данных – ещё одна проблема управления данными в архитектуре микросервисов. В архитектуре микросервисов каждый сервис может иметь свою собственную базу данных, что может привести к дублированию данных между несколькими сервисами. Дублирование данных может привести к несогласованности, увеличению затрат на хранение и снижению производительности системы. Один из шаблонов, используемых для решения проблемы дублирования данных в архитектуре микросервисов, - это архитектура, основанная на событиях.

В архитектуре, основанной на событиях, события используются для распространения изменений данных между несколькими сервисами. Когда сервис обновляет свою базу данных, он публикует событие в брокер сообщений, указывая, что данные были изменены. Другие сервисы, заинтересованные в данных, подписываются на событие и обновляют свои базы данных соответственно. Такой подход гарантирует согласованность данных между всеми сервисами и уменьшает дублирование данных.

# 2.3. Доступ к данным

Доступ к данным является еще одним вызовом в управлении данными в архитектуре микросервисов. В архитектуре микросервисов каждый сервис имеет свою базу данных, что может затруднить доступ к данным между сервисами. Один из шаблонов, используемых для решения этой проблемы, - это шаблон шлюза API.

В шаблоне шлюза API используется единая точка входа для доступа ко всем сервисам в системе. API-шлюз предоставляет унифицированный интерфейс для доступа к данным из различных сервисов. Когда клиент отправляет запрос API-шлюзу, шлюз транслирует запрос в вызовы соответствующим сервисам и агрегирует ответы, чтобы предоставить единый ответ клиенту. Такой подход упрощает взаимодействие клиента с системой и облегчает управление доступом к данным между сервисами.

# 2.4. Хранение данных

Хранение данных представляет собой еще один вызов в управлении данными в архитектуре микросервисов. В архитектуре микросервисов каждый сервис имеет свою базу данных, что может привести к увеличению затрат на хранение и снижению производительности системы. Один из шаблонов, используемых для решения этой проблемы, - это шаблон базы данных для каждого сервиса.

В шаблоне базы данных для каждого сервиса каждый сервис имеет свою базу данных, которая используется для хранения данных, специфичных для этого сервиса. Такой подход обеспечивает изоляцию между сервисами и снижает риск того, что данные будут прочитаны несколькими пользователями.

# Механизмы управления данными в микросервисных архитектурах

# Отдельные и общие базы данных

В контексте микросервисных архитектур существует два основных подхода к управлению данными: использование отдельных баз данных для каждого сервиса и использование общей базы данных для нескольких сервисов [9].

В случае отдельных баз данных каждый микросервис имеет свою собственную отдельную базу данных, к которой он имеет эксклюзивный доступ. Каждый сервис управляет только своими данными, что обеспечивает независимость и изоляцию. Это позволяет сервисам быть более автономными и уменьшает вероятность конфликтов данных между сервисами. Кроме того, этот подход обеспечивает лучшую масштабируемость, так как каждая база данных может быть масштабирована независимо от других.

Однако использование отдельных баз данных может привести к дополнительным затратам на управление множеством баз данных и обмен данными между сервисами. Также существует риск дублирования данных и возможных несоответствий в данных между разными сервисами.

В случае общих баз данных несколько микросервисов используют одну общую базу данных для доступа к данным. Это может быть полезно в случае, когда несколько сервисов имеют общие данные или требуют тесного взаимодействия между собой. Общая база данных может также обеспечить единую точку управления данными и облегчить синхронизацию и консистентность данных.

Однако использование общей базы данных может привести к проблемам согласованности данных и взаимодействия между сервисами. Изменения в одном сервисе могут негативно сказаться на других сервисах, что усложняет поддержку и развертывание системы.

В зависимости от конкретных требований и особенностей проекта, выбор между отдельными и общими базами данных должен быть внимательно взвешен и основан на принципах согласованности, доступности и разделения данных в микросервисной архитектуре.

# API для доступа к данным

В шаблоне композиции API несколько микросервисов объединяются для предоставления единого API клиенту [9]. Этот шаблон обеспечивает высокую гибкость в управлении данными, поскольку каждый микросервис может управлять своими данными независимо. Однако управление зависимостями между микросервисами может представлять определенные трудности. Шаблон шлюза API - это шаблон управления данными, который часто используется в сочетании с шаблоном базы данных для каждого сервиса. В этом шаблоне используется единый API-шлюз для предоставления единообразного интерфейса для всех сервисов, позволяя клиентам получать доступ к данным из нескольких сервисов через единый API.

Этот шаблон обеспечивает несколько преимуществ, таких как повышенная безопасность, упрощенная разработка клиентов и улучшенная производительность. Безопасность повышается, потому что API-шлюз может использоваться для применения правил аутентификации и авторизации, гарантируя, что доступ к данным из различных сервисов имеют только авторизованные клиенты. Упрощенная разработка клиентов достигается, потому что клиентам нужно взаимодействовать только с одним API, а не с несколькими API для каждого сервиса. Улучшение производительности достигается, потому что API-шлюз может использоваться для кэширования часто запрашиваемых данных, сокращая количество запросов, которые необходимо делать к различным сервисам.

Однако у шаблона шлюза API есть и некоторые недостатки. Например, он может создавать единую точку отказа, так как все запросы должны проходить через API-шлюз. Кроме того, API-шлюз может стать узким местом в производительности, если он не был правильно спроектирован и реализован.

# Событийно-ориентированное взаимодействие

В микросервисных архитектурах одним из ключевых механизмов взаимодействия между сервисами является событийно-ориентированная архитектура (Event-Driven Architecture, EDA) [11]. Этот подход основан на обмене сообщениями, или событиями, между компонентами системы, что позволяет создать легкую, гибкую и масштабируемую архитектуру.

Основные компоненты событийно-ориентированной архитектуры:

*Издатели* представляют собой компоненты системы, которые генерируют и отправляют события. Они отвечают за определение и форматирование событий, а также за их распространение в системе. В контексте микросервисов, каждый сервис может выступать в роли издателя, отправляя события о своих действиях или изменениях состояния.

*Подписчики* — это компоненты системы, которые подписываются на определенные типы событий и реагируют на них, выполняя соответствующие действия. Подписчики могут быть как самими микросервисами, так и другими компонентами системы или внешними приложениями. Они могут реагировать на события сразу или сохранять их для последующей обработки.

*Брокеры сообщений* — это посредники, которые обеспечивают маршрутизацию и доставку событий от издателя к подписчикам. Они предоставляют механизмы для хранения, маршрутизации и обработки событий, а также гарантируют надежную доставку сообщений. Популярные брокеры сообщений включают Apache Kafka, RabbitMQ, Amazon SQS и другие.

Событийно-ориентированная архитектура обладает рядом преимуществ, которые делают ее привлекательным выбором для микросервисных систем:

*Гибкость и расширяемость*. EDA позволяет создавать гибкие и расширяемые системы, где добавление новых сервисов или изменение существующих не требует больших изменений в архитектуре.

*Реактивность и отзывчивость*. Событийно-ориентированная архитектура позволяет создавать реактивные системы, способные моментально реагировать на изменения и события в окружающей среде.

*Легкая интеграция.* EDA упрощает интеграцию различных сервисов и компонентов системы, позволяя им обмениваться сообщениями независимо от их технологической реализации.

*Обработка асинхронных операций*. Событийно-ориентированная архитектура хорошо подходит для обработки асинхронных операций, где различные части системы могут работать независимо друг от друга.

*Обеспечение надежности и отказоустойчивости*. Использование брокеров сообщений позволяет обеспечить надежную доставку сообщений и управление их обработкой, что повышает отказоустойчивость и надежность системы.

**Event Sourcing**

Event Sourcing представляет собой мощный подход к сохранению истории изменений данных и обеспечению целостности данных в распределенных системах [11].

Event Sourcing предлагает хранить все изменения состояния системы в виде событий, которые записываются в хронологическом порядке. Эти события представляют собой факты, которые неизменно описывают, что произошло в системе. В контексте микросервисной архитектуры каждый сервис может вести свой собственный журнал событий, содержащий историю изменений, которые произошли в его области ответственности.

Запись изменений в виде событий позволяет системе быть устойчивой к сбоям и отказам. В случае сбоя или ошибки можно восстановить состояние системы путем повторного применения всех событий из журнала. Это обеспечивает надежное восстановление данных и истории изменений.

Event Sourcing обеспечивает надежное хранение истории изменений, позволяет эффективно восстанавливать состояние системы и поддерживает асинхронное взаимодействие между сервисами.

# Шаблон проектирования Saga

Паттерн Saga — это паттерн проектирования, который используется для управления транзакциями в распределенных системах, включая микросервисные архитектуры. Этот паттерн предлагает способ обработки долгих и сложных транзакций без блокировки и ожидания ответа от всех участвующих сервисов. Вместо этого он использует набор небольших транзакций, известных как шаги, каждый из которых либо успешно завершается, либо отменяется.

Основные принципы паттерна Saga включают в себя:

*Атомарность операций*. Каждый шаг саги должен быть атомарным - либо полностью выполнен, либо полностью отменен. Это означает, что, если один шаг завершается успешно, все последующие шаги также должны быть успешно выполнены. Если же возникает ошибка, все предыдущие шаги должны быть отменены.

*Компенсирующие действия*. Для отмены или компенсации успешно выполненных шагов используются компенсирующие действия. Эти действия должны быть обратными к оригинальным шагам и должны восстановить систему в состояние, аналогичное состоянию перед выполнением шага.

*Долговечность.* Сага должна быть долговечной, то есть информация о ее состоянии должна сохраняться даже в случае сбоев. Это позволяет восстанавливать её выполнение.

Оркеструемые саги и хореографируемые саги — это два основных подхода к реализации паттерна Saga в распределенных системах. Они предлагают разные способы координации выполнения транзакций между сервисами.

*Оркеструемые саги* предполагают наличие центрального компонента, называемого оркестратором, который координирует и управляет выполнением всех шагов транзакции. Оркестратор определяет порядок выполнения шагов, запускает их и обрабатывает ошибки, при необходимости вызывая компенсирующие действия.

Основные характеристики оркеструемых саг:

*Централизованное управление*. Оркестратор контролирует и координирует все шаги саги.

*Простота реализации*. Такой подход обеспечивает более простую реализацию, так как логика управления транзакцией сосредоточена в одном компоненте.

*Удобство отслеживания*. Оркестратор может предоставлять информацию о состоянии транзакции и ее прогрессе.

Пример оркеструемой саги можно увидеть в системе оформления заказа в интернет-магазине, где оркестратор контролирует шаги резервирования товаров, проверки оплаты, организации доставки и уведомления клиента.

В *хореографируемых сагах* отсутствует централизованный оркестратор, и каждый сервис самостоятельно управляет своими действиями и реакциями на изменения состояния системы. Каждый сервис знает о своих собственных действиях и о том, как реагировать на сообщения и события от других сервисов.

Основные характеристики хореографируемых саг:

*Децентрализованное управление*. Управление транзакцией распределено между всеми участвующими сервисами.

*Более сложная реализация*. Разработка хореографируемых саг требует более тщательного проектирования и согласования между сервисами.

*Более сложное отслеживание*. Отслеживание состояния транзакции может быть сложнее без централизованного оркестратора.

Примером хореографируемой саги может служить процесс подтверждения заказа в системе электронной коммерции, где каждый сервис (например, сервис резервирования товаров и сервис оплаты) самостоятельно обрабатывает и координирует свои действия в ответ на сообщения от других сервисов.

Оба подхода имеют свои преимущества и недостатки, и выбор между ними зависит от конкретных требований и контекста системы.

# Шаблон проектирования CQRS

В паттерне CQRS (Command Query Responsibility Segregation) один микросервис отвечает за обработку команд (операций записи), в то время как другой микросервис отвечает за обработку запросов (операций чтения). Этот шаблон разделяет задачи чтения и записи данных, что может привести к улучшению масштабируемости и производительности. Паттерн Command Query Responsibility Segregation (CQRS) представляет собой способ разделения операций записи и операций чтения в системе. В архитектуре микросервисов каждый сервис отвечает за обработку определенного набора операций. Шаблон CQRS предоставляет способ разделить операции чтения от операций записи.

В паттерне CQRS каждый сервис имеет две отдельные модели: модель записи и модель чтения. Модель записи отвечает за обработку операций записи, в то время как модель чтения отвечает за обработку операций чтения. Такой подход предоставляет несколько преимуществ, таких как лучшая масштабируемость и производительность, поскольку операции чтения и записи могут быть оптимизированы отдельно.

# ЗАКЛЮЧЕНИЕ

Задачи, поставленные в ходе выполнения научно-исследовательской работы, были выполнены в полном объёме. Были рассмотрены ключевые аспекты управления данными в микросервисных архитектурах. Основное внимание уделено проблемам, возникающим в контексте распределенного хранения данных в таких системах.

В ходе исследования были выявлены основные вызовы, связанные с управлением данными в микросервисных архитектурах, включая проблемы с целостностью данных, избыточностью данных, доступом к данным и хранением данных.

Для решения этих проблем были рассмотрены различные подходы и паттерны управления данными в микросервисных архитектурах. Были описаны такие паттерны, как «База данных на сервис», «API для доступа к данным», «Событийно-ориентированное взаимодействие» и другие.

Кроме того, были проанализированы преимущества и недостатки каждого из этих подходов, а также предложены рекомендации по выбору наиболее подходящего подхода в конкретных сценариях.

В заключении работы были сделаны выводы о том, что управление данными в микросервисных архитектурах представляет собой сложную и многогранную проблему, требующую внимательного анализа и выбора оптимальных решений для каждой конкретной ситуации.

# СПИСОК ЛИТЕРАТУРЫ

1. Аверченков, В.И. Основы научного творчества [Электронный ресурс]: учеб. пособие/ В.И. Аверченков, Ю.А. Малахов. — Брянск: Брянский государственный технический университет, 2012. — 156 c.— Режим доступа: http://www.iprbookshop.ru/7004.

2. Астанина, С.Ю. Научно-исследовательская работа студентов (современные требования, проблемы и их решения) [Электронный ресурс]: монография/ С.Ю. Астанина, Н.В. Шестак, Е.В. Чмыхова. — М.: Современная гуманитарная академия, 2012. — 156 c. — Режим доступа: <http://www.iprbookshop.ru/16934>.

3. Губарев, В.В. Квалификационные исследовательские работы [Электронный ресурс]: учеб. пособие/ В.В. Губарев, О.В. Казанская. — Новосибирск: Новосибирский государственный технический университет, 2014. — 80 c.— Режим доступа: <http://www.iprbookshop.ru/47691>.

4. Половинкин, А.И. Основы инженерного творчества. [Электронный ресурс]: учебное пособие / А.И. Половинкин. — СПб : Лань, 2019. — 364 с.—

Режим доступа: <https://e.lanbook.com/book/123469>.

5. Рыжков, И.Б. Основы научных исследований и изобретательства. [Электронный ресурс]: учеб. пособие / И.Б. Рыжков. — СПб.: Лань, 2013. — 224 с. — Режим доступа: http://e.lanbook.com/book/30202.

6. Новиков, Ю.Н. Подготовка и защита бакалаврской работы, магистерской диссертации, дипломного проекта [Электронный ресурс] : учебное пособие / Ю.Н. Новиков. — 4-е изд., стер. — Санкт-Петербург : Лань, 2019. — 34 с. — URL: <https://e.lanbook.com/book/122187>.

7. Рекомендации по написанию и оформлению курсовой работы, выпускной квалификационной работы и магистерской диссертации [Электронный ресурс]: учебно-методическое пособие/ Е.В. Зудина [и др.]. — Волгоград: Волгоградский государственный социально-педагогический университет, 2016. — 57 c. — Режим доступа: http://www.iprbookshop.ru/57785.

8. Беллермар, А. Создание событийно-управляемых микросервисов. – СПб.: БХВ-Петербург, 2022 – 320 с.

9. Ричардсон, К. Микросервисы. Паттерны разработки и рефакторинга – СПб.: Питер, 2019. – 544 с.

10. Кочер, П.С. Микросервисы и контейнеры Docker. – М.: ДМК Пресс, 2018 – 242 с.

11. Ньюман, С. От монолита к микросервисам. – СПб.: БХВ-Петербург, 2021 – 266 с.