МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

СРАВНЕНИЕ МОНОЛИТНОЙ И МИКРОСЕРВИСНОЙ АРХИТЕКТУР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Елесиной Татьяны Викторовны

Научный руководитель
зав. каф. тех. прогр., к. ф.-м. н. _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Большинство современных приложений, которые должны соответствовать требованиям масштабируемости и высокой нагрузки, распределены или, по крайней мере, используют некоторые распределенные компоненты. Поэтому одним из наиболее актуальных архитектурных подходов к построению системы являются микросервисы.

Также существует и другой, исторически наиболее ранний, архитектурный подход — построение программного обеспечения как единого компонента, а не как набора нескольких распределенных приложений. Архитектура такого программного обеспечения называется монолитной.

Актуальность данной темы состоит во все большем внимании к микросервисам и массовому переходу крупных и высоконагруженных сервисов от монолитной к микросервисной архитектуре.

Новизна исследования состоит в исправлении недостатков монолитного подхода в серверных приложениях за счет использования микросервисной архитектуры.

В данной дипломной работе будут рассмотрены принципы этих архитектурных подходов, различные средства для обеспечения работы микросервисов, а также будет проведено сравнение архитектурных принципов. Практическая часть дипломной работы посвящена разработке программного обеспечения с монолитной и микросервисной архитектурой на фреймворке Spring Boot языка Java.

Целью дипломной работы является реализация и сравнение двух систем — с монолитной и с микросервисной архитектурой.

Исходя из цели работы, были поставлены следующие задачи:

- изучить:
  - цели и методы создания распределенных систем;
  - типы распределенных систем, основные проблемы и подходы к их решению;
  - принципы проектирования микросервисов;
- применять на практике:
  - шаблоны проектирования микросервисов;
  - методы создания микросервисов с использованием программного обеспечения Spring Cloud.

1 Теоретическая часть

1.1 Архитектура программного обеспечения

Архитектура программного обеспечения относится к фундаментальной структуре программной системы и основой создания таких структур и систем. Такая структура содержит программные элементы, взаимосвязи между ними, а также свойства как элементов, так и взаимосвязей [1].

1.2 Типы архитектур программного обеспечения

Существуют несколько типов архитектуры программного обеспечения, но среди них можно выделить два наиболее распространенных типа: монолитная и микросервисная.

1.2.1 Монолитная архитектура

Большинство приложений, созданных для удовлетворения потребностей бизнеса, спроектированы таким образом, что одно приложение может выполнять множество различных функций, например, учет товаров на складе, добавление новых пользователей, проверка данных для авторизации и так далее. Такое приложение называется монолитным.

1.2.2 Микросервисная архитектура

В основе микросервисной архитектуры лежит принцип разработки системы как набора небольших независимых сервисов, которые разрабатываются и развертываются независимо [2].

1.3 Принципы микросервисной архитектуры

Можно создавать свое программное приложение с нуля, используя архитектуру микросервисов, или преобразовывать существующие приложения в микросервисы [3]. В любом случае, очень важно правильно определить размер, объем и возможности микросервисов [4].

1.4 Обмен сообщениями

В монолитных приложениях методы различных компонентов вызываются при помощи стандартных средств языка программирования.

В микросервисной архитектуре же одним из обязательных аспектов является простой и легковесный механизм передачи сообщений.

Далее будут приведены два типа обмена сообщениями.

1.4.1 Синхронная передача сообщений

Синхронной передачей сообщений называется ситуация, когда один сервис отправляет запрос другому сервису и ждет от него ответа, при этом не выполняя других действий.

1.4.2 Асинхронная передача сообщений

Для некоторых ситуаций выгоднее использовать асинхронный обмен сообщениями — когда один сервис отправляет сообщение другому, но не ждет его ответа, а продолжает работу с каким-либо другими запросами [5].

Кроме типа обмена сообщениями следует обратить внимание на две не менее важных составляющих этого обмена — формат сообщений и контракты между сервисами.

1.4.3 Формат сообщений

В большинстве приложений на основе микросервисов используются простые текстовые форматы сообщений, такие как JSON и XML, поверх передаваемые по протоколу HTTP через API [6].

1.4.4 Контракты между сервисами

Для каждого сервиса должен быть создан контракт, определяющий все детали, которые необходимы для обмена сообщения с ним.

Например, в данном случае для составления контрактов можно использовать один из стандартных языков определения REST API, например, Swagger [7].

1.5 Взаимодействие между микросервисами

Далее будут представлены несколько паттернов, которые используются для реализации связей между приложениями в микросервисной архитектуре [8].

1.5.1 Вызов микросервисов напрямую

При таком подходе вся логика маршрутизации сообщений находится на каждой конечной точке микросервиса, и сервисы могут взаимодействовать напрямую друг с другом без посредников.

1.5.2 Использование API-шлюза

API-шлюз представляет собой облегченную центральную шину обмена сообщениями, которая может обеспечить уровень абстракции для микросервисов и которую можно использовать для реализации различных нефункциональных возможностей [2].

1.5.3 Использование брокеров сообщений

Микросервисы могут быть интегрированы в сценарии асинхронного обмена сообщениями, такие как односторонние запросы и сообщения с публикацией и подпиской с использованием очередей или тем.

1.6 Управление данными

В монолитной архитектуре приложение хранит данные в единой централизованной базе данных для реализации различных возможностей приложения.

В архитектуре микросервисов функциональные возможности распределены по нескольким микросервисам, и если используется одна и та же централизованную базу данных, то микросервисы больше не будут независимы друг от друга [9].

1.7 Реестр и обнаружение микросервисов

В микросервисной архитектуре количество микросервисов, которыми нужно управлять, довольно велико. А также их местоположение динамически меняется благодаря быстрой и гибкой разработке и развертыванию микросервисов. Поэтому необходимо знать актуальное местоположение микросервиса во время выполнения приложения. Решением этой проблемы является использование реестра служб [10].

Существует два типа механизмов обнаружения служб: обнаружение на стороне клиента и обнаружение на стороне сервера.

1.7.1 Обнаружение на стороне клиента

При таком подходе клиент или API-шлюз получает местоположение экземпляра службы путем запроса к реестру служб. Клиент либо API-шлюз должен реализовать логику обнаружения службы, вызвав компонент реестра служб.

1.7.2 Обнаружение на стороне сервера

При таком подходе клиенты и API-шлюзы отправляют запрос компоненту (например, балансировщику нагрузки), который имеет постоянное местоположение. Этот компонент вызывает реестр служб и определяет текущее местоположение микросервиса.

1.8 Развертывание приложений

Когда дело доходит до архитектуры микросервисов, развертывание микросервисов играет решающую роль.

Docker — движок с открытым исходным кодом, который позволяет разработчикам и системным администраторам развертывать самодостаточные контейнеры приложений в средах Linux [11].

1.9 Устойчивость системы

Микросервисная архитектура представляет собой рассредоточенный набор сервисов и по сравнению с монолитной архитектурой увеличивает вероятность сбоев на каждом этапе обработки запроса. Поскольку службы могут выйти из строя в любое время, важно иметь возможность быстро обнаруживать сбои.

Существует несколько часто используемых шаблонов обработки ошибок в контексте микросервисов [12].

1.9.1 Circuit Breaker

Когда выполняется внешний вызов микросервиса, то настраивается компонент мониторинга сбоев при каждом вызове, и когда сбои достигают определенного порога, этот компонент останавливает любые дальнейшие вызовы службы (размыкает цепь).

1.9.2 Таймаут

Таймаута — это механизм, который позволяет перестать ждать ответа от микросервиса по истечении какого-либо периода времени. Можно настроить интервал времени, который необходимо подождать прежде, чем прерывать выполнение запроса.

2 Практическая часть

2.1 Описание разработанного программного обеспечения

В рамках данной дипломной работы было разработано два веб-приложения, объединенные одной идеей и использующие одинаковый стек технологий, но имеющие разную архитектуру — одно из них имеет монолитную архитектуру, другое — микросервисную.

Используемые технологии: язык программирования Java [13], фреймворк Spring Boot, Spring Cloud, продукты Netflix OSS [14], Apache Kafka, Docker.

Идея приложения — сервис, предоставляющий пользователям возможность публиковать текстовые записи, оставлять комментарии под записям, просматривать записи и комментарии других пользователей.

2.2 Описание API контрактов

На основе написанного кода с помощью фреймворка Spring были сгенерированы API контракты на языке Swagger. Они позволяют в удобной форме ознакомиться с тем, какие данные принимаются на вход и отдаются на выход. Также показано какие могут быть коды ошибок и что могло эти ошибки вызвать.

2.3 Описание монолитной архитектуры

Приложение с монолитной архитектурой представляет собой единую программу на языке Java с использованием фреймворка Spring Boot и пары вспомогательных библиотек для обеспечения соединения с базой данных, работы с токенами [15].

В качестве СУБД используется H2 — открытая кроссплатформенная система управления базами данных [16].

Таблицы в базе генерируются из Java кода, помеченного специальными аннотациями [17].

2.4 Описание микросервисной архитектуры

2.4.1 Компоненты архитектуры

Система с микросервисной архитектурой представляет собой несколько приложений (микросервисов) на языке Java с использованием фреймворка Spring Boot, а также используется развернутый в Docker контейнере экземпляр Kafka [18].

В качестве СУБД используется H2. Для каждого микросервиса была создана своя база данных.

2.4.2 Архитектура запросов к компонентам

`UserService` отвечает за регистрацию пользователей, выдачу токенов для логина, показ данных о достижениях пользователей.

`PostService` позволяет увидеть последние опубликованные записи, опубликовать запись, удалить запись, оставить комментарий к записи, удалить свой комментарий, пожаловаться на комментарий другого пользователя, посмотреть комментарии к записи.

`UserFeedService` позволяет подписаться на другого пользователя, отписаться от пользователя и видеть ленту, составленную из записей пользователей, на которых подписан пользователь.

2.4.3 Взаимодействие с Kafka

В Kafka были созданы три темы — `userEvents`, `postEvents` и `commentEvents`. В каждую из них записываются события, связанные с пользователями, записями и комментариями соответственно.

2.4.4 Взаимодействие компонентов друг с другом

Компоненты взаимодействуют друг с другом при следующих запросах [19]:

1. Когда пользователь хочет просмотреть новостную ленту.
2. Когда пользователь хочет посмотреть последние записи или список комментариев.

2.5 Обеспечение безопасности приложения

2.5.1 Общие идеи между монолитной и микросервисной архитектурами

В обеих версиях приложения для обеспечения безопасности используются JWT токены [20].

Пользователь регистрируется, входит в систему и получает токен.

Токены генерируются приложением, когда пользователь хочет войти в систему, а затем отдаются пользователю.

Полученный токен пользователь использует для каждого запроса.

Пользователь может быть заблокирован, если он оставит 3 или более оскорбительных поста или комментариев. Если пользователь был заблокирован, он может только читать (сообщения, комментарии, достижения). Информация о статусе пользователя шифруется в токене.

2.5.2 Безопасность в монолитной приложении

В монолитной версии приложения используется всего два главных компонента из Spring Boot Security — это `SecurityConfig`, включающий и конфигурирующий безопасность приложения, и `JwtFilter`, отвечающий за более подробную настройку.

2.5.3 Безопасность в микросервисном приложении

В микросервисной архитектуре приложения используется более сложная система безопасности по сравнению с монолитной.

API-шлюз отклоняет запросы без токенов при помощи фильтров, если эти запросы сделаны по путям, которые должны быть защищены от неавторизованного пользователя.

Токены генерирует и отдает пользователю только `UserService`. При этом валидация токенов происходит в каждом микросервисе.

Используются API-ключи — токены, созданные специально для каждого сервиса. Ключи используются только для взаимодействия между сервисами.

2.6 Подробное описание работы приложения при некоторых запросах пользователя в микросервисной системе

2.6.1 Регистрация пользователя

Данный пользовательский сценарий предназначен для регистрации пользователя в системе.

2.6.2 Публикация записи

Данный пользовательский сценарий предназначен для публикации записи от лица пользователя в системе.

2.6.3 Получение пользовательской новостной ленты

Данный пользовательский сценарий предназначен для просмотра подписчиком новостной ленты, состоящей из записей, которые опубликовали пользователи.

2.7 Разработка консольного приложения

В качестве одного из клиентов разработанной системы было создано приложение, позволяющее взаимодействовать с системой через консоль (терминал) с помощью текстовых команд.

Данное приложение разрабатывалось на языке Java и фреймворке Spring Boot с использованием зависимости Spring Shell.

2.8 Разработка сайта

В качестве альтернативного клиента разработанной системы был создан сайт. Для разработки сайта использовался язык Java и шаблонизатор Mustache, позволяющий создавать динамические HTML-страницы без использования JavaScript.

2.9 Сравнение монолитной и микросервисной архитектур

Если приложение планируется не очень большое, то лучше сделать его монолитным. Здесь нет необходимости внедрять микросервисы. Монолитное приложение будет намного проще создавать, модифицировать, развертывать и тестировать.

Микросервисная архитектура более выгодна для сложных и развивающихся приложений. Микросервисы идеальны, когда речь идет о платформах, охватывающих множество пользователей и рабочих процессов. Но без предшествующего опыта работы с микросервисами сделать надежную систему очень сложно, так как необходимо использовать множество технологий.

ЗАКЛЮЧЕНИЕ

В данной дипломной работе были рассмотрены различные подходы к построению веб-приложений. Были рассмотрены теоретические аспекты, необходимые для реализации приложений, детали реализованного проекта, а также проведено сравнение двух систем — с монолитной и с микросервисной архитектурой, что являлось целью выполнения данной работы.

В результате выполнения дипломной работы были решены следующие задачи:

- изучены цели и методы проектирования распределенных систем;
- изучены типы распределенных систем, основные проблемы и подходы к их решению;
- изучены принципы проектирования приложений микросервисов, типичные проблемы и решения;
- применены на практике шаблоны проектирования микросервисов;
- применены на практике методы создания микросервисов с использованием программного обеспечения Spring Cloud.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Мартин, Р.* Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. — Санкт-Петербург: Питер, 2018.

2 *Ньюмен, С.* Создание микросервисов / С. Ньюмен. — Санкт-Петербург: Питер, 2016.

3 *Карнелл, Д.* Микросервисы Spring в действии / Д. Карнелл, И. Санчес. — Москва: ДМК Пресс, 2021.

4 Руководство по проектированию API — Best practices for cloud applications [Электронный ресурс]. — URL: https://docs.microsoft.com/ru-ru/azure/architecture/best-practices/api-design (Дата обращения 09.03.2022). Загл. с экр. Яз. рус.

5 Apache Kafka [Электронный ресурс]. — URL: https://kafka.apache.org/intro (Дата обращения 17.03.2022). Загл. с экр. Яз. англ.

6 HTTP | MDN [Электронный ресурс]. — URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (Дата обращения 09.03.2022). Загл. с экр. Яз. англ.

7 About Swagger Specification | Documentation | Swagger [Электронный ресурс]. — URL: https://swagger.io/docs/specification/about/ (Дата обращения 09.03.2022). Загл. с экр. Яз. англ.

8 A pattern language for microservices [Электронный ресурс]. — URL: https://microservices.io/patterns (Дата обращения 22.03.2022). Загл. с экр. Яз. англ.

9 *Кинг, Г.* Java Persistence API и Hibernate / Г. Кинг, Г. Грегори, К. Бауэр. — Санкт-Петербург: Питер, 2019.

10 Spring | Cloud [Электронный ресурс]. — URL: https://spring.io/cloud (Дата обращения 12.04.2022). Загл. с экр. Яз. англ.

11 Docker Documentation | Docker Documentation [Электронный ресурс]. — URL: https://docs.docker.com (Дата обращения 16.03.2022). Загл. с экр. Яз. англ.

12 *Garsia, M.* Learn Microservices with Spring Boot: A Practical Approach to RESTful Services Using an Event-Driven Architecture, Cloud-Native Patterns, and Containerization / M. Garsia. — New York: Apress, 2020.

13 Язык программирования Java [Электронный ресурс]. — URL: https://metanit.com/java/tutorial (Дата обращения 15.03.2022). Загл. с экр. Яз. рус.

14 Spring Cloud Netflix [Электронный ресурс]. — URL: https://cloud.spring.io/spring-cloud-netflix/reference/html/ (Дата обращения 04.04.2022). Загл. с экр. Яз. англ.

15 Inversion of Control and Dependency Injection with Spring | Baeldung [Электронный ресурс]. — URL: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring (Дата обращения 14.03.2022). Загл. с экр. Яз. англ.

16 Tutorial [Электронный ресурс]. — URL: https://www.h2database.com/html/tutorial.html (Дата обращения 02.04.2022). Загл. с экр. Яз. англ.

17 Hibernate. Everything data. — Hibernate [Электронный ресурс]. — URL: https://hibernate.org (Дата обращения 15.03.2022). Загл. с экр. Яз. англ.

18 Spring | Microservices [Электронный ресурс]. — URL: https://spring.io/microservices (Дата обращения 13.04.2022). Загл. с экр. Яз. англ.

19 *Динеш, Р.* Spring. Все паттерны проектирования / Р. Динеш. — Санкт-Петербург: Питер, 2019.

20 *Фелипе, Г.* Spring Boot 2: лучшие практики для профессионалов / Г. Фелипе. — Санкт-Петербург: Питер, 2020.