Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: РиАТ

ОТЧЕТ

по выполненной работе

Выполнили

студенты: гр. 151003 Барановский Р.А.,

Матошко И.В.

Проверил: Новицкая Н.В.

Минск 2023

Архитектурное проектирование

|  |  |
| --- | --- |
| Микросервисы | |
| AuthService | Отвечает за аутентификацию(JWT) и авторизацию(проверку ролей) UR в системе. Выдает и валидирует refreshToken и accessToken |
| OrderService | Отвечает за действия, связанные с заказами UR(просмотр, оформление). Редактирование и удаление, если пользователь является AR |
| UserService | Отвечает за действия, связанные с информацией о UR(данные, указанные при регистрации, статистика о пользователе и т.д.), позволяет редактировать эту информацию |
| ProductService | Отвечает за действия, связанные с продуктами, представленными в онлайн-сервисе. В эти действия входят добавление, удаление продуктов из корзины, просмотр продукта. Добавление, удаление и редактирование продукта, если UR имеет роль AR |
| CategoryService | Отвечает за действия, связанные с категориями, представленными в онлайн-сервисе. В эти действия входит просмотр категории, а также ее продуктов. Добавление, удаление и редактирование, если UR имеет роль AR |
| SearchService | Отвечает за сортировку, поиск и фильтрацию продуктов в соответствии с критериями, заданными UR |
| RegistrationService | Отвечает за регистрацию UR в системе |

Для эффективного взаимодействия микросервисов друг с другом будем использовать брокер сообщений вместо прямого взаимодействия. В качестве брокера сообщений будет использоваться Apache Kafka. Выбор в пользу данного брокера сообщений был сделан в силу следующих преимуществ: Кластер Kafka обладает высокой масштабируемостью и отказоустойчивостью: при поломке одного из узлов, другие узлы берут на себя его работу, обеспечивая непрерывность работы без потери данных, а также события можно прочитать столько раз, сколько необходимо. В этом отличие Kafka от традиционных систем обмена сообщениями: после чтения события не удаляются. Можно настроить, как долго Kafka хранит события.

Выбор технологий

Языком разработки будет выступать Java. В качестве основы для разработки будем использовать Spring Boot для упрощения конфигурации. В качестве СУБД используем MySql. Для обеспечения взаимодействия с СУБД будем использовать Spring Data и JPA/Hibernate. Для обеспечения безопасности микросервисов и осуществления механизмов аутентификации и авторизации используем Spring Security(JWT authentication). Для управления миграциями БД используем Flyway. В целом придерживаемся REST-архитектуры. Для удобства запустим kafka в докере. Для этого создадим файл docker-compose.yaml, который будет содержать в себе параметры запуска Kafka, Kafdrop и Zookeeper. В настройках конфигурации создадим топик для каждого микросервиса(пример: OrderService – order, UserService – user и т.д.). Особенностью Apache Kafka является то, что в отличие от традиционных систем обмена сообщениями, события после прочтения не удаляются. Установим время жизни события – 1 день.

Разработка микросервисов

Для обеспечения отказоустойчивости микросервисов будем применять следующие меры(заранее обозначим, что в качестве балансировщика нагрузки применяется kubernetes):

1. Каждый экземпляр микросервиса будет отправлять балансировщику нагрузки свое состояние с определенной периодичностью, чтобы балансировщик нагрузки мог исключать из маршрутизации нерабочие экземпляры сервисов и направлять трафик на рабочие экземпляры.
2. В дополнение к предыдущему пункту, стоит реализовать с помощью сторонней службы процесс мониторинга состояния экземпляров микросервисов и их перезапуска, в случае, если экземпляр слишком долго находиться в нерабочем состоянии.
3. Для предотвращения перегрузки сервиса и его отказа стоит реализовать ограничение частоты запросов. В нашем случае этот вариант уместен, так как нет необходимости в событиях в режиме реального времени.
4. Для повышения отказоустойчивости и сохранения работоспособности критически важных функций сервиса в случае непредвиденных ситуаций стоит реализовать паттерн «Bulkheads», суть состоит в разделении ресурсов сервиса на несколько независымых блоков. В нашем случае это будет реализовываться путем создания двух пулов подключений к базе данных: один для критических задач, второй для остальных. Таким образом, полностью занятый один из пулов не приведет к полной остановке работоспособности сервиса и позволит ему продолжить работу в ограниченном режиме(обработка только критически важных запросов).
5. Для предотвращения зависания операций ограничим время выполнения запроса. Для корректной реализации будем использовать паттерн «Предохранитель». Предохранитель защищает ресурсы сервиса (соединения, дескрипторы), не позволяя их использовать в случае отказа вызываемого сервиса. Вместе с тем, вызываемый сервис быстрее восстановится после отказа, так как на него не поступают запросы от других сервисов.Предохранитель открывается, когда зафиксирует некоторое количество ошибок за короткий промежуток времени. Все последующие запросы через него будут возвращать ошибку сразу, без отправки во внешний сервис. Предохранитель сохраняет такое состояние в течение заданного промежутка времени, после чего закрывается и снова пропускает все запросы во внешний сервис.

Благодаря используемой архитектуре и технологиям достигается хорошая масштабируемость каждого микросервиса в частности. В свою очередь возможность создания нескольких экземпляров каждого микросервиса обеспечивает хорошую масштабируемость системы в целом.

Тестирование

Для модульного тестирования необходимо использовать unit-тесты, что обусловлено выбором технологий java. Данный вид тестирования необходимо использовать для проверки каждого модуля системы по отдельности. Для проведения интеграционного тестирования предлагается проверить работоспособность выполнения запросов к БД. Тестирование будет являться интеграционным, так как для тестирования методов с запросами к БД необходимо поднимать отдельную БД в оперативной памяти, подменяющую основную. Будут созданы таблицы с тестовыми данными, затем мы проверяем корректность отработки методов класса-репозитория. В качесте БД в оперативной памяти необходимо использовать MariaDB, так как она максимально совместима с MySQL и у скриптов MySQL не будет конфликтов с диалектом MariaDB. Так как разрабатываемое приложение относительно небольшое, следует обойтись ручным системным тестированием. В случае увеличения размера приложения, следует перейти к автоматическому тестированию. Системное тестирование следует проводить в последнюю очередь, в случае обнаружения ошибок следует вернуться к интеграционному и модульному тестированию.

Деплой и масштабирование

Для развертывания микросервисов в среде production будем, как говорилось ранее, использовать Docker. Для создания контейнера на основе нескольких образов необходимо использовать средство docker-compose. Для брокера сообщений следует задеплоить отдельный контейнер. Для балансировки нагрузки необходимо использовать kubernetes. Благодаря используемой архитектуре и технологиям достигается хорошая масштабируемость каждого микросервиса в частности. В свою очередь возможность создания нескольких экземпляров каждого микросервиса обеспечивает хорошую масштабируемость системы в целом.