МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра вычислительной математики

Курсовой проект

Разработка микросервисных масштабируемых приложений на языке Golang

Выдренко Егор Дмитриевич

студент 3 курса 5 группы,
специальность «прикладная математика»

Эксперт-консультант от филиала кафедры:
Черник Дмитрий Викторович

Руководитель от кафедры:
Мандрик Павел Алексеевич

Минск, 2021

ОГЛАВЛЕНИЕ

ЗАДАНИЕ

Создать контейнерезированное микросервисное приложение по выбранной тематике.

Требования к реализации:

1. От 3 разделенных по доменным областям сервисов, пример: сервис управления пользователями, сервис для обработки логического состояния, сервис каталогизации / справочной информации.

2. Разделить хранение данных на несколько изолированных хранилищ (база данных).

3. Реализовать коммуникацию и хранение промежуточного состояния используя либо систему обмена сообщениями (rabbitmq, service bus, etc...), либо протокол бинарного взаимодействия (grpc).

4. Подготовить каждое из приложений для запуска в изолированном окружении, используя любую систему для запуска и управления контейнерами (docker swarm, kubernetes)

5. Реализовать авторизацию на основе базовой (basic) или на основе токена авторизации (jwt)

6. Удостовериться в корректности работы приложения, добавив автоматические юнит-тесты.

7. Протестировать конечную реализацию на наличие состояний гонки.

Оформить отчетность, отражающую детали реализации, обоснование выбранных средств и технологий, описание архитектуры и проблем с которыми столкнулись. Добавить рекомендации по улучшению.

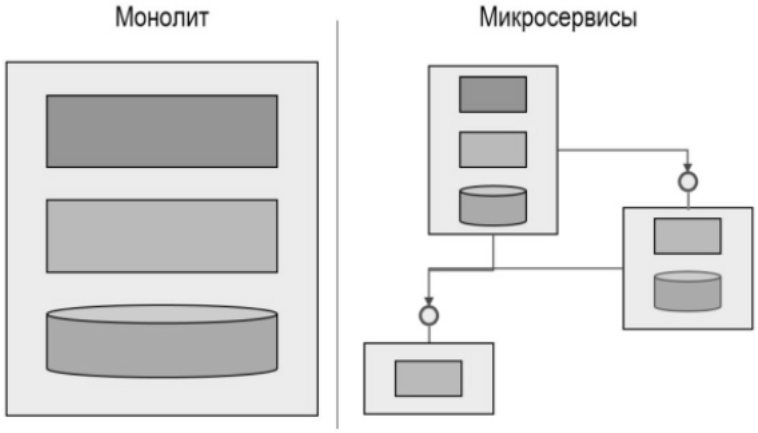
АРХИТЕКТУРА ПРИЛОЖЕНИЯ

Микросервисная архитектура – вариант сервис-ориентированной архитектуры программного обеспечения, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей – *микросервисов*. То есть микросервисная архитектура – это слабо связанная архитектура, компоненты которой независимы по отношению друг к другу

Микросервис - небольшое независимое приложение, которое отвечает за конкретную задачу, имеет собственное хранилище данных и общается с другими сервисами через сетевые коммуникационные протоколы.

Чаще всего микросервисы противопоставляют крупному приложению – *монолиту*.

Рисунок



Свойства микросервисной архитектуры

- Легкозаменяемые модули
- Модули могут быть реализованы с использованием различных языков программирования

- Независимость модулей (если в каком-нибудь из процессов возникает ошибка, остальные процессы не будут)
- Приложения легко масштабируемы
- Простота развертывания (можно развертывать только изменяющиеся микросервисы, независимо от остальной системы, что позволяет производить обновления чаще и быстрее.)

ОСНОВНАЯ ЧАСТЬ

Логика приложения

Микросервисное приложение “личная библиотека” позволяет пользователю

- сохранять книги, которые он прочитал
- смотреть книги, которые прочитал другой пользователь
- искать книги определенных авторов

В процессе реализации было создано два микросервиса

1) *UserService*
2) *BookService*

Первый используется для создания, получения, обновления (в будущем для регистрации) пользователей. Второй используется выполнения тех же операций с книгами. Для хранения информации о пользователях используется *PostreSQL*. Данные о книгах хранятся в *MongoBD*. Для коммуникации и хранения промежуточного состояния используется протокол бинарного взаимодействия *gRPC*.

Программная реализация

gRPC – высокопроизводительный фреймворк, разработанный компанией google для вызовов удаленны процедур (RPC). Высокая производительность достигается за счет использования протокола HTTP/2 и Protocol Buffers. Для передачи данных используются proto-запросы и proto-ответы.

API RPC

Как предшественник REST, RPC (удаленный вызов процедур) представляет собой программную архитектуру, восходящую к 1970-м годам. RPC

позволяет вызывать функцию на удаленном сервере в определенном формате и получать ответ в том же формате. Не имеет значения, какой формат использует сервер, выполняющий запрос, и не имеет значения, локальный это сервер или удаленный. RPC позволяет вызывать функцию на сервере и получать результат в том же формате.

Основная концепция RPC API аналогична концепции REST API. RPC API определяет правила взаимодействия и методы, которые клиент может использовать для взаимодействия с ним. Клиенты отправляют вызовы, которые используют «аргументы» для вызова этих методов. Однако в случае RPC API метод находится в URL-адресе. Аргументы, вызывающие методы, находятся в строке запроса

API gRPC

Как вариант архитектуры RPC, gRPC был создан Google для ускорения передачи данных между микросервисами и другими системами, которым необходимо взаимодействовать друг с другом. По сравнению с REST API, gRPC API уникальны в следующих отношениях:

- Протобуф (Protobuf) вместо JSON
- Построен на HTTP 2 вместо HTTP 1.1
- Создание собственного кода вместо использования сторонних инструментов, таких как Swagger
- Передача сообщений в 7-10 раз быстрее
- Более долгая имплементация и реализация, чем REST

Protobuf вместо JSON / XML

И REST API, и RPC API отправляют и получают сообщения с использованием форматов обмена сообщениями JSON или XML. Они также могут использовать другие форматы, но наиболее распространены JSON и XML. Из них JSON стал самым популярным форматом, поскольку он гибкий, эффективный, платформенно-независимый и не зависит от языка. Он также основан на тексте и удобочитаем для человека, что упрощает работу операторам. Проблема в том, что для определенных случаев использования JSON недостаточно быстр или легковесен при передаче данных между системами.

В отличие от REST и RPC, gRPC преодолевает проблемы, связанные со скоростью и весом, и предлагает большую эффективность при передаче сообщений, используя формат обмена сообщениями Protobuf (буферы протокола). Вот несколько подробностей о Protobuf:

- Независимость от платформы и языка, например как JSON
- Сериализует и десериализует структурированные данные для передачи в двоичном формате.
- Поскольку он является сильно сжатым форматом, он не обеспечивает удобочитаемости JSON.
- Ускоряет передачу данных, удаляя множество обязанностей, которыми управляет JSON, поэтому он может сосредоточиться исключительно на сериализации и десериализации данных.
- Передача данных происходит быстрее, потому что Protobuf уменьшает размер сообщений и служит легким форматом обмена сообщениями.

Proto – язык разметки интерфейсов, декларативное описание модели и контактов. Каждый сервис имеет имя, список методов (с ключевым словом RPC), входящие данные и результирующие данные. В message тоже есть имя и список полей, имя поля и индекс. При сериализации protobuf использует индекс. Код клиента и шаблон для сервера генерируется на основе proto, причем это возможно для любого поддерживаемого языка.

Реализация user.proto

```
7   service UserService {
8     rpc GetUser(GetUserRequest) returns (GetUserResponse) {}
9     rpc CreateUser(User) returns (CreateUserResponse) {}
10    rpc GetAllUsers(GetAllUsersRequest) returns (GetAllUsersResponse) {}
11  }
12
13  message User {
14    int64 id = 1;
15    string name = 2;
16    string email = 3;
17    string password = 4;
18    repeated string orderedBooks = 5;
19  }
20
21  message GetUserRequest {
22    int64 id = 1;
23  }
24
25  message GetUserResponse {
26    string name = 1;
27    string email = 2;
28    string password = 3;
29    repeated string orderedBooks = 4;
30  }
31
32  message CreateUserResponse {
33    User user = 1;
34  }
35
36  message GetAllUsersRequest {
37
38  }
39
40  message GetAllUsersResponse {
41    repeated User allUsers = 1;
42  }
```

Пример сгенерированного интерфейса

```
type UserServiceClient interface {
        GetUser(ctx context.Context, in *GetUserRequest, opts ...grpc.CallOption) (*GetUserResponse, error)
        CreateUser(ctx context.Context, in *User, opts ...grpc.CallOption) (*CreateUserResponse, error)
        GetAllUsers(ctx context.Context, in *GetAllUsersRequest, opts ...grpc.CallOption) (*GetAllUsersResponse, error)
}
```

БАЗЫ ДАННЫХ

MongoDB — система управления базами данных, которая работает с документоориентированной моделью данных. В отличие от реляционных СУБД, MongoDB не требуются таблицы, схемы или отдельный язык запросов. Информация хранится в виде документов либо коллекций.

Разработчики позиционируют продукт как промежуточное звено между классическими СУБД и NoSQL. MongoDB не использует схемы, как это делают реляционные базы данных, что повышает производительность всей системы.

Особенности

У MongoDB есть ряд свойств, которые выделяют ее на фоне других продуктов:

1. Кроссплатформенность. СУБД разработана на языке программирования C++, поэтому с легкостью интегрируется под любую операционную систему (Windows, Linux, MacOS и др.).
2. Формат данных. MongoDB использует собственный формат хранения информации — Binary JavaScript Object Notation (BSON), который построен на основе языка JavaScript.
3. Документ. Если реляционные БД используют строки, то MongoDB — документы, которые хранят значения и ключи.
4. Вместо таблиц MongoDB использует коллекции. Они содержат разные типы наборов данных
5. Репликация. Система хранения информации в СУБД представлена узлами. Существует один главный и множество вторичных. Данные реплицируются между точками. Если один первичный узел выходит из строя, то вторичный становится главным.

6. Индексация. Технология применяется к любому полю в документе на усмотрение пользователя. Проиндексированная информация обрабатывается быстрее.
7. Для сохранения данных большого размера MongoDB использует собственную технологию GridFS, состоящую из двух коллекций. В первой (files) содержатся имена файлов и метаданные по ним. Вторая (chunks) сохраняет сегменты информации, размер которых не превышает 256 Кб.
8. СУБД осуществляет поиск по специальным запросам. Например, пользователь может создать диапазонный запрос и мгновенно получить ответ.
9. Балансировщик нагрузки используется в СУБД не только для распределения нагрузки между разными базами данных, но и для горизонтального масштабирования. Сегменты БД распределяются по разным узлам, что повышает производительность. При этом базы данных, расположенные на разных узлах, синхронизированы между собой и обеспечивают целостность информации для клиента.
10. MongoDB может поставляться для конечного клиента как облачное решение.

СУБД используют для хранения событий в системе (логирование), записи информации с датчиков мониторинга на предприятии, а также в сфере электронной коммерции и мобильных приложений. Часто MongoDB применяют как хранилище в сфере машинного обучения и искусственного интеллекта.

MongoDB относится к классу NoSQL СУБД и работает с документами, а не с записями. Это кроссплатформенный продукт, который легко внедряется в любую операционную систему. Ряд уникальных особенностей позволяет использовать СУБД под определённые задачи, в которых она обеспечивает максимальную производительность и надежность.

Подключение к базе данных

```
func Connect(ctx context.Context, uri string) (*mongo.Client, error) {
        client, err := mongo.Connect(ctx, options.Client().ApplyURI(uri))
        return client, err
}
```

PostgreSQL — это популярная свободная объектно-реляционная система управления базами данных. PostgreSQL базируется на языке SQL и поддерживает многочисленные возможности.

Преимущества PostgreSQL:

- поддержка БД неограниченного размера;
- мощные и надёжные механизмы транзакций и репликации;
- расширяемая система встроенных языков программирования и поддержка загрузки C-совместимых модулей;
- наследование;
- легкая расширяемость.

Текущие ограничения PostgreSQL:

- Нет ограничений на максимальный размер базы данных
- Нет ограничений на количество записей в таблице
- Нет ограничений на количество индексов в таблице
- Максимальный размер таблицы — 32 Тбайт
- Максимальный размер записи — 1,6 Тбайт
- Максимальный размер поля — 1 Гбайт
- Максимум полей в записи250—1600 (в зависимости от типов полей)

Особенности PostgreSQL:

Функции в PostgreSQL являются блоками кода, исполняемыми на сервере, а не на клиенте БД. Хотя они могут писаться на чистом SQL, реализация дополнительной логики, например, условных переходов и циклов, выходит за рамки собственно SQL и требует использования некоторых языковых расширений. Функции могут писаться с использованием различных языков

программирования. PostgreSQL допускает использование функций, возвращающих набор записей, который далее можно использовать так же, как и результат выполнения обычного запроса. Функции могут выполняться как с правами их создателя, так и с правами текущего пользователя. Иногда функции отождествляются с хранимыми процедурами, однако между этими понятиями есть различие.

Триггеры в PostgreSQL определяются как функции, инициируемые DML-операциями. Например, операция INSERT может запускать триггер, проверяющий добавленную запись на соответствия определённым условиям. При написании функций для триггеров могут использоваться различные языки программирования. Триггеры ассоциируются с таблицами. Множественные триггеры выполняются в алфавитном порядке.

Механизм правил в PostgreSQL представляет собой механизм создания пользовательских обработчиков не только DML-операций, но и операции выборки. Основное отличие от механизма триггеров заключается в том, что правила срабатывают на этапе разбора запроса, до выбора оптимального плана выполнения и самого процесса выполнения. Правила позволяют переопределять поведение системы при выполнении SQL-операции к таблице.

Индексы в PostgreSQL следующих типов: B-дерево, хэш, R-дерево, GiST, GIN. При необходимости можно создавать новые типы индексов, хотя это далеко не тривиальный процесс.

Многоверсионность поддерживается в PostgreSQL — возможна одновременнуя модификация БД несколькими пользователями с помощью механизма Multiversion Concurrency Control (MVCC). Благодаря этому соблюдаются требования ACID, и практически отпадает нужда в блокировках чтения.

Расширение PostgreSQL для собственных нужд возможно практически в любом аспекте. Есть возможность добавлять собственные преобразования типов, типы данных, домены (пользовательские типы с изначально наложенными ограничениями), функции (включая агрегатные), индексы, операторы (включая переопределение уже существующих) и процедурные языки.

Наследование в PostgreSQL реализовано на уровне таблиц. Таблицы могут наследовать характеристики и наборы полей от других таблиц

(родительских). При этом данные, добавленные в порождённую таблицу, автоматически будут участвовать (если это не указано отдельно) в запросах к родительской таблице.

Подключение к БД

```
func CreateConnection() (*sqlx.DB, error) {
        config := fmt.Sprintf("port=%d host=%s user=%s "+
                "password=%s dbname=%s sslmode=disable",
                host_port, hostname, username, password, database_name)
        db, err := sqlx.Connect("postgres", config)
        if err != nil {
                return nil, err
        }
        return db, nil
}
```

DOCKER(ТЕОРИЯ)

Docker (Докер) — программное обеспечение с открытым исходным кодом, применяемое для разработки, тестирования, доставки и запуска веб-приложений в средах с поддержкой контейнеризации. Он нужен для более эффективного использование системы и ресурсов, быстрого развертывания готовых программных продуктов, а также для их масштабирования и переноса в другие среды с гарантированным сохранением стабильной работы. Основной принцип работы Docker — контейнеризация приложений. Этот тип виртуализации позволяет упаковывать программное обеспечение по изолированным средам — контейнерам. Каждый из этих виртуальных блоков содержит все нужные элементы для работы приложения. Это дает возможность одновременного запуска большого количества контейнеров на одном хосте. Контейнеры позволяют упаковать в единый образ приложение и все его зависимости: библиотеки, системные утилиты и файлы настройки. Это упрощает перенос приложения на другую инфраструктуру.

Например, разработчики создают приложение в системе разработки, там все настроено и приложение работает. Когда приложение готово, его нужно перенести в систему тестирования и затем в продуктивную среду. И если в этих системах будет не хватать какой-нибудь зависимости, то приложение не будет работать. В этом случае программистам придется отвлечься от разработки и совместно с командой поддержки разбираться в ситуации. Контейнеры позволяют избежать такой проблемы, потому что они содержат в себе все необходимое для запуска приложения. Программисты смогут сосредоточиться на разработке, а не решении инфраструктурных проблем.

ЗАКЛЮЧЕНИЕ

Итоги выполненной работы:

- Создано приложение состоящее из 2 микросервисов
- Использовано 2 хранилища (postgreSQL, MongoDB)
- Для хранения и коммуникации используется протокол бинарного взаимодействия gRPC
- Изучена теория контейнеризации

В будущем нужно реализовать:

- Отдельный микросервис для связи уже имеющихся
- Разобраться с докером и добавить контейнеризацию
- Реализовать авторизацию пользователей

ИСПОЛЬЗУЕМЫЕ ТЕОРЕТИЧЕСКИЕ МАТЕРИАЛЫ

https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0

https://habr.com/ru/company/raiffeisenbank/blog/346380/

https://mcs.mail.ru/blog/prostym-jazykom-o-mikroservisnoj-arhitekture

https://habr.com/ru/post/565020/

https://tproger.ru/articles/grpc-integration-experience/

https://habr.com/ru/company/yandex/blog/484068/

https://ru.wikipedia.org/wiki/GRPC

https://medium.com/maddevs-io/introduction-to-grpc-6de0d9c0fe61

https://itglobal.com/ru-by/company/glossary/mongodb/

https://web-creator.ru/articles/postgresql

https://eternalhost.net/blog/razrabotka/chto-takoe-docker

https://selectel.ru/blog/what-is-docker/