

Обзор учебного микросервиса

Денис Адамчук



Цель лекции

- Познакомиться с архитектурой микросервиса
- Обзор технологий и обоснование выбора:
 - Каркас сервиса (gRPC, Protobuf)
 - Система сборки (Conan, CMake)
 - Внешние API (Overpass, Nominatim, OpenMeteo)
- Подготовиться к практической работе с кодом



Структура лекции

- Обзор микросервиса
- Архитектура
- Технологии
- Протокол
- Реализация
- Работа с кодом
- Демонстрация и Q&A



Коротко о сервисе

Geo Service – это обучающий пример микросервиса на базе gRPC (*Google RPC* – сетевой фреймворк для создания микросервисов). Он предоставляет функциональность для поиска городов и регионов по различным критериям, используя открытые API.

Технические особенности:

- Реализован на C++
- Создан на базе gRPC
- Собирает информацию с других сервисов, расположенных в Интернете

<https://github.com/cqginternship/geo-service>

В ветке master здесь живет прототип, предназначенный для расширения

Коротко о сервисе

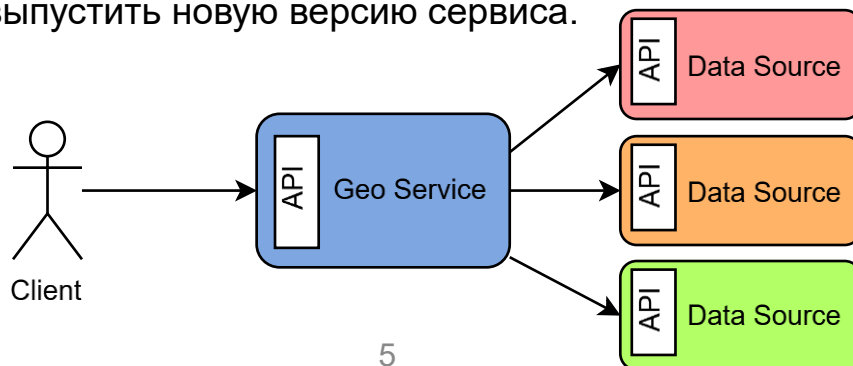
Концепция прокси-сервиса

Различные источники данных скрываются для пользователя за единым API.

В чем смысл этого подхода:

- Пользователю не нужно знать особенности API каждого источника данных.
- Пользователю не придется менять код, если:
 - Один из источников данных выйдет из строя.
 - Изменится API источника данных.

Хотя, возможно, придется выпустить новую версию сервиса.



Возможности сервиса

- Поиск городов по имени/координатам

Найти город по имени
"Samara"

Geo Service

Имя: Самара (Samara)
Страна: Россия (Russia)
Координаты: 53.1956255,50.1014927

Найти город по координатам
55.991971, 37.214241

Geo Service

Имя: Зеленоград (Zelenograd)
Страна: Россия (Russia)
Координаты: 55.991971,37.214241

Возможности сервиса

- Поиск городов по имени/координатам
- Поиск регионов с фильтрами

Найти регионы
с международными аэропортами
в радиусе 200км
от точки 35,35

Geo Service

Имя: Hatay (Hatay)
Страна: Türkiye (Turkey)
Координаты: 36.3451332,36.0748022

Имя: محافظة جبل لبنان (Mount Lebanon Governorate)
Страна: لبنان (Lebanon)
Координаты: 33.737305,35.5998898

Реализованные критерии:

- наличие международных аэропортов
- наличие горных вершин (выше заданной величины)
- наличие морских пляжей
- наличие соленых озер

Критерии можно комбинировать.

Возможности сервиса

- Поиск городов по имени/координатам
- Поиск регионов с фильтрами
- Прогноз погоды

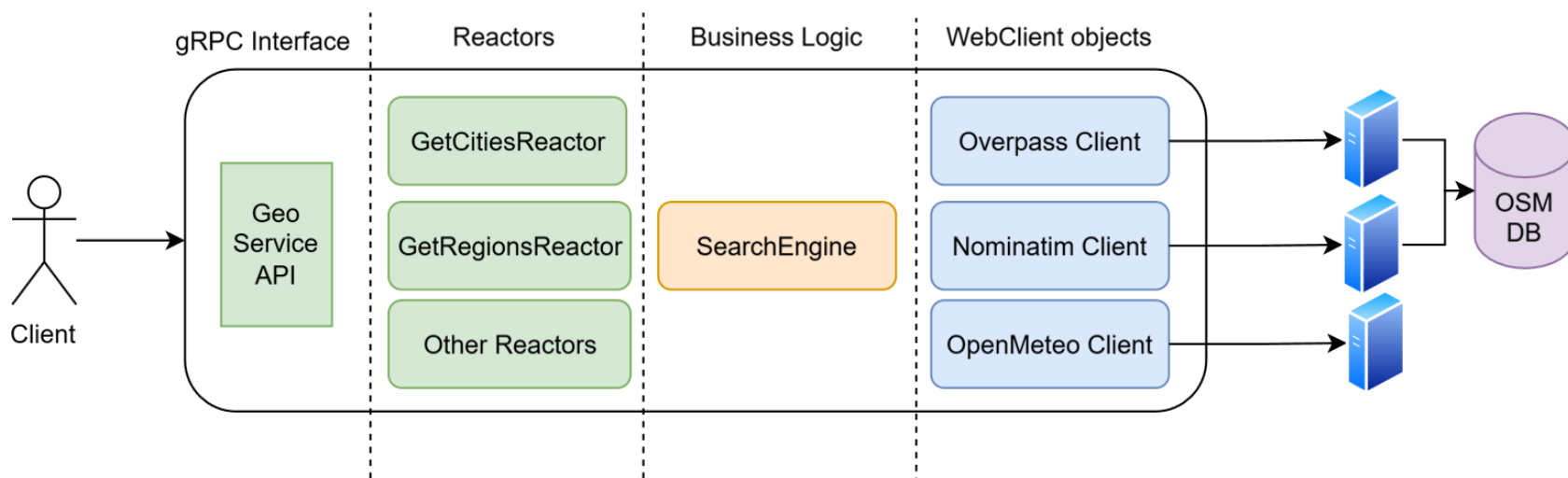
Получить прогноз погоды
в точке 55.7558, 37.6173
на начало января
следующего года

(прогноз вычисляется
по историческим данным
за последние 5 лет)

Geo Service

Минимальная температура: -9.5°C
Максимальная температура: 4.8°C
Средняя температура: -3.7°C

Архитектура





Технологии






gRPC: универсальный язык общения микросервисов

gRPC — это современный сетевой фреймворк от Google для обмена сообщениями между сервисами.

- Полноценный фреймворк для создания сервисов
- Позволяет описывать API в .proto-файле
- Генерирует клиентский и серверный код
- Применяется в микросервисной архитектуре

Технологии

gRPC: ключевые особенности

-  **Производительность:** бинарная сериализация + HTTP/2
-  **Поддержка стриминга:** клиентские, серверные и двунаправленные потоки
Поток – вид запроса, который предполагает многократные ответы
-  **Генерация кода:** поддержка многих языков (C++, Python, Go и др.)
-  **Строгая типизация:** меньше ошибок на этапе интеграции
-  **Прозрачность:** Автоматическое преобразование структур между языками



Технологии

gRPC: зачем он в обучении?

gRPC удобен еще и тем, что упрощает обучение.
Без него пришлось бы (и это необходимый минимум!):

- Выбирать решение (библиотека, протокол) для сетевого взаимодействия
- Писать код обработки входящих сетевых сообщений
- Создавать свою потоковую модель
- Реализовывать поддержку stream (это не тривиально)

Технологии

gRPC: сервер

```
int main(int argc, char** argv)
{
    std::string server_address("0.0.0.0:50051");
    GeoServiceImpl service; // GeoServiceImpl - фабрика «реакторов»

    grpc::ServerBuilder builder;
    builder.AddListeningPort(server_address, grpc::InsecureServerCredentials());

    builder.RegisterService(&service);

    std::unique_ptr<grpc::Server> server(builder.BuildAndStart());

    std::this_thread::sleep_for(std::chrono::seconds(300));
    server->Shutdown(std::chrono::system_clock::now() + std::chrono::seconds(1));
    return 0;
}
```

gRPC создает сервер, который «слушает» запросы на порту 50051.
Клиент, запущенный локально, может подключаться к 127.0.0.1:50051.



Технологии

Protobuf: формат описания данных

Protobuf — формат описания и сериализации данных, разработанный Google.

- Используется в gRPC для описания структур и API
- Работает на основе .proto-файлов
- Генерирует код на разных языках
- Позволяет сохранять, передавать и валидировать сложные структуры



Технологии

Protobuf: ключевые особенности

- ♥ Простота: синтаксис .proto файлов легко освоить
- 📦 Компактность: бинарный формат, намного меньше JSON
- 🚀 Быстрота: эффективная сериализация и десериализация
- 🧱 Строгая типизация: ошибки возникают на ранних этапах
- 🔄 Поддержка эволюции: можно добавлять поля без поломки старых клиентов



Технологии

Зачем нужен менеджер пакетов?

Без менеджера зависимостей требуется решать эти задачи самостоятельно:

- Поиск сторонних библиотек определенной конфигурации (архитектура, тип сборки) или сборка их из исходного кода



Технологии

Зачем нужен менеджер пакетов?

Без менеджера зависимостей требуется решать эти задачи самостоятельно:

- Поиск сторонних библиотек определенной конфигурации (архитектура, тип сборки) или сборка их из исходного кода
- Разруливание совместимости версий



Технологии

Зачем нужен менеджер пакетов?

Без менеджера зависимостей требуется решать эти задачи самостоятельно:

- Поиск сторонних библиотек определенной конфигурации (архитектура, тип сборки) или сборка их из исходного кода
- Разруливание совместимости версий
- Обеспечение повторяемости сборки на машине каждого разработчика



Технологии

Зачем нужен менеджер пакетов?

Проблемы `apt install libsome`:

- На разных машинах может установиться разная версия
- Версии устанавливаются глобально и могут конфликтовать друг с другом



Технологии

Conan: менеджер пакетов для C++





В отличие от многих современных языков, в C++ нет встроенного механизма управления зависимостями.

Conan - это сторонний менеджер пакетов для C++.

- Позволяет описывать зависимости проекта в конфигурационном файле
- Работает аналогично pip (Python) или npm (JavaScript)
- Позволяет создать единое, повторяемое окружение

Технологии

Copan: ключевые особенности

-  Автоматическая загрузка и сборка библиотек
-  Поддержка разных профилей сборки (debug/release, gcc/clang)
-  Локальный кэш — повторное использование библиотек без пересборки
-  Управление версиями и зависимостями

Технологии

Conan: пример

Список зависимостей (conanfile.txt)

```
[requires]
grpc/1.65.0
libcurl/8.9.1
fmt/11.0.2
rapidjson/cci.20230929
```

```
[generators]
CMakeDeps
CMakeToolchain
```

generators создают файлы
для интеграции с CMake

Профиль (conan-profile-debug)

```
[settings]
arch=x86_64
build_type=Debug
compiler=gcc
compiler.cppstd=gnu17
compiler.libcxx=libstdc++11
compiler.version=12
os=Linux
```

\$ conan install . -pr:h /root/conan-profile-debug -pr:b /root/conan-profile-debug

1. Установить зависимости, соответствующие conan-profile-debug в conan cache (папка ~/.conan2)
2. Создать файлы, которые позволят использовать директиву find_package() в CMake



Технологии

Conan и «поиск сторонних библиотек»

Пример: Вам нужен конкретный вариант библиотеки OpenSSL:

- Для архитектуры x86_64
- С типом сборки Debug
- С включенной поддержкой TLS 1.3

Без менеджера зависимостей:

- Нужно искать исходники
- Разбираться с флагами сборки
- Настраивать вручную CMake или Makefile

С Conan используем conan profile либо просто указываем ключи:

```
conan install . -s arch=x86_64 -s build_type=Debug -o openssl:enable_tls1_3=True
```

Технологии

Conan и «совместимость версий/повторяемость сборки»

Пример: Проект использует SuperLib. Вы пишете код, используя версию 1.0.

```
#include <SuperLib/version.h>
void printVersion() {
    std::string s = super_lib::current_version::toString();
    std::cout << "SuperLib version=" << s << std::endl;
}
```

Другой программист взял версию 2.0, и не может скомпилировать код:

```
#include <SuperLib/version.h>
void sendVersion() {
    std::string s = super_lib::current_version::toString(); // Ошибка компиляции!
    std::cout << "SuperLib version=" << s << std::endl;
}
```

Причина: В версии 2.0 функция переименована в ToString().

Решение с помощью conanfile.txt:

```
[requires]
superLib/1.0
```




Технологии

СMake: кроссплатформенная система сборки для C++

СMake — это генератор сборочных файлов (например, Makefile или Visual Studio).
Сборочные файлы — это набор параметров
(пути, флаги компиляции и ссылки на зависимости)
для каждой единицы трансляции (translation unit) в C++.

Без СMake даже GeoService собрать было бы очень трудно.

- Позволяет описывать структуру проекта декларативно в едином файле CMakeLists.txt
- Поддерживает зависимости между компонентами проекта
- Интегрируется с Conan и VSCode



Технологии

СMake: ключевые особенности

- 🔧 Поддержка мультиплатформенности
- 🧩 Интеграция с Conan для управления зависимостями
- 💡 Гибкая конфигурация сборки: флаги, опции, профили
- 🔄 Поддерживается всеми IDE (VSCode, CLion и др.)



Технологии


DevContainer: единая среда для разработки

Контейнер – изолированная среда для выполнения программы.

По умолчанию программа внутри контейнера не имеет доступа к ресурсам хоста и не может повлиять на систему (при правильной настройке).

DevContainer – среда для разработки, которая разворачивается в VSCode и обеспечивает одинаковое окружение для всех разработчиков.

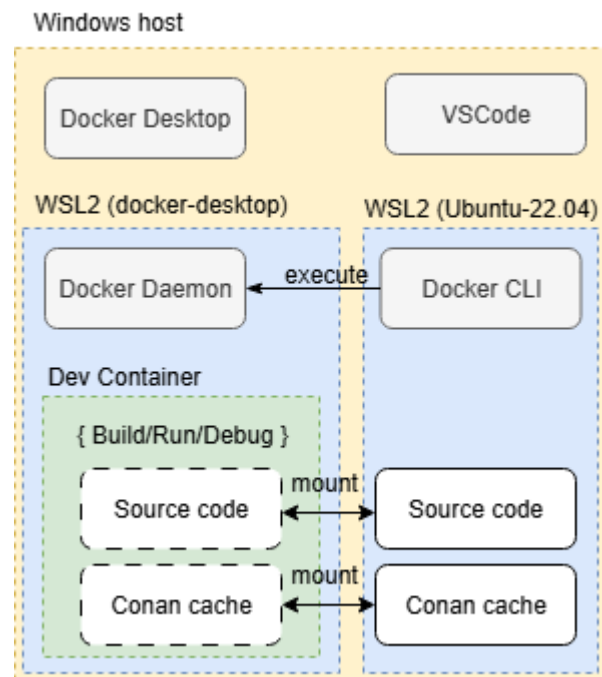
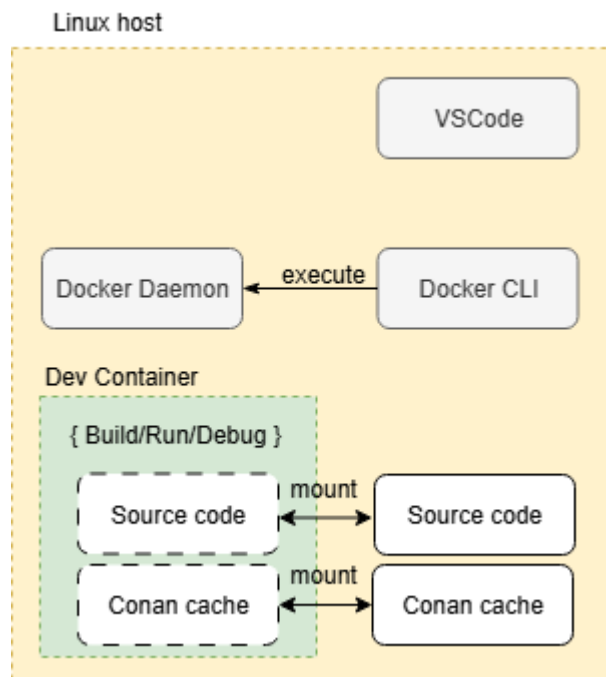
 Контейнеры — временные, поэтому важно сохранять то, что “дорого”

 Всё, что не хранится на хосте, будет утеряно при пересоздании

 Пример: Монтируем Copan-кэш снаружи, чтобы не пересобирать зависимости каждый раз

Технологии

DevContainer: варианты использования





Технологии

DevContainer: зачем он в обучении?

- Одинаковая среда для всех студентов, а значит, вероятно, одинаковые проблемы и их решения (это полезно и студентам и преподавателю)
- Упрощает старт: VSCode сам скачает все нужное (расширения и зависимости)
- Версии утилит и библиотек зафиксированы заранее, следовательно нет конфликтов между версиями Conan, CMake и компиляторов

Поддержка DevContainer: папка .devcontainer в репозитории.

Обычная минимальная конфигурация: devcontainer.json и Dockerfile.

Технологии

DevContainer: Описание контейнера (devcontainer.json)

```
{
  "name": "C++",
  "build": {
    "dockerfile": "Dockerfile"
  },
  // To forward a port from a container into WSL2, not into Windows.
  // Rancher Desktop forwards 50051 to Windows automatically.
  "runArgs": [ "-p", "50051:50051" ],
  "remoteUser": "root",
  "mounts": [ "type=bind,source=${localWorkspaceFolder}/../../.conan2,target=/root/.conan2" ],
  "postStartCommand": "conan install . --output-folder=build --build=missing -pr:h .devcontainer/conan-profile-debug
-pr:b .devcontainer/conan-profile-debug",
  "customizations": {
    "vscode": {
      "extensions": [
        "ms-vscode.cpptools-extension-pack"
      ]
    }
  }
}
```

Технологии

DevContainer: Отображение conan cache

```
"remoteUser": "root",  
"mounts": [ "type=bind,source=${localWorkspaceFolder}/../../.conan2,target=/root/.conan2" ],  
"postStartCommand": "conan install . --output-folder=build --build=missing -pr:h .devcontainer/conan-profile-  
debug -pr:b .devcontainer/conan-profile-debug",
```

“**remoteUser**” – root, следовательно conan cache ~/.conan2 - это /root/.conan2

“**mounts**” – отобразить /root/.conan2 на папку вне контейнера для быстрого восстановления зависимостей при перезапуске контейнера

“**postStartCommand**” – выполнить сбор зависимостей на старте контейнера. После выполнения этой команды контейнер готов к запуску CMake для сборки проекта.



Технологии

DevContainer: Dockerfile

```
FROM gcc:14.2.0

USER root
WORKDIR /root

RUN apt update; \
    apt install -y python3-pip cmake gdb

RUN pip install conan --break-system-packages
```

Dockerfile для нашего devcontainer максимально простой. В нем только компилятор, cmake, gdb и conan.

Этот Dockerfile только для разработки!
Для разворачивания сервиса используют другие Dockerfile, подробно об этом будет рассказано в лекции №5.



Архитектура и интерфейс

Структура .proto-файла

gRPC-сервис описан в .proto-файле:

```
message Point { ... }
message Place { ... }
message Weather { ... }
message CitiesRequest { ... }
message CitiesResponse { ... }

service Geo {
  rpc GetCities(CitiesRequest) returns (CitiesResponse);
  rpc GetRegions(RegionsRequest) returns (RegionsResponse);
  ...
}
```

- Содержит определения структур (Point, Place, Weather)
- Запросы и ответы (CitiesRequest, CitiesResponse, ...)
- Интерфейс сервиса (Geo) с методами
- Используется для генерации серверного и клиентского кода (поставляется клиентам!)



Протокол

Структуры данных

Описание структуры данных Place:

```
// Place represents a geographical entity, such as a city or region.
message Place
{
    // TaggedFeature represents a geographical feature with a position and metadata tags.
    message TaggedFeature
    {
        Point position = 1; // Geographical position of the feature.
        map<string, string> tags = 2; // Metadata tags associated with the feature (e.g., "type": "airport").
    }

    string name = 1;           // Localized name of the place.
    string name_en = 2;        // English name of the place.
    string country = 3;        // Localized name of the country where the place is located.
    string country_en = 4;     // English name of the country where the place is located.
    Point center = 5;          // Geographical center of the place.
    repeated TaggedFeature features = 6; // List of tagged features within the place.
}
```

Протокол

Запросы

Пример запроса CitiesRequest:

```
// CitiesRequest is used to request information about cities.
message CitiesRequest
{
    oneof request
    {
        Point position = 1; // Search for cities near this geographical point.
        string name = 2;    // Search for cities by name (e.g., "New York").
    }

    optional bool include_details = 3; // If true, include detailed information about the cities.
}
```

Протокол

Ответы

Пример ответа CitiesResponse:

```
// CitiesResponse contains a list of cities matching the request.  
message CitiesResponse  
{  
  repeated Place cities = 1; // List of cities matching the search criteria.  
}
```




Протокол

Описание сервиса

```
// Geo provides geographical services, such as finding cities and regions.
service Geo
{
    // GetCities returns a list of cities based on the request criteria.
    rpc GetCities(CitiesRequest) returns (CitiesResponse) {}

    // GetRegions returns a list of regions within a specified square box.
    rpc GetRegions(RegionsRequest) returns (RegionsResponse) {}

    // GetRegionsStream streams regions within a specified square box as they are found.
    rpc GetRegionsStream(RegionsRequest) returns (stream RegionsResponse) {}

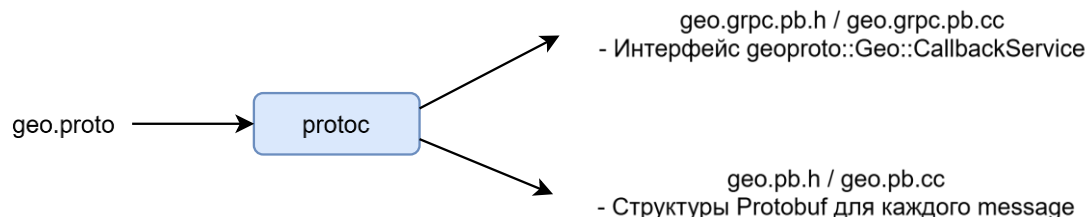
    // GetWeather returns a list of weather information for specific places and times.
    rpc GetWeather(WeatherRequest) returns (WeatherResponse) {}
}
```

Протокол

protoc – компилятор Protobuf


C++ компилятор: компилирует программный код в машинный код

protoc: “компилирует” protobuf в программный код на C++



Реализация

Стек вызовов при обработке запроса клиента



```
geo::SearchEngine::FindCitiesByName() (SearchEngine.cc:183)
geo::GetCitiesReactor::GetCitiesReactor() (GetCitiesReactor.cc:24)
geo::GeoServiceImpl::GetCities() (GeoServiceImpl.cc:79)
geoprotos::Geo::WithCallbackMethod_GetCities<geoprotos::Geo::WithCallbackMethod_GetRegions<geoprotos::Geo::WithCallbackMethod_GetRegionsStream<geoprotos::Geo::Service> >
>::WithCallbackMethod_GetCities()::{}::operator()() const() (proto\geo.grpc.pb.h:228)
{gRPC}
{glibc}
```

geo:: - Код сервиса

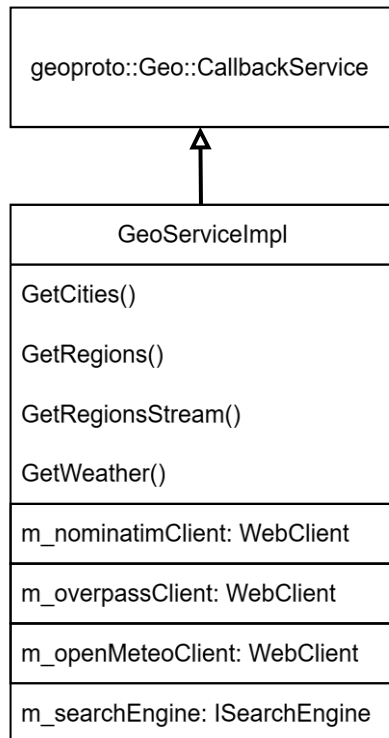
geoprotos:: - Код, сгенерированный protoc

{gRPC} - Код библиотеки gRPC

{glibc} - Системный код запуска потока

Реализация

GeoServiceImpl – фабрика “реакторов”



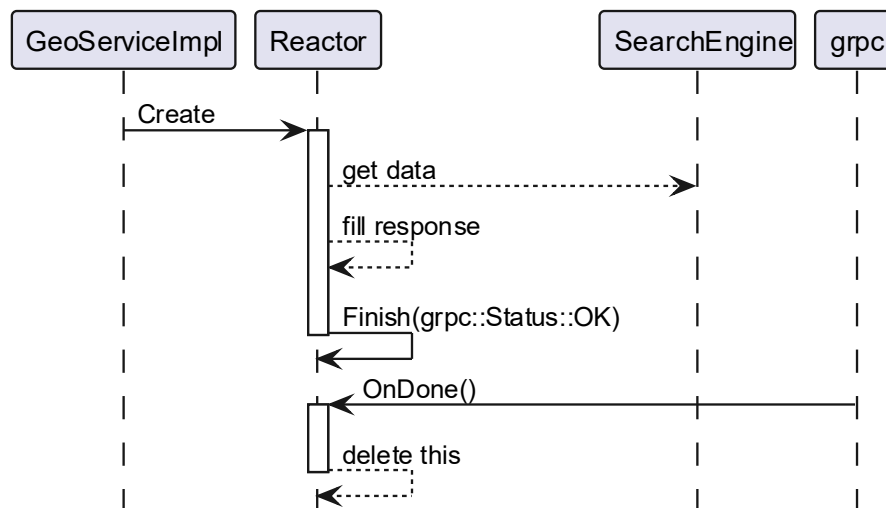
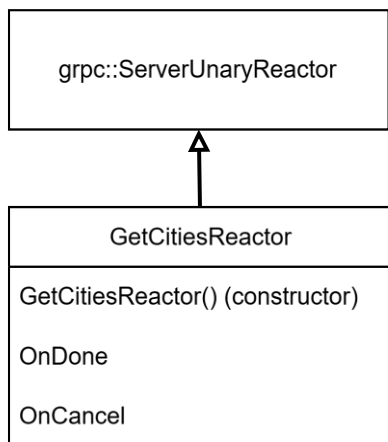
- Реализует виртуальные методы CallbackService
- Методы соответствуют запросам сервиса в geo.proto
- Методы имеют тривиальную реализацию:

```
grpc::ServerUnaryReactor* GeoServiceImpl::GetCities(  
    grpc::CallbackServerContext* context, const geoproto::CitiesRequest* request,  
    geoproto::CitiesResponse* response)  
{  
    // Check if the CitiesRequest is valid.  
    return isCitiesRequestValid(*context, *request) ?  
        new GetCitiesReactor(*request, *response, *m_searchEngine) :  
        nullptr;  
}
```

Реактор – обработчик одного вида запросов из протокола.

Реализация

Простой реактор



1 запрос – 1 ответ

Реализация

Stream-реактор

grpc::ServerWriteReactor<geoproto::RegionsResponse>



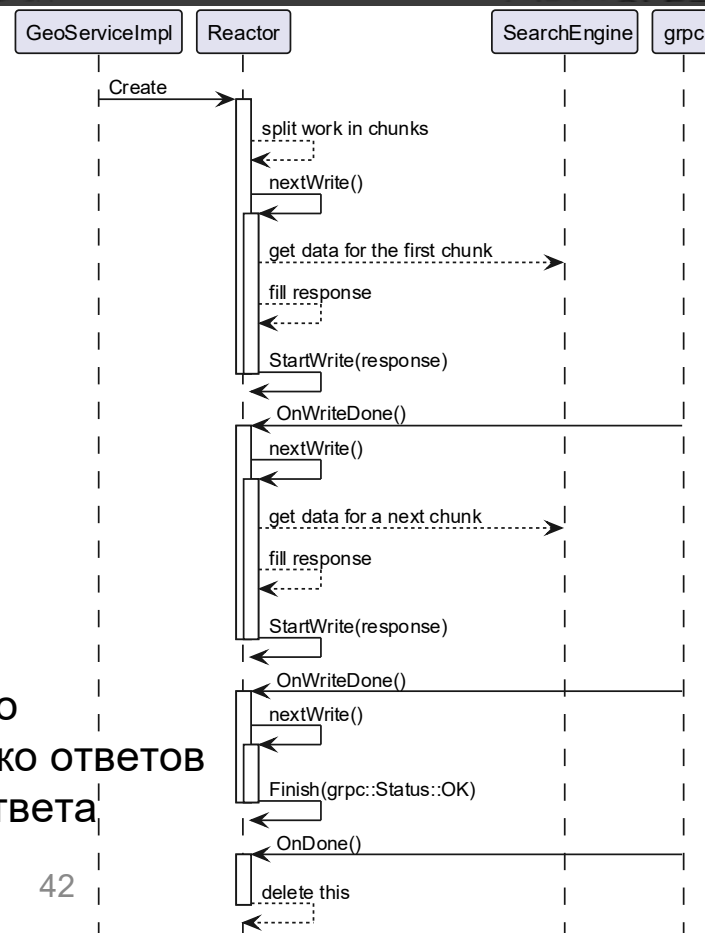
GetRegionsStreamReactor

GetRegionsStreamReactor() (constructor)

OnDone

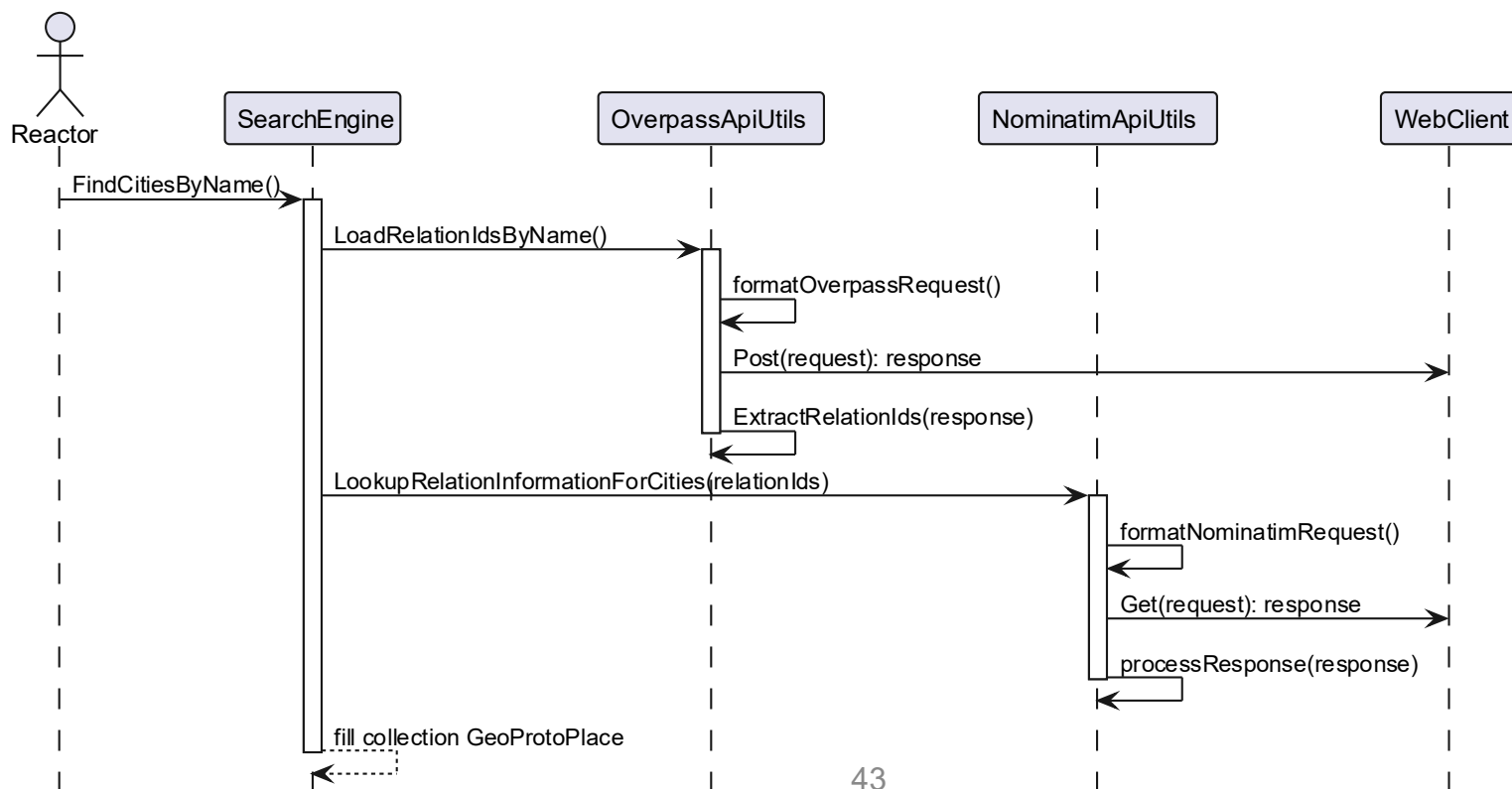
OnCancel

OnWriteDone



Реализация

SearchEngine





Реализация

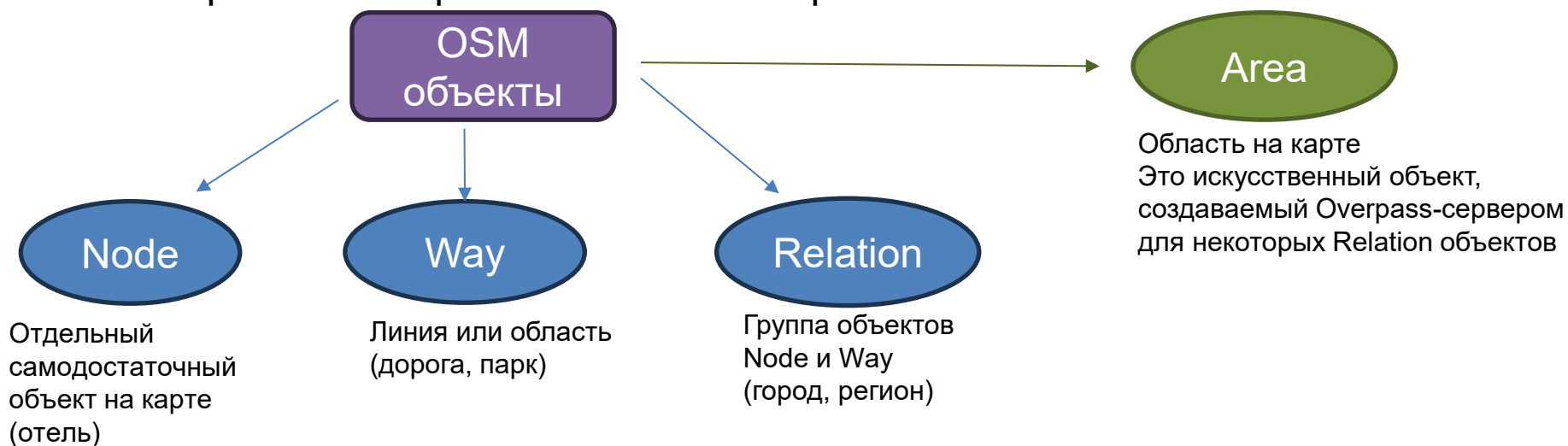
WebClient

- C++ обертка над libcurl
cURL (“Client for URL”) – утилита для работы с HTTP из консоли
libcurl – библиотека в основе cURL
- HTTP поддерживает 9 методов: GET, POST, PUT, PATCH, DELETE и другие. WebClient поддерживает только GET и POST.
- GET - для Nominatim и OpenMeteo (оба - REST API)
REST API - способ коммуникации программ, используя протокол HTTP.
Работает по принципу «клиент-сервер», где клиент отправляет запросы, а сервер возвращает ответы.
- POST - для Overpass Query Language

Реализация

Overpass API

Overpass API — это поисковый движок для OpenStreetMap (OSM), к которому можно отправлять запросы на языке OverpassQL.



Каждый объект имеет текстовые теги в формате ключ-значение. Примеры: "place"="region" или "tourism"="museum".

Реализация

Overpass QL - язык запросов для Overpass API

Пример запроса:

```
// Запрашиваем output в формате JSON
[out:json];

// Оператор выборки rel добавляет relation объект в набор данных по умолчанию
rel(364087); // Этот ID получен с помощью другого запроса к Overpass QL

// Заменить relation объект в наборе данных на объект типа area
map_to_area;

// Операторы выборки node добавляют объекты типа node, имеющие заданные теги и
// принадлежащие заданной области в именованные наборы данных
node[tourism=museum](area) -> .museums;
node[tourism=hotel](area) -> .hotels;

// Объединяем два набора данных в один
(.museums;.hotels);

// Оператор вывода в режиме body включает такие атрибуты как id, теги и координаты
out body;
```

Реализация

Overpass QL - язык запросов для Overpass API

Фрагмент результата:

```
{
// ...
"elements": [
// ...
{
"type": "node",
"id": 1437249373,
"lat": 40.1790204,
"lon": 44.5143603,
"tags": {
"name": "Գրականություն և արվեստի թանգարան",
"name:ca": "Museu Estatal de Literatura i Art",
"name:de": "Staatliches Museum für Literatur und Kunst",
"name:en": "State Museum of Literature and Art",
"name:ru": "Государственный музей литературы и искусства",
"opening_hours": "Tu-Fr 11:00-18:00; Sa-Su 11:00-17:00",
"tourism": "museum"
}
},
// ...
]
```



Реализация

Nominatim

Nominatim – мы используем его как обратный геокодер: по OSM идентификатору получаем подробное описание места.

Пример запроса:

https://nominatim.openstreetmap.org/lookup?format=json&osm_ids=R364087&accept-language=en-US

```
[
  {
    "place_id": 193413846,
    "licence": "Data © OpenStreetMap contributors, ODbL 1.0. http://osm.org/copyright",
    "osm_type": "relation",
    "osm_id": 364087,
    "lat": "40.1777112",
    "lon": "44.5126233",
    "class": "boundary",
    "type": "administrative",
    "place_rank": 7,
    "importance": 0.6906969081971016,
    "addresstype": "city",
    "name": "Yerevan",
    "display_name": "Yerevan, Armenia",
    "address": {
      "city": "Yerevan",
      "ISO3166-2-lvl4": "AM-ER",
      "country": "Armenia",
      "country_code": "am"
    },
    "boundingbox": [
      "40.0658518",
      "40.2417737",
      "44.3621070",
      "44.6217689"
    ]
  }
]
```


Реализация

OpenMeteo

OpenMeteo API – бесплатный API для запроса Погоды (температура, осадки, ветер и прочее). Поддерживает запрос исторических данных и прогнозов. Пример запроса и ответа:

https://archive-api.open-meteo.com/v1/archive?latitude=30.050755&longitude=31.246909&start_date=2022-07-01&end_date=2022-07-05&daily=temperature_2m_max

Пояснение:

Запрос максимальной температуры на высоте 2 метра над уровнем моря в Каире (Египет) с 1 по 5 июля 2022 года.

```
{
  "latitude": 30.052723,
  "longitude": 31.295218,
  "generationtime_ms":
0.04184246063232422,
  "utc_offset_seconds": 0,
  "timezone": "GMT",
  "timezone_abbreviation": "GMT",
  "elevation": 22,
  "daily_units": {
    "time": "iso8601",
    "temperature_2m_max": "°C"
  },
  "daily": {
    "time": [
      "2022-07-01",
      "2022-07-02",
      "2022-07-03",
      "2022-07-04",
      "2022-07-05"
    ],
    "temperature_2m_max": [
      35.6,
      36.7,
      37.6,
      38.3,
      37.9
    ]
  }
}
```



Реализация

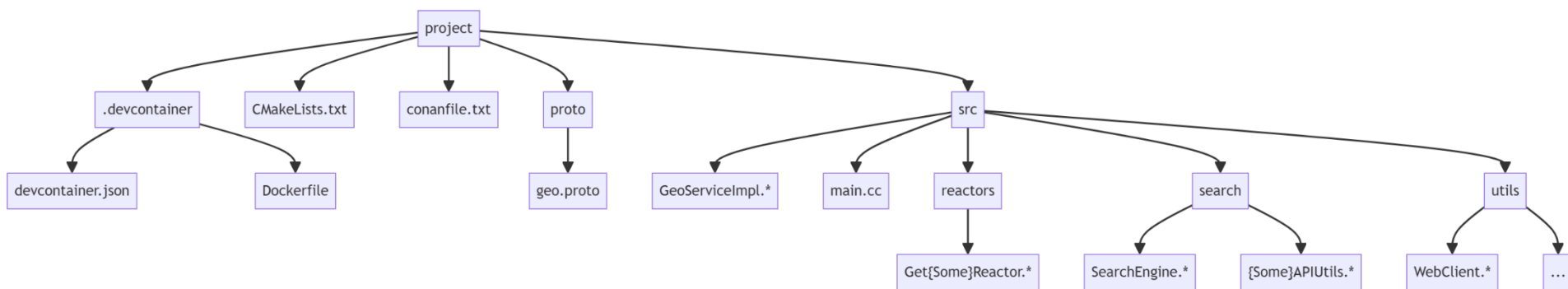
Сравнение внешних API

Критерий	Overpass API	Nominatim API	Open-Meteo API
Назначение	Поиск OSM-объектов по фильтрам	Описание мест	Погодные данные (архив и прогноз)
Тип API	Декларативный язык OverpassQL	REST API	REST API
Формат запроса	HTTP POST с текстом запроса	HTTP GET с параметрами в URL	HTTP GET с параметрами в URL
Что возвращает	OSM-объекты с геометрией и тегами	Название, страна, координаты, тип	Температура, ветер, осадки и др. по датам
Формат ответа	JSON	JSON	JSON

Мы выбрали открытые источники, чтобы вы могли с ними экспериментировать без авторизации и ключей.

Структура проекта

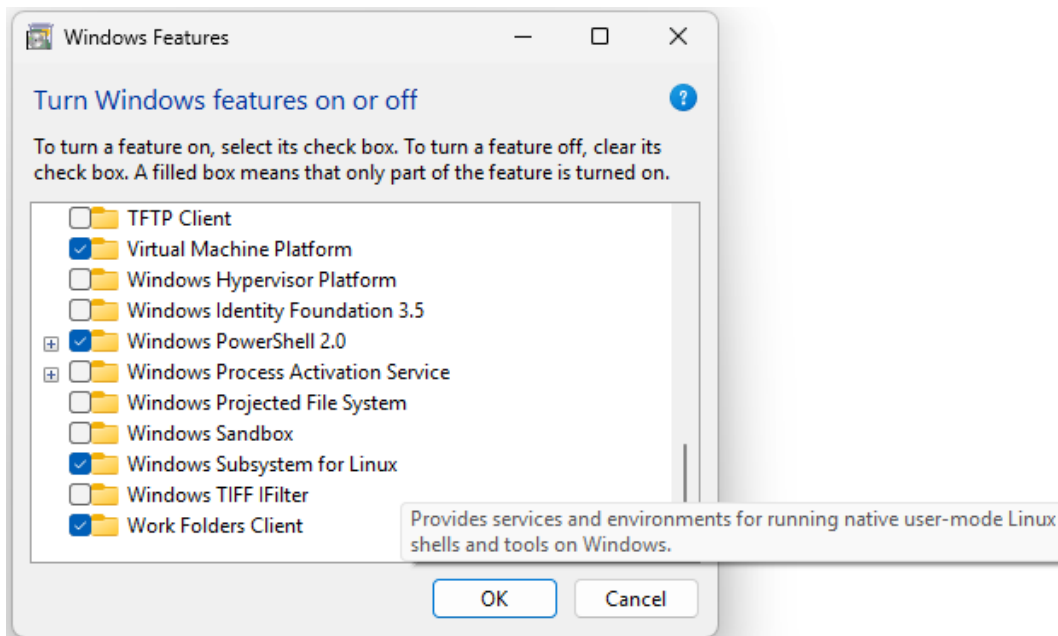
Дерево содержимого проекта



Работа с кодом

Подготовка к работе с кодом через WSL

1. Включите WSL в Windows:





Работа с кодом

Подготовка к работе с кодом через WSL

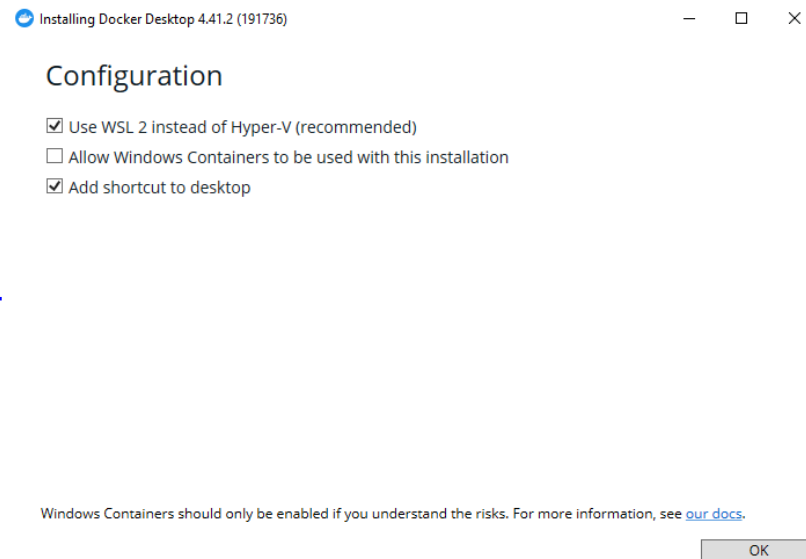
1. Включите WSL в Windows
2. Установите дистрибутив Ubuntu в WSL.
В командной строке Windows выполните:
 - > wsl --install -d Ubuntu-22.04
 - > wsl --update
 - > wsl --shutdown

Работа с кодом

Подготовка к работе с кодом через WSL

1. Включите WSL в Windows
2. Установите дистрибутив Ubuntu в WSL
3. Установите Docker Desktop:

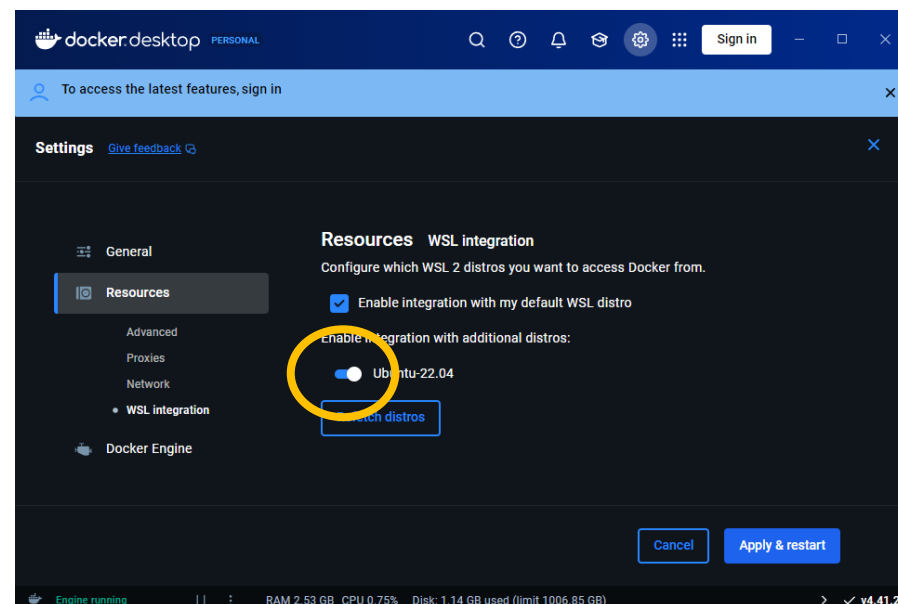
[Docker Desktop:](#)
[The #1 Containerization Tool for Developers](#)



Работа с кодом

Подготовка к работе с кодом через WSL


1. Включите WSL в Windows
2. Установите дистрибутив Ubuntu в WSL
3. Установите Docker Desktop
4. Включите интеграцию с Ubuntu WSL в Docker Desktop



Работа с кодом

Первый запуск VSCode через WSL

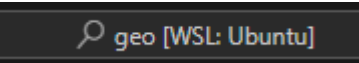
1. Установите VSCode, расширения WSL и Dev Containers
2. Запустите WSL. В командной строке Windows выполните:
> wsl -d Ubuntu
3. Выполните команды в WSL:
\$ cd ~
\$ mkdir -p geo/.conan2
\$ mkdir -p geo/code
\$ cd geo/code
\$ git clone <https://github.com/cqginternship/geo-service> geo
\$ cd geo
\$ code .
4. VSCode запустится в WSL режиме:

 geo [WSL: Ubuntu]

Работа с кодом

Последующие запуски VSCode через WSL

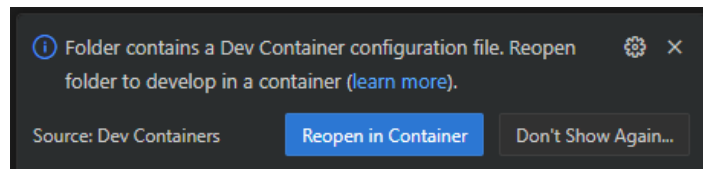
1. Запустите WSL. В командной строке Windows выполните:
> wsl -d Ubuntu
2. Выполните команды в WSL:
\$ cd ~/geo/code/geo
\$ code .
3. VSCode запустится в WSL режиме:



Работа с кодом

Запуск DevContainer

Иногда VSCode на старте предлагает открыть код в DevContainer. Соглашайтесь:

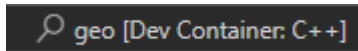


Если не предложил, перейдите в режим Dev Containers вручную. Нажмите Ctrl-Shift-P и выберите:



Запуск DevContainer может занять какое-то время.

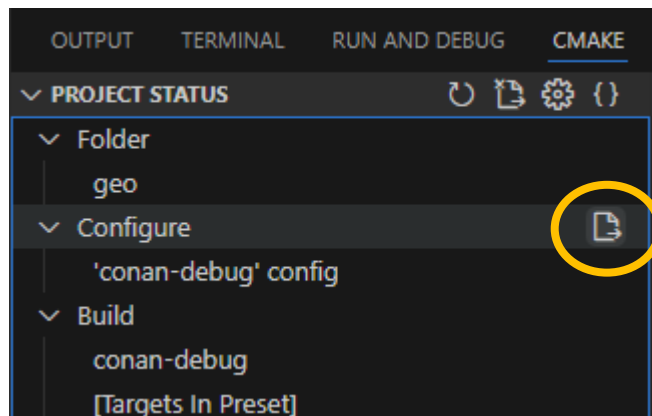
Результат:



Работа с кодом

Сборка проекта в DevContainer

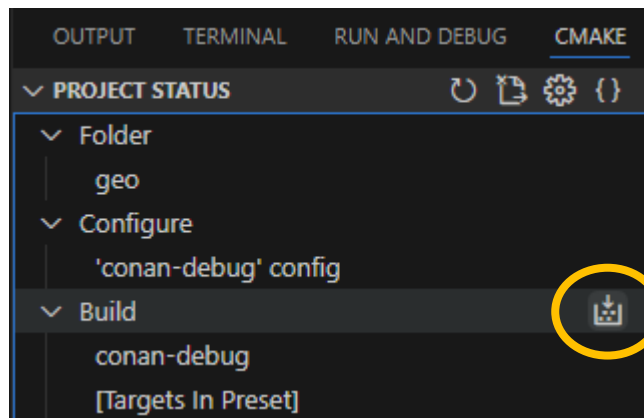
1. Выполните CMake Configure:



Работа с кодом

Сборка проекта в DevContainer

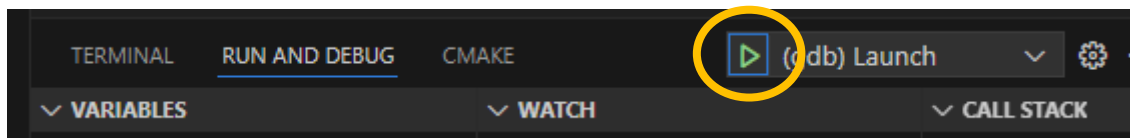
1. Выполните CMake Configure
2. Выполните CMake Build:



Работа с кодом

Запуск сервиса в DevContainer

Для запуска сервиса в отладочном режиме нажмите Start Debugging (F5):





Работа с кодом

Варианты работы без WSL

1. Компьютер с Linux с установленными Docker и VSCode.
VSCode с установленным расширением Dev Containers.
2. Виртуальная машина Linux с установленным Docker.
VSCode на хосте с расширениями Dev Containers и Remote – SSH.



Работа с кодом

Инструкции для работы без WSL

1. Выполните команды в Linux:

```
$ cd ~  
$ mkdir -p geo/.conan2  
$ mkdir -p geo/code  
$ cd geo/code  
$ git clone https://github.com/cqginternship/geo-service geo  
$ cd geo
```

2. Запустите VSCode и откройте в нем папку geo
3. Дальнейшие инструкции смотрите в слайдах “Работа с кодом - Запуск DevContainer” и далее



Демонстрация

- Запускаем DevContainer



Демонстрация

- Запускаем DevContainer
- Выполняем тестовый запуск сервиса



Демонстрация

- Запускаем DevContainer
- Выполняем тестовый запуск сервиса



Q&A