# Лабораторная работа №6 Использование архитектурных решений для обеспечения качества ПО

1. Цель работы
   1. Изучить процесс разработки приложений с использованием микросервисной архитектуры.
   2. Изучить процесс развертывания приложений с использованием микросервисной архитектуры.
2. Литература
   1. Зверева В. П., Сопровождение и обслуживание программного обеспечения компьютерных систем : учебник для студ. учреждений сред. проф. Образования / В. П. Зверева, А. В. Назаров. – М. : Издательский центр «Академия», 2018. – 256 с.
3. Подготовка к работе
   1. Повторить теоретический материал (см. п.2).
   2. Изучить описание лабораторной работы.
4. Основное оборудование
   1. Персональный компьютер.
5. Задание
   1. Подготовка
      1. Создать или использовать готовую виртуальную машину Ubuntu,
      2. В настройках сети виртуальной машины установить тип подключения «Сетевой мост»
      3. Выполнить команду sudo ip link set dev eth0 mtu 1400, где вместо eth0 укажите название сетевого интерфейса виртуальной машины.
      4. Поочередно выполнить команды из инструкции для установки Docker: https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository
   2. Разработка API
      1. Разработать 2 отдельных api-сервиса со следующими методами

Для разработки рекомендуется использовать минимальные Api

ProductService:

GetProducts

GetProduct(id)

CreateProduct(Product)

IsAlive()

OrderService:

GetOrders

GetOrder(id)

CreateOrder(Order)

IsAlive()

* + 1. Разработать модели контексты БД для данных api-сервисов

Базы данных должны быть независимы, и таблицы между ними не должны иметь связи между друг другом.

В качестве БД использовать MySQL, а строку подключения получать из переменной окружения.

services.AddDbContext<ProductServiceDbContext>(options => options.UseMySQL(Environment.GetEnvironmentVariable("CONNECTION\_STRING")));

* + 1. Сгенерировать скрипт для создания бд на основе разработанных моделей данных.
    2. Сохранить скрипты создания и заполнения БД в папке решения указав ему имя {имя\_бд}\_init.sql.
  1. Разработка API-шлюза
     1. Добавить новый проект ApiGateway типа Web-API, удалить стандартный код.
     2. Установить в проекте nuget-пакет Ocelot.
     3. При помощи Ocelot добавить в Program.cs api-шлюз, который будет выполнять роль посредника для данных api-сервисов.

var builder = WebApplication.CreateBuilder(args);

builder.Configuration.AddJsonFile("ocelot.json", optional: false, reloadOnChange: true);

builder.Services.AddOcelot();

var app = builder.Build();

app.UseRouting();

app.MapControllers();

app.UseOcelot().Wait();

app.Run();

* 1. Указать настройки Api-шлюза в файле ocelot.json, вместо localhost укажите адрес виртуальной машины, добавить аналогичную настройку для сервиса заказов

{

"Routes": [

{

"DownstreamPathTemplate": "/api/products/{everything}",

"DownstreamHostAndPorts": [

{

"Host": "productservice",

"Port": 8080

}

],

"UpstreamPathTemplate": "/products/{everything}",

"UpstreamHttpMethod": [ "Get", "Post", "Put", "Delete" ]

}

//то же для сервиса заказов

],

"GlobalConfiguration": {

"BaseUrl": "http://localhost:5000"

}

}

* 1. Настроить решение для работы с контейнерами.
     1. В контекстном меню обозревателя решений во всех проектах выбрать опцию Добавить > Поддержка оркестратора контейнеров, где выберите Docker
     2. Получившаяся структура решения должна выглядеть следующим образом:

Решение/

ApiGateway/

ProductService/

OrderService/

docker-compose.yml

* + 1. В файл docker-compose вписать конфигурацию для развертывания разработанной системы. В листинге представлена часть конфигурации для ProductService и ApiGateway. Самостоятельно добавьте сервис заказов в настройки файла.

services:

apigateway:

container\_name: "apigateway"

image: ${DOCKER\_REGISTRY-}apigateway

build:

context: .

dockerfile: ApiGateway/Dockerfile

environment:

- ASPNETCORE\_ENVIRONMENT=Development

ports:

- "5000:8080"

depends\_on:

- productservice

links:

- productservice

networks:

- net

productservice:

container\_name: "productservice"

image: ${DOCKER\_REGISTRY-}productservice

build:

context: .

dockerfile: ProductService/Dockerfile

environment:

- ASPNETCORE\_ENVIRONMENT=Development

- CONNECTION\_STRING=Server=productdb;Database=Products;User=root;Password=root;Port=3306

depends\_on:

- productdb

links:

- productdb

networks:

- net

productdb:

container\_name: "productdb"

image: mysql:latest

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: Products

volumes:

- productdb:/var/lib/mysql

- ./products\_init.sql:/docker-entrypoint-initdb.d/products\_init.sql:ro

networks:

- net

volumes:

productdb:

networks:

net:

* + 1. В строке ./products\_init.sql:/docker-entrypoint-initdb.d/products\_init.sql:ro укажите имя файла инициализации БД, который создали вы.
    2. В Docker-файле ApiGateway укажите EXPOSE 5000 вместо 8080
    3. Подключиться по ssh к виртуальной машине c Docker и отправить на нее проект (при помощи git или scp).
    4. Запустить docker-compose (sudo docker compose up -d) и проверить корректность работы API по адресу http://{ip-адрес ВМ}:5000/products/.

1. Порядок выполнения работы
   1. Повторить теоретический материал п. 3.1;
   2. Выполнить задания 5.1-5.4
   3. Ответить на контрольные вопросы п. 8;
   4. Заполнить отчет п. 7.
2. Содержание отчета
   1. Титульный лист;
   2. Цель работы;
   3. Ответы на контрольные вопросы п. 6.3;
   4. Вывод по проделанной работе.
3. Контрольные вопросы
   1. Какие преимущества и недостатки существуют у микросервисной архитектуры