

ГАЙД ПО РАЗРАБОТКЕ МИКРОСЕРВИСА

Стек: Python, FastAPI, Celery, Redis, RabbitMQ, Git, Docker

1. Для кого и для чего этот гайд?

Большинство из нас писали в резюме: «Написал микросервис по...», но все ли в действительности понимают, что такое микросервис, как он взаимодействует с другим микросервисом, какие задачи может или не может решать?

В данном гайде вы найдете пример разработки микросервиса, который, надеемся, поможет вам как при подготовке к собеседованию, так и на будущей работе.

2. К какому проекту можно приурочить данный микросервис?

В гайде вы найдете пример небольшого микросервиса, который может внедриться в разные проекты, например:

1. Интернет-магазин, где микросервис будет обрабатывать заказы;
2. Корпоративный портал, где микросервис будет забирать задачи со стороннего сервиса и производить с ними дальнейшие действия;
3. Веб-приложение, где микросервис будет выполнять долгие / ресурсоемкие задачи в фоновом режиме.

0 Как понять когда и какой микросервис вам нужен

Перед тем как начать писать код, необходимо понять какая будет архитектура, за что будет отвечать микросервис, как микросервисы будут общаться между собой и многое другое.

Ниже описаны основные шаги при разработке микросервисов:

ЭТАП 0. ПОДГОТОВИТЕЛЬНЫЙ

- Определить зону ответственности микросервисов
- Создать проект
- Набросать структуру папок
- Инициализировать git репозиторий
- Создать репозиторий на GitHub / GitLab
- Запустить туда проект

ЭТАП 1. ВЫБОР СТЕКА ТЕХНОЛОГИЙ И НАСТРОЙКА ВИРТУАЛЬНОГО ОКРУЖЕНИЯ

- Выбрать фреймворк, базу данных, брокер сообщений и тд.
- Создать виртуальное окружение
- Установить необходимые зависимости и перенести их в requirements.txt

0 Как понять когда и какой микросервис вам нужен

ЭТАП 2. НАПИСАНИЕ МИКРОСЕРВИСОВ И ОПРЕДЕЛЕНИЕ СВЯЗИ МЕЖДУ НИМИ (в нашем примере 2 микросервиса, на след. слайде описан их функционал)

- Определить необходимые эндпоинты первого микросервиса, подключиться к базе данных, создать html шаблон (опционально)
- Инициализировать celery, подключить для него брокер сообщений, определить задачи и периодические задачи (опционально)
- Определить как микросервисы будут общаться между собой (через API, HTTP, БД) и настроить эту связь

ЭТАП 3. ЗАПУСК И ПРОВЕРКА РАБОТЫ

- Запустить микросервисы, поднять выбранный брокер сообщений и проверить совместную работу

ЭТАП 4. КОНТЕЙНЕРИЗАЦИЯ

- Создать dockerfile или docker-compose.yml
- Протестировать собранные образы

ЭТАП 5. НАСТРОЙКА CI/CD

ЭТАП 6. НАСТРОЙКА МОНИТОРИНГА И ЛОГИРОВАНИЯ, ОПРЕДЕЛИТЬ СТРАТЕГИЮ РАЗВЕРТЫВАНИЯ И ЗАПУСК В ПРОДАКШН

1 Описание микросервиса и создание проекта

Гайд написан на примере микросервиса, который подключается к Базе Данных другого микросервиса, берет задачи в работу, имитирует обработку и меняет статус готовности задач в БД



2 Инициализация репозитория git

Для начала необходимо инициализировать локальный репозиторий для нашего будущего проекта

git init – инициализируем репозиторий

git add . – добавляем в индекс изменения

git commit -m "..." – коммитим изменения

Затем синхронизируем наш локальный репозиторий с репозиторием на GitHub

1. Создаем новый репозиторий на GitHub

2. Выполните команды:

git remote add origin <https://github.com/<ваш аккаунт>/<название вашего нового репозитория>.git>

git branch -M main

git push -u origin main

3 Разработка вспомогательного микросервиса

Вспомогательный микросервис представляет собой простые ручки на FastAPI для добавления / получения / удаления задач

Сервис подключен к Базе Данных SQLite

После добавления задачи направляются в очередь RabbitMQ

ШАГИ ПРИ РАЗРАБОТКЕ:

1. Создать виртуальное окружение
2. Импортировать необходимые модули, создать файл requirements.txt с помощью команды `pip freeze`
3. Подключиться к БД
4. Подключиться к RabbitMQ
5. Создать html шаблон для записи задач
6. Создать ручки добавления / получения / удаления задач
7. Объявить очередь для отправки задач

ЗАПУСК:

`python main.py`

4 Разработка основного микросервиса

Второй (основной) микросервис представляет собой программу, которая забирает задачи из очереди RabbitMQ и определяет категорию задачи, в случае обычной задачи – происходит обработка и обновления статуса в БД, в случае «Долгой задачи» - Celery забирает задачу для фоновой обработки, имитирует ее долгую обработку и обновляет статус в БД.

После выполнения задачи вторым микросервисом в первом микросервисе при получении задач можно увидеть измененные статусы

ШАГИ ПРИ РАЗРАБОТКЕ:

- 1-4. Аналогичны предыдущему слайду
5. Подключиться к Redis и Celery
6. Объявить чтение очереди задач RabbitMQ
7. Создать функцию обработки обычной задачи
8. Создать таску для фоновой обработки задач с помощью Celery

ЗАПУСК:

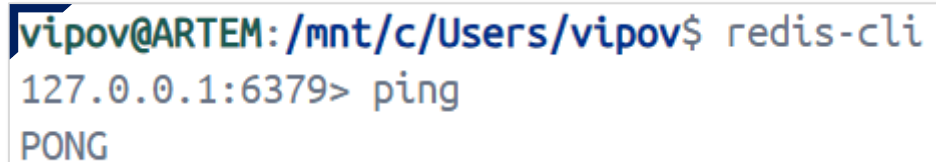
1. Запуск воркера: **celery -A main worker --pool threads -l info**
2. Запуск основной программы: **python main.py**

5 Поднятие Redis из Docker

Для реализации второго микросервиса необходимо поднять Redis из контейнера.

ШАГИ ПРИ ПОДНЯТИИ:

1. Скачать образ с Docker Hub – **docker pull redis:7.0-alpine**
2. Запустить контейнер на определенном порту – **docker run --name my-redis-container -d -p 6379:6379 redis**
3. Проверить работу контейнера с помощью команды – **redis-cli** (если все сделано верно при написании слова ping Redis ответит словом pong)



```
vipov@ARTEM: /mnt/c/Users/vipov$ redis-cli
127.0.0.1:6379> ping
PONG
```

A terminal window showing the execution of the Redis CLI. The prompt is 'vipov@ARTEM: /mnt/c/Users/vipov\$'. The command 'redis-cli' is entered. The prompt changes to '127.0.0.1:6379>'. The command 'ping' is entered, and the response 'PONG' is displayed. A blue line connects the third step of the list to this terminal window.

6 Поднятие RabbitMQ из Docker

Для реализации второго микросервиса необходимо также поднять RabbitMQ из контейнера.

ШАГИ ПРИ ПОДНЯТИИ:

1. Скачать образ с Docker Hub – **`docker pull rabbitmq:3-management`**
2. Запустить контейнер на определенном порту – **`docker run -d --name my-rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management`**
3. Проверить работу контейнера – **`http://localhost:15672`**

7 Контейнеризация

РАБОТА С DOCKERFILE

1. Создать два Dockerfile для наших микросервисов
2. Собрать Docker – образы: **docker build -t fastapi-app .** и **docker build -t celery-worker .**
3. Запустить наши контейнеры командами либо через Docker Desktop:

```
docker run -d \  
  --name celery-worker \  
  --link redis-server:redis \  
  -v /path/to/your/micro1/data:/app/external_db \  
  -e CELERY_BROKER_URL=redis://redis:6379/0 \  
  -e CELERY_RESULT_BACKEND=redis://redis:6379/0 \  
  -e DATABASE_URL=sqlite:///app/external_db/test.db \  
  celery-worker
```

МИКРОСЕРВИС С CELERY

```
docker run -d -p 8000:8000 --name fastapi-container fastapi-app
```

МИКРОСЕРВИС С CRUD ОПЕРАЦИЯМИ

4. Также запустить контейнеры с Redis и RabbitMQ
5. Проверить работу приложений – <http://localhost:8000>

РАБОТА С DOCKER-COMPOSE

1. Создать docker-compose.yml для одновременного запуска микросервисов и поднятия Redis с RabbitMQ (файл должен находиться на уровне директорий микросервисов)
2. Запустить docker-compose командой – **docker-compose up -d**

8

Репозиторий с готовым кодом

В репозитории находится готовый код для обоих микросервисов:

<https://github.com/Victory-hvop01/Micro>

P.S. Не судите строго код, это мой первый микросервис, готова принимать критику, комментарии и предложения для улучшения:))