НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №1

з предмету «Проектування розподілених систем»

Виконав:

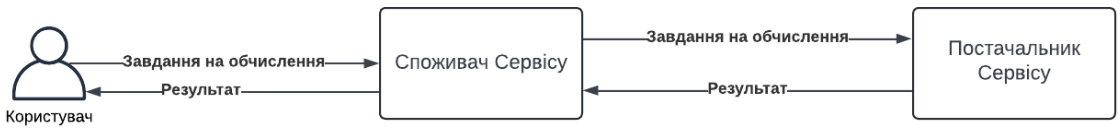
студент групи ІМ-31мн

Рекечинський Дмитро

Київ 2024

Завдання

- Реалізувати синхронну комунікація між 2ма сервісами. Споживач Сервісу генерує завдання на обчислення і чекає відповіді від Постачальник Сервісу.
- Постачальник Сервісу має підраховувати час обчислення і логувати його для подальшого аналізу
- Споживач Сервісу має підраховувати час виконання запиту і логувати його для подальшого аналізу
- Розгорнути Load Balancer перед Споживачем Сервісу і/або Постачальником сервісу
- *Опціонально: реалізувати протокол gRPC*
- *Опціонально: авторизація на рівні Споживача Сервісу*
- *Опціонально: авторизація на рівні Постачальника Сервісу*



Виконання завдання

Для виконання завдання було створено 4 типи сервісів (всього 6 інстансів):

- Споживач сервісу consumer-service
  - Інстанс consumer-service-1
  - Інстанс consumer-service-2
- Постачальник сервісу provider-service
  - Інстанс provider-service-1
  - Інстанс provider-service-2
- Load balancer для consumer-service (lb_consumer)
- Load balancer для provider-service (lb_provider)

Споживач сервісу має авторизацію, побудовану на основі JWT-токенів. Маршрут /generate_random_jwt створює токен для авторизації, який можна використати у майбутніх запитах:

```
app.get('/generate_random_jwt', (req, res) ⇒ {
    const token = generateRandomJwt();
    res.json({ token });
});
```

Потім цей токен використовується для авторизації при запиті на маршрут /generate_task, який приймає на вхід параметр URL input_data. Ці дані потім передаються на постачальника сервісу, а потім дані з нього повертаються користувачу:

```
app.get('/generate_task', verifyJwtToken, async (req, res) ⇒ {
    const { input_data } = req.query;

    const token = generateRandomJwt();
    const headers = {
        Authorization: `Bearer ${token}`,
        Accept: 'application/json',
        'Content-Type': 'application/json'
    };

    const startTime = Date.now();

    try {
        const response = await fetch(PROVIDER_URL, {
            method: 'POST',
            headers,
            body: JSON.stringify({ input_data })
        });

        const requestTime = (Date.now() - startTime) / 1000;
        const responseBody = await response.json();

        return res.json({
            response: responseBody,
            request_time: requestTime,
            consumer: NAME,
```

```
        });
    } catch (error) {
        console.dir(error, {depth: null});
        return res.status(500).json({ error: 'Failed to connect to
provider' });
    }
});
```

Там же провайдер очікує на запит, а коли його отримає, виконує доволі цікаві обчислення:

```
// Convert number to BigInt for big computations
number = BigInt(number);

// Do random calculations which take a plenty of time
for (let i = 0; i < 500_000; i++) {
    const randomNumber = BigInt(Math.round(Math.random() * 10**20));
    number = (number + randomNumber) ** 13n % (10n ** 40n);
}
```

Іншими словами, виконується 500 тисяч ітерацій, під час яких до числа додається випадкове 20-значне число, отриманий результат підноситься до 13 степеню, а потім із цього беруться останні 40 цифр. Сенсу від цього небагато, але це дає доволі відчутне навантаження.

Для балансування було застосовано Nginx, який підтримує цю функцію. Оскільки для конфігурування мережі контейнерів застосовується Docker Compose, на момент написання лабораторної роботи використання Nginx в якості Load balancer є рекомендованою опцією.

Конфігурація балансувальника споживача сервісу:

```
events {}

http {
    upstream consumer_service {
        server consumer-service-1:8000;
        server consumer-service-2:8000;
    }

    server {
        listen 80;

        location / {
            proxy_pass http://consumer_service;
        }
    }
}
```

Суть конфігурації така: як тільки прилітає запит на сервер, він перенаправляє на балансувальник consumer_service. Оскільки в конфігурації балансувальника не вказано нічого, крім переліку серверів, в якості алгоритму балансування береться простий round-robin, який по колу шукає вільний сервер. У ситуації, коли прилітає один запит на певний проміжок часу, round-robin передає запит то на перший сервер, то на другий, і так повторюється.

Сервер балансувальника слухає порт 80, який є стандартним для HTTP-з’єднань. Це зроблено для того, щоб запити до API виконувались без явного вказання номеру порту, і це дуже зручно.

Оскільки порт 80 зайнятий балансувальником споживача сервісу, балансувальнику постачальника сервісу не залишається нічого, окрім як взяти і послухати порт 5080:

```
events {}

http {
    upstream provider_service {
        server provider-service-1:9000;
        server provider-service-2:9000;
    }

    server {
        listen 5080;

        location / {
            proxy_pass http://provider_service;
        }
    }
}
```

Система в цьому плані доволі схожа із балансувальником споживача сервісу. Єдине, що сам сервіс споживача знаходиться у контейнері, і він не може робити ці дві речі:

- Звертатись до балансувальника постачальника сервісу lb_provider через localhost (а тільки через, власне, адресу http://lb_provider)
- Звертатись до балансувальника постачальника сервісу без явно вказаного порту (бо порт 80 уже зайнято). Втім, це не є проблемою, оскільки порт відомий, тому просто прописуємо http://lb_provider:5080 і все працює як слід

Демонстрація результатів:

```
~
❯ curl "http://localhost/generate_random_jwt" | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   201  100   201    0     0   8781      0 --:--:-- --:--:-- --:--:--  9136
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoiN2U5Zjc0MWYtZjM1Zi00YmY1LWExOWItMDZiMmM1ZGM3NmU5IiwiaWF0IjoxNzM0NTgxMTQ0LCJleHAiOjE3MzQ1ODE3NDR9.zChAUL8BVMfKRikEIlJU5pMIiiRFwKxf7_RXNdXGOJE"
}
```

```
~
❯ curl "http://localhost/generate_task?input_data=42" -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoiN2U5Zjc0MWYtZjM1Zi00YmY1LWExOWItMDZiMmM1ZGM3NmU5IiwiaWF0IjoxNzM0NTgxMTQ0LCJleHAiOjE3MzQ1ODE3NDR9.zChAUL8BVMfKRikEIlJU5pMIiiRFwKxf7_RXNdXGOJE" | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   172  100   172    0     0    225      0 --:--:-- --:--:-- --:--:--   225
{
  "response": {
    "result": "19711764481763478680787648613972589936 64",
    "computation_time": 0.715,
    "name": "provider-service_2"
  },
  "request_time": 0.752,
  "consumer": "consumer-service_1"
}
```

Повна версія коду проекту розміщена за веб-адресою:
https://github.com/rocket111185/distribution-systems/tree/release/lab1