Лабораторна робота №4.

Мета роботи: ознайомитись з гетерогенністю мікросервісної архітектури, яка дозволяє застосовувати свою технологію кожному сервісу.

Задача: на основі попередніх робіт розгорнути «багатомовний» додаток на локальному комп'ютері, протестувати його та дослідити принцип роботи з різними технологіями на рівні мікросервісів. Необхідно розгорнути кілька мікросервісів на різних технологіях і показати, що вони між собою взаємодіють.

Хід роботи

Було додано новий мікросервіс який написаний на JavaScript. Він має ендпоинт який отримує рядок, котрий потрібно записати в файл.

```
app.get('/writeToFile', (req, res) => {
 const { data } = req.query;

 if (!data) {
    return res.status(400).json({ error: 'use param data to send data'
});
 }
 const host = os.hostname();
 const content = `Host: ${host} data: ${data}\n`;
 fs.appendFile('logs.txt', content, (err) => {
   if (err) {
     return res.status(500).json({ error: 'Error.' });
   }

   return res.json({ message: 'Data was written.' });
 });
});
```

Було розширено Nginx балансувальник, щоб він також розкидав запити до логувальника.

```
http {
       server {
              listen 8080;
              location / {
                proxy_pass http://calculator:81;
              }
       }
       server {
              listen 3000;
              location / {
                proxy_pass http://logger:3000;
              }
       }
}
```

Тому цей мікросервіс піддається масштабуванні, і завдяки монтуванні файл через docker compose усі логи записуються в один файл.

```
  logger:
   build: logger
   expose:
     - "3000"
   volumes:
     - ./logger/log.txt:/server/logs.txt
   deploy:
     mode: replicated
     replicas: 2
```

Оновили мікросервіс calculator щоб він відправляв логи обрахунків:

```
@app.get("/calculate")
def get_records(expression: str = '5+5/5-3'):
  print(f"calculating {expression}")
  data = {
     "host": myhost,
     "result": f"{expression} = {eval(expression)}"
  }
        requests.get('http://nginx:3000/writeToFile',    params={'data':
f"calculator: {data['host']}, result: {data['result']}"})
  return data
```

Для тестування використаємо тестову програму, яка відправляє на головне API купу запитів.

```
(study) → lab_4 git:(soa_lab_3-5) ✗ python test.py
5+5-3 => {"host":"7401d211d4e8","result":"5+5-3 = 7"}
5+5-2 => {"host":"7401d211d4e8","result":"5+5-2 = 8"}
5+11 => {"host":"ccfbd2af8b79","result":"5+11 = 16"}
5+5 => {"host":"468d1f98b008","result":"5+5 = 10"}
5+5+3 => {"host":"7401d211d4e8","result":"5+5+3 = 13"}
5+5-3 => {"host":"ccfbd2af8b79","result":"5+5-3 = 7"}
5+5%3 => {"host":"ccfbd2af8b79","result":"5+5%3 = 7"}
5+5-10 => {"host":"468d1f98b008","result":"5+5-10 = 0"}
5+5/5 => {"host":"7401d211d4e8","result":"5+5/5 = 6.0"}
(5+5)*8 => {"host":"ccfbd2af8b79","result":"(5+5)*8 = 80"}
5+5//5 => {"host":"468d1f98b008","result":"5+5//5 = 6"}
5+5-5 => {"host":"468d1f98b008","result":"5+5-5 = 5"}
```

І перевіримо файл логування.

lab_4 > logger > log.txt

```
1  Host: ce1d2d1724be data: calculator: 7401d211d4e8, result: 5+5-2 = 8
2  Host: 088c0f64e8e1 data: calculator: 468d1f98b008, result: 5+5+3 = 13
3  Host: ce1d2d1724be data: calculator: ccfbd2af8b79, result: 5+11 = 16
4  Host: 088c0f64e8e1 data: calculator: 468d1f98b008, result: 5+5-5 = 5
5  Host: ce1d2d1724be data: calculator: 7401d211d4e8, result: 5+5/5 = 6.0
6  Host: ce1d2d1724be data: calculator: 7401d211d4e8, result: 5+5-10 = 0
7  Host: ce1d2d1724be data: calculator: ccfbd2af8b79, result: 5+5-3 = 7
8  Host: 088c0f64e8e1 data: calculator: 468d1f98b008, result: 5+5 = 10
9  Host: 088c0f64e8e1 data: calculator: ccfbd2af8b79, result: 5+5//5 = 6
10 Host: 088c0f64e8e1 data: calculator: ccfbd2af8b79, result: (5+5)*8 = 80
11 Host: 088c0f64e8e1 data: calculator: 468d1f98b008, result: 5+5%3 = 7
12 Host: ce1d2d1724be data: calculator: 7401d211d4e8, result: 5+5-3 = 7
13
```

Як бачимо, різні контейнери записували логи у спільний файл.

Як зазначалося раніше, мікросервіси здатні на масштабування
docker-compose up --build --scale calculator=3 --scale logger=3

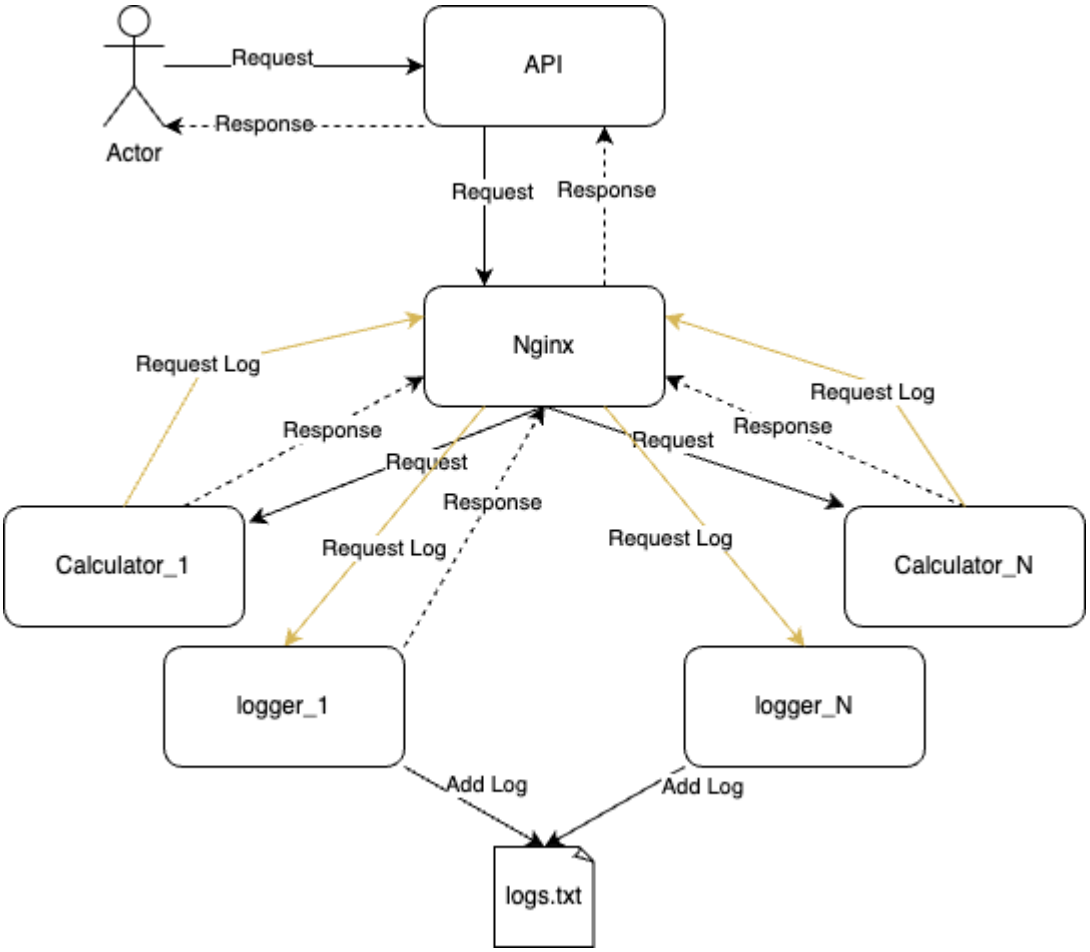
```
✔ Container lab_4-logger-3      Created
✔ Container lab_4-logger-1      Recreated
✔ Container lab_4-logger-2      Recreated
✔ Container lab_4-calculator-3  Running
✔ Container lab_4-calculator-2  Running
✔ Container lab_4-calculator-1  Running
✔ Container lab_4-nginx-1       Running
✔ Container lab_4-api-1         Running
```

Висновки: На лабораторній роботі було розширено додаток з попередньої роботи – додано новий мікросервіс, який використовує іншу технологію запуска Node.js та мову програмування JavaScript. Даний мікросервіс має на меті, з отриманих запитів записувати дані в файл логування.

Було оновлено балансувальник та інший мікросервіс, для відправки логування подій. Дану логіку було вдало протестовано.

Увесь код можна знайти на [репозиторії](#).

Оновлена діаграма архітектури:



Відповіді на контрольні питання:
1. Принципи і протоколи комунікації між сервісами.

Принципи:

Комунікація через мережу: Сервіси зазвичай взаємодіють один з одним через мережу. Це може бути HTTP-запити, протоколи обміну повідомленнями, такі як AMQP або MQTT, або інші мережеві технології.

Балансування запитів між копіями мікросервісу: трасування запитів між копіями сервісу задля поділу навантаження.

Протоколи комунікації:

HTTP/REST: Засновані на HTTP протоколі та архітектурному стилі REST. Кожен сервіс використовує URL-адреси для доступу до ресурсів та HTTP-методи (GET, POST, PUT, DELETE) для взаємодії з ними.

Message Queues: Використання черг повідомлень для асинхронної комунікації між сервісами. Приклади Apache Kafka та інші.

GraphQL: Мова запитів та середовище виконання для взаємодії з API. Забезпечує гнучкий та ефективний спосіб отримання та відправлення даних між сервісами.

WebSocket: Протокол забезпечення зв'язку повно-дуплексного каналу між клієнтом і сервером. Використовується для реального часу та інших випадків, де необхідна багаторазова взаємодія. WebSocket базується на TCP-протоколі.

2. Яким чином сервіси знаходять екземпляри один одного?

Кожен мікросервіс отримує свій унікальний ідентифікатор та належність до мережі, де вже через DNS можна до нього відправити запит.

Завдяки балансувальнику навантаження потік запитів рівномірно розподіляється між мікросервісами, але якщо нагрузка збільшується певної межі, то можливе Автоматичне масштабування – підняття додаткових контейнерів.

Централізований реєстр: Великі системи часто використовують централізований реєстр, де кожен сервіс реєструє свою наявність та інформацію про свою роботу.

Контейнерні оркестратори: У віртуалізованих середовищах, таких як Docker, контейнерні оркестратори, такі як Kubernetes або Docker Swarm, можуть взяти на себе відповідальність за виявлення та управління екземплярами сервісів.

3. (*) Механізм відслідковування тайм-ауту при ланцюжковому (послідовному) виклику мікросервісів.

Тайм-аути в HTTP запитах: У випадку використання HTTP запитів між мікросервісами, встановлення тайм-аутів на рівні HTTP може бути ефективним рішенням. Багато HTTP-клієнтів дозволяють встановлювати тайм-аути для запитів. Це дозволяє зупиняти виклик, якщо немає відповіді протягом визначеного періоду.

Circuit Breaker Pattern: Circuit Breaker - це патерн проектування, який дозволяє виявити та управляти помилками в розподіленому середовищі. Він може автоматично відключати мікросервіс від ланцюжка викликів, якщо він перевищує встановлену кількість помилок або неправильно виконується, забезпечуючи тим самим зменшення тайм-аутів та швидше відновлення системи.