

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Розрахункова робота

з предмету «Проектування розподілених систем»

Виконав:

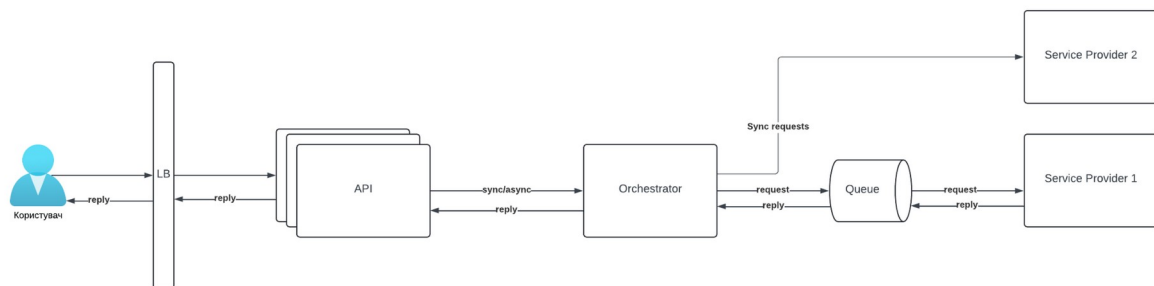
студент групи ІМ-31мн

Рекечинський Дмитро

Київ 2024

Завдання

- Реалізувати патерн оркестратор який буде керувати процесом розрахунку мат.моделі
- Побудувати математичну модель системи масового обслуговування (стая дронів) і розрахувати вплив кількості Постачальників сервісу на швидкість обробки завдань
- Зробити опис системи



Виконання завдання

Проект було створено з реалізацію паттерну Enterprise на основі моделі Event Sourcing.

Для виконання завдання було створено 6 сервісів:

- Load balancer load-balancer
- API api-service (3 реплікації)
- Оркестратор orchestrator
- Постачальник сервісу 1 queue-provider
- Постачальник сервісу 2 sync-provider
- Брокер повідомлень RabbitMQ (rabbitmq)

Сервіс load-balancer насправді є вхідною точкою до сервісу api-service. Конфігурація NGINX виглядає таким чином:

```
events {}

http {
    server {
        listen 80;

        location / {
            proxy_pass http://api-service:8000;
        }
    }
}
```

Якщо так, то за рахунок чого виконується балансування навантаження?

Docker Compose, починаючи із червня 2023 року підтримує налаштування реплікацій безпосередньо у файлі конфігурації `docker-compose.yml`.
Налаштування `api-service` виглядає таким чином:

```
api-service:
  build:
    context: ./api
  deploy:
    mode: replicated
    replicas: 3
  environment:
    ORCHESTRATOR_URL: "http://orchestrator:9000/calculate"
    PORT: 8000
  networks:
    - my-network
```

Найбільш цікавою частиною є атрибут `deploy`. Саме в ньому визначається, в якому режимі запускається сервіс. В даному випадку встановлено режим «реплікований», і цих реплікацій — три. За умовчуванням, за реплікації та їх балансування відповідає Docker Swarm.

Якщо так, навіщо тоді вхідна точка у вигляді `load-balancer`?

Суть проста: для того, щоб надати доступ до сервісу `api-service`, треба надати доступ до порту, від контейнера до хоста. Втім, коли цей доступ надається для однієї реплікації, інша реплікація вже не може використати цей порт. Сервіс `load-balancer` допомагає уникнути цієї проблеми, оскільки він і так має доступ до сервісу `api-service`.

Оркестратор `orchestrator` виконує запити до `queue-provider` та `sync-provider`, тому не дивно, що в `docker-compose.yml` прописано залежність оркестратора від цих сервісів.

`sync-provider` запускається незалежно, в той час, як `queue-provider` залежить від того, чи запустився сервіс `rabbitmq`, чи ні. Для того, щоб перевірити, чи сервіс не просто запущений, а ще й готовий до підключення, використовується атрибут `healthcheck`:

```
healthcheck:
  test: rabbitmq-diagnostics check_port_connectivity
  interval: 10s
  timeout: 5s
  retries: 10
  start_period: 5s
```

Втім, цього лише недостатньо, щоб залежний сервіс спрацьовував лише тоді, коли сервіс `rabbitmq` готовий до підключень. За умовчуванням, `depends_on` вказує на те, що залежний контейнер запуститься тоді, коли потрібні контейнери просто запустяться, незважаючи на статус `healthcheck`.

Але і це можна виправити, якщо перетворити `depends_on` зі списку на об'єкт:

```
depends_on:
  rabbitmq:
    condition: service_healthy
```

Після цього, сервіс queue-provider буде запущений тільки тоді, коли healthcheck rabbitmq поверне успішний статус. Це, в свою чергу, впливає на сервіс orchestrator, який також залежить від rabbitmq, але оскільки orchestrator залежить і від rabbitmq, і від queue-provider, достатньо для orchestrator бути залежним від queue-provider.

Щодо суті математичної моделі, у цій системі реалізовано розв'язання множини систем лінійних алгебраїчних рівнянь методом Гауса. Приклад тіла запиту:

```
{
  "systems": [
    {
      "coefficients": [
        [10, -4, -3, 9],
        [-2, -4, -8, 2],
        [-7, 1, -8, 5],
        [-10, -4, -5, -6]
      ],
      "values": [21, -17, 17, 45]
    }
  ],
  "priority": 2,
  "format": "full"
}
```

- systems — це масив об'єктів СЛАР
 - coefficients — масив коефіцієнтів залежних змінних

- values — масив результатів рівнянь
- priority — пріоритет виконання в черзі RabbitMQ
- format — формат виводу результатів

Демонстрація результатів

```
~/Documents/kpi/master/3sem/distribution-systems/rgr · (release±)
> curl -X POST "http://localhost/generate_task" -H "Content-Type: application/json" -d @request-sample-1.json | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100  9857  100  3397  100  6460    509    969   0:00:06  0:00:06 --:--:--  761
{
  "response": {
    "results": {
      "results": [
        {
          "result": [
            -5.303104634266193,
            -6.420900628526437,
            8.80801128450083,
            7.7136685261740645,
            0.7087752172673821,
            -7.791784097649051,
            17.47008121068695,
            -12.098784491992118,
            -9.890520857753017,
            -2.2292975777640867
          ]
        }
      ]
    }
  }
}
```

Рис. 1.1 — Запит 1 (пріотритет 2, виконувався першим)

```

        -11.502666666666666,
        4.644296296296297,
        7.089185185185187
    ],
    "computationTime": 0.159,
    "id": 18
  },
  {
    "result": [
      -17.016528925619824,
      -22.165289256198335,
      -1.9504132231404947,
      0.35537190082644177
    ],
    "computationTime": 0.159,
    "id": 19
  }
]
},
"calculationTime": 6.552
},
"requestTime": 6.641
}

```

Рис. 1.2 — Запит 1 (пріоритет 2, виконувався першим)

```

~/Documents/kpi/master/3sem/distribution-systems/rgr - (release±)
> curl -X POST "http://localhost/generate_task" -H "Content-Type: application/js
on" -d @request-sample-2.json | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100  9862  100  3402  100  6460    574   1090   0:00:05   0:00:05   -:--:--   721
{
  "response": {
    "results": {
      "results": [
        {
          "result": [
            0.24444444444444535,
            -10.844444444444443,
            -64.71111111111111,
            -21.088888888888889
          ],
          "computationTime": 0.167,
          "id": 0
        },
        {
          "result": [
            7.096958174904946,
            0.4017110066666667
          ],
          "computationTime": 0.159,
          "id": 1
        }
      ]
    }
  }
}

```

Рис 2.1 — Запит 2 (пріоритет 1, виконувався другим)


```

    },
    {
      "result": [
        2.9288015901747624,
        21.30794709842615,
        6.163729608296911,
        1.2059664966631907,
        -0.7383345962745986,
        -2.9368909783777823,
        1.6977765525912343,
        -4.878931991298214,
        6.310516492340378,
        12.746753365079114
      ],
      "computationTime": 0.155,
      "id": 19
    }
  ]
},
"calculationTime": 5.865
},
"requestTime": 5.903
}

```

Рис 2.2 — Запит 2 (пріотритет 1, виконувався другим)

Повна версія коду проекту розміщена за веб-адресою:

<https://github.com/rocket111185/distribution-systems/tree/release/rgr>