Bazy danych NoSQL

laboratorium

Kamil Staśko

NIESTACJONARNE, GRUPA: L1

Opracowanie laboratorium (Mongo, MariaDB)

Uniwersytet Pedagogiczny
im. Komisji Edukacji Narodowej
w Krakowie

Wstęp

Laboratorium zostało przeprowadzone z wykorzystaniem:

- Środowisko: VMware® Workstation 14 Player
- Wersja środowiska: 14.1.8 build-14921873
- System operacyjny: Debian GNU/Linux 10 (buster)
- Procesor: Intel(R) Core(TM) i7-6700HQ CPU @2.60GHz 2.60GHz
- Architektura: 64-bit
- Pamięć RAM: 2GB

Rozdział 1

Komunikacja MariaDB - Mongo

1.1 Kody plików

- lab1.yml - dla docker-compose - ustawienie środowiska, obrazy (mongo, mariadb, python)

```
version: '3'
networks:
    cluster:
        external:
            name: cluster
```

Rys. 1.1: Stworzenie klastra

```
services:
    mongo:
        container_name: "mongo"
        networks:
          cluster:
            ipv4_address: 10.100.100.2
            aliases:
              - mongo
        image: mongo
```

Rys. 1.2: Konfiguracja Mongo

```
  mariadb:
      container_name: "mariadb"
      networks:
        cluster:
          ipv4_address: 10.100.100.3
          aliases:
            - mariadb
      image: mariadb
      ports:
        - "3306:3306"
      volumes:
          - /root/ZADANIE:/DATA
      restart: always
      environment:
          MYSQL_ROOT_PASSWORD: root
          MYSQL_USER: root
          MYSQL_PASSWORD: root
          MYSQL_DATABASE: LAB_KAMIL
      healthcheck:
        test: ["CMD", "mysql", "--user=root", "-proot", "-e
          ↪ show databases"]
        interval: 2s
        timeout: 1s
        retries: 20
```

Rys. 1.3: Konfiguracja MariaDB

```
    python:
        container_name: "python"
        networks:
          cluster:
            ipv4_address: 10.100.100.4
            aliases:
              - python
        image: deb_py
        volumes:
            - /root/ZADANIE:/PY3
        command: tail -f /etc/passwd > /dev/null
```

Rys. 1.4: Konfiguracja python3

- lab1.sql - plik odtwarzający tabele w bazie maria (10000 rekordów)

```
USE LAB_KAMIL;

CREATE TABLE TABLE_KAMIL(
        id INT NOT NULL AUTO_INCREMENT,
        val0 INT NOT NULL,
        val1 INT NOT NULL,
        val2 INT NOT NULL,
        val3 INT NOT NULL,
        val4 INT NOT NULL,
        val5 INT NOT NULL,
        val6 INT NOT NULL,
        val7 INT NOT NULL,
        val8 INT NOT NULL,
        val9 INT NOT NULL,
        PRIMARY KEY (id)
);
```

Rys. 1.5: Stworzenie tabeli

```
drop procedure if exists myLoop;
DELIMITER //
CREATE PROCEDURE myLoop()
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE j INT DEFAULT 1;
WHILE (i <= 10000) DO
    INSERT INTO TABLE_KAMIL
        (val0, val1, val2, val3, val4, val5, val6, val7, val8,
            ↪ val9)
        VALUES
        (10*j, 10*j+1, 10*j+2, 10*j+3, 10*j+4, 10*j+5, 10*j+6,
            ↪ 10*j+7, 10*j+8, 10*j+9);
        SET i = i+1;
        SET j = j+1;
END WHILE;
END;
//

CALL myLoop();
```

Rys. 1.6: Procedura wprowadzenia danych

- lab1.py - plik przepisujący tabele z mariadb do kolekcji mongo

```
#!/usr/bin/python
import time
import mysql.connector
from pymongo import MongoClient

conn = mysql.connector.connect(user='root', password='root',
    ↪ host='10.100.100.3', database='LAB_KAMIL')
cur = conn.cursor()
cur.execute('SELECT id, val0, val1, val2, val3, val4, val5, val6
    ↪ , val7, val8, val9 FROM TABLE_KAMIL')
```

Rys. 1.7: Połączenie z MariaDB

```
client = MongoClient('10.100.100.2:27017')
baza = client['LAB_KAMIL']
kolekcja = baza['KOLEKCJA_KAMIL']
```

Rys. 1.8: Połączenie z Mongo

```
start_time = time.time()

for (id, val0, val1, val2, val3, val4, val5, val6, val7, val8,
    ↪ val9) in cur:
        kolekcja.insert_one({'_id': id, 'val0': val0, 'val1':
            ↪ val1, 'val2': val2, 'val3': val3, 'val4': val4, '
            ↪ val5': val5, 'val6': val6, 'val7': val7, 'val8':
            ↪ val8, 'val9': val9})

end_time = time.time() - start_time

print(end_time)
```

Rys. 1.9: Przepisanie danych oraz obliczenie czasu procesu

- lab1.sh - skrypt bash automatyzujący cały proces

```
#!/bin/sh

cd /root/
mkdir ZADANIE
cd ./ZADANIE
```

Rys. 1.10: Stworzenie katalogu z danymi

```
echo "version: '3'
networks:
    cluster:
        external:
            name: cluster
services:
    mongo:
        container_name: \"mongo\"
        networks:
          cluster:
            ipv4_address: 10.100.100.2
            aliases:
              - mongo
        image: mongo
    mariadb:
        container_name: \"mariadb\"
        networks:
          cluster:
            ipv4_address: 10.100.100.3
            aliases:
              - mariadb
        image: mariadb
        ports:
          - \"3306:3306\"
        volumes:
            - /root/ZADANIE:/DATA
        restart: always
        environment:
            MYSQL_ROOT_PASSWORD: root
            MYSQL_USER: root
            MYSQL_PASSWORD: root
            MYSQL_DATABASE: LAB_KAMIL
        healthcheck:
          test: [\"CMD\", \"mysql\", \"--user=root\", \"-proot\",
             ↪  \"-e show databases\"]
          interval: 2s
```

```
36          timeout: 1s
37          retries: 20
38      python:
39          container_name: \"python\"
40          networks:
41            cluster:
42              ipv4_address: 10.100.100.4
43              aliases:
44                - python
45          image: deb_py
46          volumes:
47              - /root/ZADANIE:/PY3
48          command: tail -f /etc/passwd > /dev/null
49  " > lab1.yml
```

Rys. 1.11: Stworzenie pliku lab1.yml

```
echo "USE LAB_KAMIL;

CREATE TABLE TABLE_KAMIL(
        id INT NOT NULL AUTO_INCREMENT,
        val0 INT NOT NULL,
        val1 INT NOT NULL,
        val2 INT NOT NULL,
        val3 INT NOT NULL,
        val4 INT NOT NULL,
        val5 INT NOT NULL,
        val6 INT NOT NULL,
        val7 INT NOT NULL,
        val8 INT NOT NULL,
        val9 INT NOT NULL,
        PRIMARY KEY (id)
);

drop procedure if exists myLoop;
DELIMITER //
CREATE PROCEDURE myLoop()
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE j INT DEFAULT 1;
WHILE (i <= 10000) DO
    INSERT INTO TABLE_KAMIL
        (val0, val1, val2, val3, val4, val5, val6, val7, val8,
            ↪ val9)
        VALUES
        (10*j, 10*j+1, 10*j+2, 10*j+3, 10*j+4, 10*j+5, 10*j+6,
            ↪ 10*j+7, 10*j+8, 10*j+9);
        SET i = i+1;
        SET j = j+1;
END WHILE;
END;
//

```

```
CALL myLoop();
" > lab1.sql
```

Rys. 1.12: Stworzenie pliku lab1.sql

```
echo "#!/usr/bin/python
import time
import mysql.connector
from pymongo import MongoClient

conn = mysql.connector.connect(user='root', password='root',
    ↪ host='10.100.100.3', database='LAB_KAMIL')
cur = conn.cursor()
cur.execute('SELECT id, val0, val1, val2, val3, val4, val5, val6
    ↪ , val7, val8, val9 FROM TABLE_KAMIL')

client = MongoClient('10.100.100.2:27017')
baza = client['LAB_KAMIL']
kolekcja = baza['KOLEKCJA_KAMIL']

start_time = time.time()

for (id, val0, val1, val2, val3, val4, val5, val6, val7, val8,
    ↪ val9) in cur:
        kolekcja.insert_one({'_id': id, 'val0': val0, 'val1':
            ↪ val1, 'val2': val2, 'val3': val3, 'val4': val4, '
            ↪ val5': val5, 'val6': val6, 'val7': val7, 'val8':
            ↪ val8, 'val9': val9})

end_time = time.time() - start_time

print(end_time)
" > lab1.py
```

Rys. 1.13: Stworzenie pliku lab1.py

```
1  docker network create --subnet=10.100.100.0/24 cluster
2  wait $!
3  docker-compose -f lab1.yml up -d
4  wait $!
5
6  value=$(docker inspect --format='{{.State.Health.Status}}'
       ↪ mariadb)
7  while [ "$value" != "healthy" ]
8  do
9    value=$(docker inspect --format='{{.State.Health.Status}}'
         ↪ mariadb)
10 done
11
12 docker exec -it mariadb bash -c "mysql -u root -proot -e 'source
       ↪  /DATA/lab1.sql'"
13 wait $!
14
15 docker exec -it python bash -c "virtualenv --no-site-packages --
       ↪ python=python3 PY3"
16 wait $!
```

Rys. 1.14: Przygotowanie środowiska

```
docker exec -it python bash -c "cd PY3/; source bin/activate;
    ↪ pip install pymongo; pip install mysql-connector; python
    ↪ lab1.py > result.txt"
wait $!

docker-compose -f lab1.yml down
wait $!
docker network rm cluster
wait $!

result=$(cat result.txt)
echo ""
echo ""
echo "KONIEC PROGRAMU"
echo ""
echo "CZAS KOPIOWANIA [10000 REKORDÓW] MARIADB->MONGO WYNIÓSŁ:
    ↪ $result s"
echo ""
```

Rys. 1.15: Przygotowanie python3 oraz wykonanie testu

1.2 Wykonanie testów

Pomiar czasu pobrania rekordów z MariaDB oraz wpisania ich do kolekcji w Mongo.

Tabela 1.1: Testy komunikacji MariaDB - Mongo

LICZBA REKORDÓW	CZAS [s]
10	0.0170135498046875
100	0.07358860969543457
1000	0.6900181770324707
10000	9.058728456497192
100000	65.4701337814331

Rozdział 2

Komunikacja Mongo - Mongo

2.1 Kody plików

- lab2.yml - dla docker-compose - ustawienie środowiska, obrazy (mongo1, mongo2, python)

```
version: '3'
networks:
    cluster:
        external:
            name: cluster
```

Rys. 2.1: Stworzenie klastra

```
services:
    mongo1:
        container_name: "mongo1"
        networks:
          cluster:
            ipv4_address: 10.100.100.2
            aliases:
              - mongo1
        image: mongo
```

Rys. 2.2: Konfiguracja Mongo1

```
    mongo2:
        container_name: "mongo2"
        networks:
          cluster:
            ipv4_address: 10.100.100.3
            aliases:
              - mongo2
        image: mongo
```

Rys. 2.3: Konfiguracja Mongo2

```
    python:
        container_name: "python"
        networks:
          cluster:
            ipv4_address: 10.100.100.4
            aliases:
              - python
        image: deb_py
        volumes:
            - /root/ZADANIE:/PY3
        command: tail -f /etc/passwd > /dev/null
```

Rys. 2.4: Konfiguracja python3

- lab2create.py - plik przygotowujący kolekcję danych w mongo1

```
#!/usr/bin/python
from pymongo import MongoClient

client = MongoClient('10.100.100.2:27017')
baza = client['LAB_KAMIL']
kolekcja = baza['KOLEKCJA_KAMIL']
```

Rys. 2.5: Połączenie z Mongo1

```
i = 1
j = 1

while i <= 1000:
        kolekcja.insert_one({'_id': i, 'val0': 10*j, 'val1': 10*j
            ↪ +1, 'val2': 10*j+2, 'val3': 10*j+3, 'val4': 10*j
            ↪ +4, 'val5': 10*j+5, 'val6': 10*j+6, 'val7': 10*j
            ↪ +7, 'val8': 10*j+8, 'val9': 10*j+9})
        i += 1
        j += 1
```

Rys. 2.6: Wypełnienie danymi kolekcji Mongo1

- lab2test.py - plik wykonujący test podstawowych operacji (find, read, write)

```
#!/usr/bin/python
from pymongo import MongoClient
import random
import time

client1 = MongoClient('10.100.100.2:27017')
baza1 = client1['LAB_KAMIL']
kolekcja1 = baza1['KOLEKCJA_KAMIL']
```

Rys. 2.7: Połączenie z Mongo1

```
client2 = MongoClient('10.100.100.3:27017')
baza2 = client2['LAB_KAMIL']
kolekcja2 = baza2['KOLEKCJA_KAMIL']
```

Rys. 2.8: Połączenie z Mongo2

```
start_time = time.time()

while kolekcja1.count_documents({}) != kolekcja2.count_documents
    ↪ ({}):
        val = 0

        while val == 0:
                val = random.randint(1, kolekcja1.count_documents
                    ↪ ({}))

                if kolekcja2.count_documents({"_id": val}) > 0:
                        val = 0

        kolekcja2.insert_one(kolekcja1.find_one({'_id': val}))

end_time = time.time() - start_time

print(end_time)
```

Rys. 2.9: Wykonanie testu

2.2 Wykonanie testów

Pomiar czasu wykonania następującego cyklu zapytań (liczba zależna od liczby rekordów w Mongo1):

- Mongo1 - count_document (określenie liczby rekordów)
- Mongo2 - count_document (określenie liczby rekordów)
- wykonanie cyklu zapytań (liczba zależna od wylosowania id, które nie znajduje się w Mongo2)
  - Mongo1 - count_document (w celu wylosowania id)
  - Mongo2 - count_document (w celu sprawdzenia czy istnieje element o danym id)
- Mongo1 - find_one (pobranie rekordu o podanym id)
- Mongo2 - insert_one (wstawienie pobranego rekordu z Mongo1)

Tabela 2.1: Testy wydajności Mongo - Mongo

LICZBA REKORDÓW	CZAS [s]
10	0.08119702339172363
100	0.9195506572723389
1000	18.200843334197998
10000	658.4297559261322

Listings

Spis tabel