Проектування розподілених систем
Task 2 - Розгортання і робота з distributed in-memory data structures на основі Hazelcast: Distributed Map
Мартиненко Денис ФБ-42мп

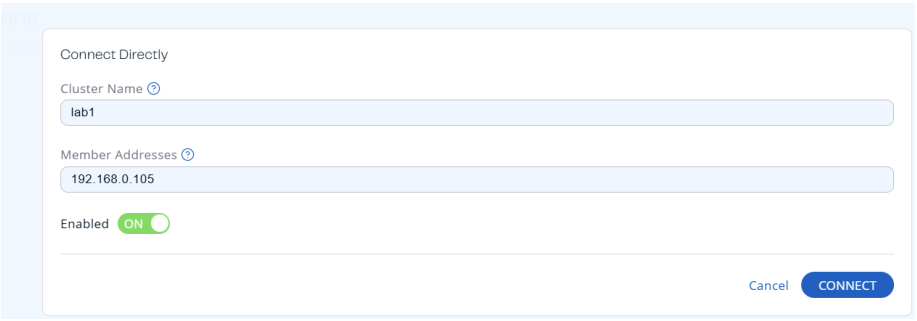
1. **Встановити і налаштувати Hazelcast:**

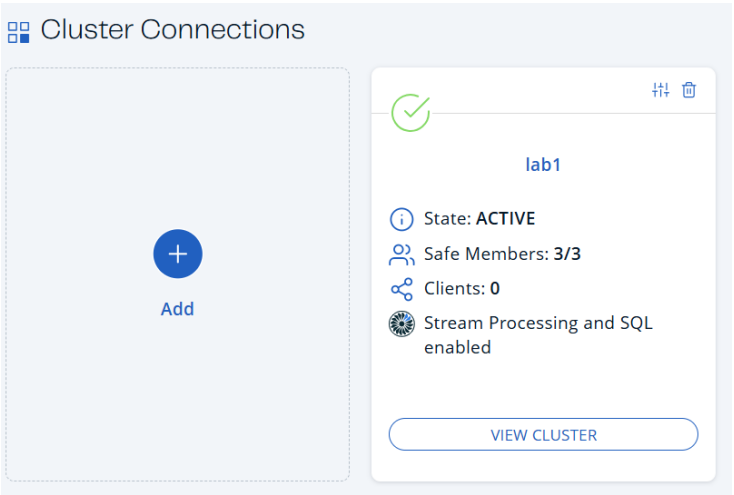
pip install hazelcast-python-client
docker network create -d bridge hazelcast-network
docker-compose up

- Було зроблено у мене на хості ще минулого семестру

2. **Сконфігурувати і запустити 3 ноди (інстанси) об'єднані в кластер або як частину Java-застосування, або як окремі застосування**

http://localhost:8080/cluster-connections

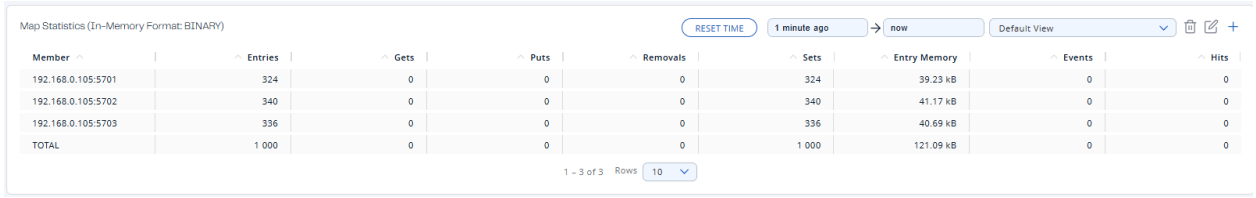




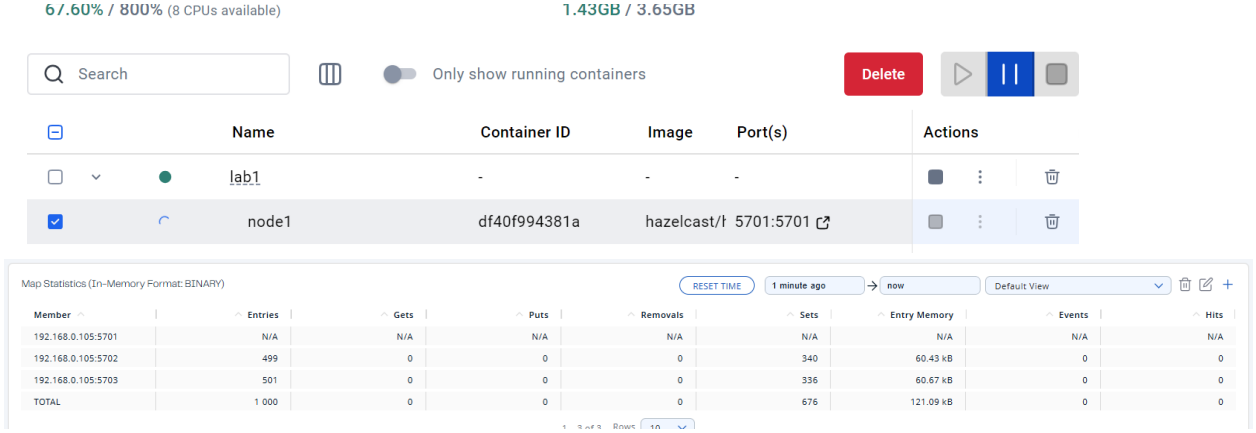
3. Продемонструйте роботу Distributed Map

- використовуючи API створіть Distributed Map
- запишіть в неї 1000 значень з ключем від 0 до 1к

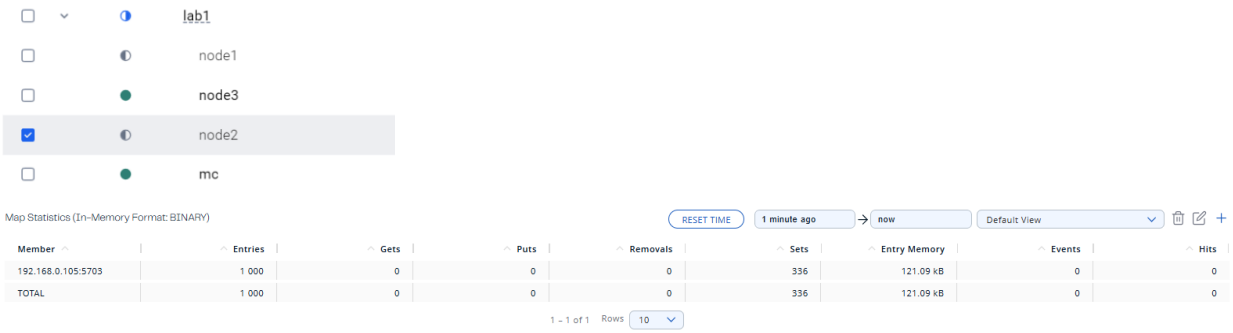
- за допомогою Management Center подивіться на розподіл значень по нодах



- подивитись як зміниться розподіл даних по нодах:
  - якщо відключити одну ноду



- відключити дві ноди



- Чи буде втрата даних?

  Втрати даних не було.

4. Продемонструйте роботу Distributed Map without locks використовуючи 3 клієнта, на кожному з них одночасно запустіть інкремент значення для одного й того самого ключа в циклі на 10К ітерацій:

```
import multiprocessing
import hazelcast
import time

def increment_without_lock(iterations=10000, batch_size=10):
    """
    Кожен процес локально інкрементує значення batch_size разів,
    після чого робить один мережевий запит get/put.
    """
    client = hazelcast.HazelcastClient()
    distributed_map = client.get_map("distributed-map").blocking()

    # Ініціалізація ключа "key", якщо він ще не встановлений
    if distributed_map.get("key") is None:
        distributed_map.put("key", 0)

    local_increments = 0
    for i in range(iterations):
        local_increments += 1

        # Кожні batch_size ітерацій оновлюємо значення у Hazelcast
        if (i + 1) % batch_size == 0:
            current_val = distributed_map.get("key")
            new_val = current_val + local_increments
            distributed_map.put("key", new_val)
            local_increments = 0

    # Записуємо залишок (якщо є)
    if local_increments > 0:
        current_val = distributed_map.get("key")
```

```
        new_val = current_val + local_increments
        distributed_map.put("key", new_val)

    client.shutdown()


if __name__ == "__main__":
    processes = []
    start_time = time.time()

    # Запуск 3 процесів, кожен робить 10_000 інкрементів
    for i in range(3):
        p = multiprocessing.Process(
            target=increment_without_lock,
            name=f"P{i+1}",
            args=(10000, 10)  # 10_000 ітерацій, оновлення кожні 10
        )
        p.start()
        processes.append(p)

    for p in processes:
        p.join()

    # Підключаємося окремо для виведення фінального значення
    client = hazelcast.HazelcastClient()
    final_value = client.get_map("distributed-map").blocking().get("key")
    elapsed = time.time() - start_time
    print(f"Фінальне значення ключа 'key' (без блокувань): {final_value}. Час
виконання: {elapsed:.2f} с.")
    client.shutdown()
```

- подивиться яке кінцеве значення для ключа “key” буде отримано (чи вийде 30К?)

```
Фінальне значення ключа 'key' (без блокувань): 25739. Час виконання: 187.34 с.
```

5. **Зробіть те саме з використанням песимістичним блокування та поміряйте час:**

```
import multiprocessing
import hazelcast
import time

def increment_with_pessimistic_lock(iterations=10000):
    client = hazelcast.HazelcastClient()
    distributed_map = client.get_map("distributed-map").blocking()
    lock = client.get_lock("key_lock")

    # Ініціалізація ключа "key"
    if distributed_map.get("key") is None:
        distributed_map.put("key", 0)

    for i in range(iterations):
        lock.lock()
        try:
            value = distributed_map.get("key")
            new_value = value + 1
            distributed_map.put("key", new_value)
        finally:
            lock.unlock()


    client.shutdown()

if __name__ == "__main__":
    processes = []
    start_time = time.time()

    for i in range(3):
        p = multiprocessing.Process(
            target=increment_with_pessimistic_lock,
            name=f"P{i+1}",
            args=(10000,)
        )
        p.start()
        processes.append(p)

    for p in processes:
        p.join()

    client = hazelcast.HazelcastClient()
    final_value = client.get_map("distributed-map").blocking().get("key")
    elapsed = time.time() - start_time
    print(f"Фінальне значення ключа 'key' (песимістичне блокування):
{final_value}. Час: {elapsed:.2f} с.")
    client.shutdown()
```

```
Фінальне значення ключа 'key' (песимістичне блокування): 30000. Час виконання: 486.89 с.
```

6. **Зробіть те саме з використанням оптимістичним блокуванням та поміряйте час:**

```python
import multiprocessing
import hazelcast
import time

def increment_with_optimistic_lock(iterations=10000):
    client = hazelcast.HazelcastClient()
    distributed_map = client.get_map("distributed-map").blocking()

    # Ініціалізація ключа "key"
    if distributed_map.get("key") is None:
        distributed_map.put("key", 0)

    for i in range(iterations):
        while True:
            old_value = distributed_map.get("key")
            new_value = old_value + 1
            # replace_if_same повертає True, якщо заміна пройшла успішно
            if distributed_map.replace_if_same("key", old_value, new_value):
                break

    client.shutdown()

if __name__ == "__main__":
    processes = []
    start_time = time.time()

    for i in range(3):
        p = multiprocessing.Process(
            target=increment_with_optimistic_lock,
            name=f"P{i+1}",
            args=(10000,)
        )
        p.start()
        processes.append(p)

    for p in processes:
        p.join()

    client = hazelcast.HazelcastClient()
    final_value = client.get_map("distributed-map").blocking().get("key")
    elapsed = time.time() - start_time
    print(f"Фінальне значення ключа 'key' (оптимістичне блокування):
{final_value}. Час: {elapsed:.2f} с.")
```

```
    client.shutdown()
```

```
Фінальне значення ключа 'key' (оптимістичне блокування): 30000. Час виконання: 351.12 с.
```

7. **Порівняйте результати кожного з запусків**

- для реалізації без блокувань маєте спостерігату втрату даних;
- для реалізації з песимістичним та оптимістичним блокуванням мають бути однакові результати
- песимістичний чи оптимістичний підхід працює швидше?
    - Оптимістичний працює швидше

8. **Робота з Bounded queue**

- на основі Distributed Queue налаштуйте Bounded queue на 10 елементів

```
<queue name="default">
    <statistics-enabled>true</statistics-enabled>
    <max-size>10</max-size>
    <backup-count>1</backup-count>
    <async-backup-count>0</async-backup-count>
    <empty-queue-ttl>-1</empty-queue-ttl>
    <merge-policy batch-size="100">com.hazelcast.spi.merge.PutIfAbsentMergePolicy</merge-policy>
</queue>
<multimap name="default">
```

- запустіть одного клієнта який буде писати в чергу значення 1..100, а двох інших які будуть читати з черги
    - під час вичитування, кожне повідомлення має вичитуватись одразу

```
[Producer] Додаємо значення: 1
[Producer] Додаємо значення: 2
[Consumer 1] Отримано значення: 1
[Producer] Додаємо значення: 3
[Consumer 2] Отримано значення: 2
[Producer] Додаємо значення: 4
[Consumer 1] Отримано значення: 3
[Producer] Додаємо значення: 5
[Consumer 2] Отримано значення: 4
[Producer] Додаємо значення: 6
[Consumer 1] Отримано значення: 5
[Producer] Додаємо значення: 7
[Consumer 2] Отримано значення: 6
[Producer] Додаємо значення: 8
[Consumer 1] Отримано значення: 7
[Producer] Додаємо значення: 9
[Consumer 2] Отримано значення: 8
[Producer] Додаємо значення: 10
[Consumer 1] Отримано значення: 9
[Producer] Додаємо значення: 11
[Consumer 2] Отримано значення: 10
[Producer] Додаємо значення: 12
[Consumer 1] Отримано значення: 11
[Producer] Додаємо значення: 13
[Consumer 2] Отримано значення: 12
...
[Producer] Надсилаємо сигнали завершення (None) для споживачів.
[Consumer 1] Отримано сигнал завершення. Завершення роботи.
[Consumer 2] Отримано сигнал завершення. Завершення роботи.
Усі процеси завершено. Робота Bounded Queue демонструвана успішно.
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```python
import multiprocessing
import hazelcast
import time

def producer():
    client = hazelcast.HazelcastClient()
    queue = client.get_queue("bounded-queue").blocking()

    # Запис чисел від 1 до 100
    for value in range(1, 101):
        print(f"[Producer] Додаємо значення: {value}")
        queue.put(value)  # Якщо черга заповнена (максимум 10 елементів), ця операція блокуватиметься
        time.sleep(0.1)    # Невелика затримка для наочності

    # Після завершення виробництва надсилаємо "сигнали завершення" для споживачів
    print("[Producer] Надсилаємо сигнали завершення (None) для споживачів.")
    # Оскільки споживачів два, надсилаємо два None
    queue.put(None)
    queue.put(None)

    client.shutdown()

def consumer(consumer_id):
    client = hazelcast.HazelcastClient()
    queue = client.get_queue("bounded-queue").blocking()

    while True:
        item = queue.take()  # Блокується, якщо черга порожня
        if item is None:
            print(f"[Consumer {consumer_id}] Отримано сигнал завершення. Завершення роботи.")
            break
        print(f"[Consumer {consumer_id}] Отримано значення: {item}")

    client.shutdown()

if __name__ == "__main__":
    processes = []

    # Запуск процесу-продюсера
    prod_process = multiprocessing.Process(target=producer, name="Producer")
    prod_process.start()
    processes.append(prod_process)

    # Запуск двох процесів-споживачів
    for i in range(1, 3):
        cons_process = multiprocessing.Process(target=consumer, name=f"Consumer-{i}", args=(i,))
        cons_process.start()
        processes.append(cons_process)
```

```
# Очікуємо завершення всіх процесів
for p in processes:
    p.join()

print("Усі процеси завершено. Робота Bounded Queue демонстрована успішно.")
```

- яким чином будуть вичитуватись значення з черги двома клієнтами?
    - Оскільки черга в Hazelcast – це структура з черговим видаленням, кожне значення, записане в чергу, після операції take() видаляється. Тобто, якщо два споживача читають із черги одночасно, кожен з них отримає деякі елементи (розподіл може бути нерівномірним залежно від того, хто першим виконає take()), і кожне значення буде спожите лише одним клієнтом.
- перевірте яка буде поведінка на запис якщо відсутнє читання, і черга заповнена
    - Якщо продюсер намагається додати елемент в уже заповнену чергу (10 елементів), то метод put() буде блокуватися до тих пір, поки хоча б один елемент не буде забраний з черги. Якщо використовувати offer() з таймаутом, то при переповненні метод поверне False після закінчення таймауту, що дозволяє обробити ситуацію.