



МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра Інформаційної Безпеки

Проектування розподілених систем
Лабораторна робота №2
Розгортання і робота з distributed in-memory data structures
на основі Hazelcast: Distributed Map

Виконав:
студент V курсу
групи ФБ-41мп
Африканський О. М.

Київ 2025

Мета: навчитися розгортати та конфігурувати кластер Hazelcast, працювати з розподіленими структурами даних (Map, Queue), дослідити механізми реплікації, блокування та відмовостійкості, а також оцінити продуктивність при конкурентному доступі до спільних ресурсів.

Хід роботи

1. Встановити і налаштувати Hazelcast

Для встановлення кластеру було виконано наступні кроки:

```
sudo docker network create hz-net
```

Маємо такий compose.yml:

```
services:
  hz-1:
    container_name: 'hz-1'
    image: 'hazelcast/hazelcast:5.4.0'
    network_mode: 'hz-net'
    environment:
      - HZ_CLUSTERNAME=dev-map
      - HZ_NETWORK_PUBLICADDRESS=172.18.0.1:5701
    ports:
      - '5701:5701'

  hz-2:
    container_name: 'hz-2'
    image: 'hazelcast/hazelcast:5.4.0'
    network_mode: 'hz-net'
    environment:
      - HZ_CLUSTERNAME=dev-map
      - HZ_NETWORK_PUBLICADDRESS=172.18.0.1:5702
    ports:
      - '5702:5701'

  hz-3:
    container_name: 'hz-3'
    image: 'hazelcast/hazelcast:5.4.0'
    network_mode: 'hz-net'
    environment:
      - HZ_CLUSTERNAME=dev-map
      - HZ_NETWORK_PUBLICADDRESS=172.18.0.1:5703
    ports:
      - '5703:5701'

  hazelcast-management:
    container_name: 'distmap-management-center'
```

```
image: 'hazelcast/management-center:5.4.0'
network_mode: 'hz-net'
depends_on:
  - hz-1
  - hz-2
  - hz-3
ports:
  - '8080:8080'
```

```
networks:
  hz-net:
    driver: bridge
```

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

Для запуску кластеру виконуємо наступну команду:

```
sudo docker compose up -d
[+] Running 4/4
✓ Container hz-1                Started           1.2s
✓ Container hz-2                Started           1.4s
✓ Container hz-3                Started           1.3s
✓ Container distmap-management-center Started           2.9s
```

Перевіримо чи кластер об'єднався:

```
sudo docker logs hz-1
...
Members {size:3, ver:3} [
  Member [172.18.0.1]:5703 - 933fbd5c-d765-4c0d-aa5d-f70083f5482c
  Member [172.18.0.1]:5701 - 31d7b057-ab23-4a51-be10-ed3bc34d7c4d this
  Member [172.18.0.1]:5702 - 461786fd-f4bb-4d3e-bde2-408d808ed317
]
...
```

Тепер перевіримо на менеджері. Спершу додамо кластер:

Connect Directly

Cluster Name ?

dev-map

Member Addresses ?

172.18.0.1:5703,172.18.0.1:5702,172.18.0.1:5701

Enabled ☒

Cancel

CONNECT

Потім перевіримо під'єднання:

Cluster Status 

 State	Active
 Migrations	0 waiting migrations
 Filtering	Disabled / Allow

3. Продемонструйте роботу Distributed Map

Спершу створимо код, який створить необхідні записи у Distributed Map (для виконання завдання використовувалась мова програмування GO):

```
package main

import (
    "context"
    "fmt"
    "log"
    "time"

    "github.com/hazelcast/hazelcast-go-client"
)

func main() {
    // Ініціалізація клієнта Hazelcast
    config := hazelcast.NewConfig()
    config.Cluster.Name = "dev-map"
    config.Cluster.Network.SetAddresses("172.18.0.1:5701")

    // Запуск клієнта
    ctx := context.Background()
    client, err := hazelcast.StartNewClientWithConfig(ctx, config)
    if err != nil {
        log.Fatalf("Помилка запуску клієнта Hazelcast: %v", err)
    }
    defer client.Shutdown(ctx)

    // Отримання або створення розподіленої карти
    hzMap, _ := client.GetMap(ctx, "demo-distributed-map")
}
```

```

// Очищення карти
hzMap.Clear(ctx)

// Вимірювання часу запису
startTime := time.Now()

// Запис 1000 значень у карту
for i := 0; i < 1000; i++ {
    hzMap.Set(ctx, i, i)
}

endTime := time.Now()
duration := endTime.Sub(startTime)

fmt.Printf("Запис 1000 ключів зайняв: %v\n", duration)

fmt.Println("Успішно збережено 1000 ключів у Distributed Map")

// Виведення розміру карти
size, _ := hzMap.Size(ctx)
fmt.Printf("Розмір карти: %d\n", size)

// Виведення перших 5 значень
for i := 0; i < 5; i++ {
    value, _ := hzMap.Get(ctx, i)
    fmt.Printf("Ключ: %d, Значення: %v\n", i, value)
}
}

```

Після виконання коду, отримуємо наступний результат:

```

user@ubuntu:~/Documents/2_sem/dist_sys/2$ go run map_demo.go
2025/04/16 22:01:10 INFO : trying to connect to cluster: dev-map
2025/04/16 22:01:10 INFO : connected to cluster: dev-map
2025/04/16 22:01:10 INFO :

Members {size:3, ver:3} [
    Member 172.18.0.1:5703 - 933fbd5c-d765-4c0d-aa5d-f70083f5482c
    Member 172.18.0.1:5701 - 31d7b057-ab23-4a51-be10-ed3bc34d7c4d
    Member 172.18.0.1:5702 - 461786fd-f4bb-4d3e-bde2-408d808ed317
]

Запис 1000 ключів зайняв: 2.584167964s
Успішно збережено 1000 ключів у Distributed Map
Розмір карти: 1000
Ключ: 0, Значення: 0
Ключ: 1, Значення: 1
Ключ: 2, Значення: 2
Ключ: 3, Значення: 3
Ключ: 4, Значення: 4

```

Перевіримо результат у manager console:

Map Statistics (In-Memory Format: BINARY)

Member ^	^ Entries	^ Gets
172.18.0.1:5701	364	18
172.18.0.1:5702	320	12
172.18.0.1:5703	316	0
TOTAL	1,000	30

Тепер виконаємо відключення однієї ноди та переглянемо результати:

```
user@ubuntu:~/Documents/2_sem/dist_sys/2$ sudo docker stop hz-1
hz-1
user@ubuntu:~/Documents/2_sem/dist_sys/2$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND
CREATED        STATUS        PORTS
NAMES
e3446afa166e   hazelcast/management-center:5.4.0   "bash ./bin/mc-start..."
5 days ago    Up 37 minutes  8081/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 8443/tcp   distmap-management-center
1769f0e37e15   hazelcast/hazelcast:5.4.0           "hz start"
5 days ago    Up 37 minutes  0.0.0.0:5702->5701/tcp, :::5702->5701/tcp
hz-2
8e78fb74f3b2   hazelcast/hazelcast:5.4.0           "hz start"
5 days ago    Up 37 minutes  0.0.0.0:5703->5701/tcp, :::5703->5701/tcp
hz-3
```

Тепер передивимось manager console:

Map Statistics (In-Memory Format: BINARY)

Member ^	^ Entries	^ Gets
172.18.0.1:5702	496	12
172.18.0.1:5703	504	0
TOTAL	1,000	12

Як можна побачити, в кластері залишилось 2 ноди та дані розділились між ними двома. Вимкнемо другу ноду:

```
user@ubuntu:~/Documents/2_sem/dist_sys/2$ sudo docker stop hz-2
hz-2
user@ubuntu:~/Documents/2_sem/dist_sys/2$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND
CREATED        STATUS        PORTS
NAMES
e3446afa166e   hazelcast/management-center:5.4.0   "bash ./bin/mc-start..."
5 days ago    Up 39 minutes  8081/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 8443/tcp   distmap-management-center
8e78fb74f3b2   hazelcast/hazelcast:5.4.0           "hz start"
5 days ago    Up 39 minutes  0.0.0.0:5703->5701/tcp, :::5703->5701/tcp
hz-3
```

Map Statistics (In-Memory Format: BINARY)

Member ^	^ Entries	^ Gets
172.18.0.1:5703	1,000	0
TOTAL	1,000	0

Як можна побачити, кластер налічує ноду та вона має усі дані. Повернемо дві ноди назад.

```
user@ubuntu:~/Documents/2_sem/dist_sys/2$ sudo docker start hz-2
hz-2
user@ubuntu:~/Documents/2_sem/dist_sys/2$ sudo docker start hz-1
hz-1
user@ubuntu:~/Documents/2_sem/dist_sys/2$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND
CREATED        STATUS        PORTS
NAMES
e3446afa166e   hazelcast/management-center:5.4.0   "bash ./bin/mc-start..."
5 days ago    Up 42 minutes  8081/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 8443/tcp   distmap-management-center
1769f0e37e15   hazelcast/hazelcast:5.4.0           "hz start"
5 days ago    Up 31 seconds  0.0.0.0:5702->5701/tcp, :::5702->5701/tcp
hz-2
```

```

a44ec65b8992    hazelcast/hazelcast:5.4.0    "hz start"
5 days ago    Up 21 seconds    0.0.0.0:5701->5701/tcp, :::5701->5701/tcp
hz-1
8e78fb74f3b2    hazelcast/hazelcast:5.4.0    "hz start"
5 days ago    Up 42 minutes    0.0.0.0:5703->5701/tcp, :::5703->5701/tcp
hz-3

```

Map Statistics (In-Memory Format: BINARY)

Member ^	^ Entries	^ Gets
172.18.0.1:5701	344	0
172.18.0.1:5702	340	0
172.18.0.1:5703	316	0
TOTAL	1,000	0

Як можна побачити, після повернення двох нод усі дані рівномірно розсіялись між трьома нодами. Тепер прирвемо роботу двох нод одночасно:

```

user@ubuntu:~/Documents/2_sem/dist_sys/2$ sudo docker kill hz-1 hz-2
hz-1
hz-2
user@ubuntu:~/Documents/2_sem/dist_sys/2$ sudo docker ps
CONTAINER ID    IMAGE                                COMMAND
CREATED        STATUS          PORTS
NAMES
e3446afa166e    hazelcast/management-center:5.4.0    "bash ./bin/mc-start..."
5 days ago    Up 45 minutes    8081/tcp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 8443/tcp    distmap-management-center
8e78fb74f3b2    hazelcast/hazelcast:5.4.0            "hz start"
5 days ago    Up 45 minutes    0.0.0.0:5703->5701/tcp, :::5703->5701/tcp
hz-3

```

Map Statistics (In-Memory Format: BINARY)

Member ^	^ Entries	^ Gets
172.18.0.1:5703	653	0
TOTAL	653	0

Як можна побачити з `manager console`, деяка частина даних була втрачена.

Для того, щоб данні не втратились потрібно створити конфігураційний файл `hazelcast.yml` та в ньому вказати назву `map` та наступний параметр: `backup-count = 2`. Це надасть можливість відновити усі дані навіть після втрати двох нод.

4. Продемонструйте роботу **Distributed Map without locks**

Для цього було створено, який запускає код з записом в один ключ паралельно три рази (`map_demo-2.go`). Після його запуску отримуємо наступний результат:

```
Клієнт 1 завершив 10000 ітерацій  
Клієнт 2 завершив 10000 ітерацій  
Клієнт 0 завершив 10000 ітерацій  
Кінцеве значення ключа 'key': 13122  
Очікуване значення: 30000  
Час виконання: 44.454360353s
```

Результат із кінцевим значенням 13,122 замість 30,000 чітко ілюструє проблему `race conditions` у `Distributed Map` без синхронізації. Час виконання (44.45 секунд) вказує на значне навантаження через конкуренцію.

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

Для цього було створено, який запускає код з записом в один ключ паралельно три рази (`map_demo-3.go`). З песимістичним блокуванням використовуються методи `Lock` і `Unlock` для блокування ключа "key" перед кожним інкрементом, забезпечуючи песимістичне блокування, яке гарантує ексклюзивний доступ і усуває `race conditions`. Після його запуску отримуємо наступний результат:

```
Клієнт 1 завершив 10000 ітерацій  
Клієнт 0 завершив 10000 ітерацій  
Клієнт 2 завершив 10000 ітерацій  
Кінцеве значення ключа 'key': 30000  
Очікуване значення: 30000  
Час виконання: 4m31.656283683s
```

Код із песимістичним блокуванням досяг точного значення ключа "key" (30,000), але виконався повільно (4 хв 31.66 с) через накладні витрати на блокування.

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

Для виконання завдання створено код (map_demo-4.go), який запускає три паралельні клієнти для інкременту одного ключа в Distributed Map. З оптимістичним блокуванням використовується метод `ReplaceIfSame` для атомарного оновлення ключа "key", що перевіряє поточне значення перед записом нового, повторюючи спробу при зміні значення іншим клієнтом. Це забезпечує оптимістичне блокування, яке усуває `race conditions` без ексклюзивного доступу. Після запуску коду отримуємо наступний результат:

```
Клієнт 0 завершив 10000 ітерацій  
Клієнт 2 завершив 10000 ітерацій  
Клієнт 1 завершив 10000 ітерацій  
Кінцеве значення ключа 'key': 30000  
Очікуване значення: 30000  
Час виконання: 43.653763775s
```

Цей результат демонструє успішну конкурентну роботу з розподіленою мапою Hazelcast без явного використання блокувань. Завдяки внутрішнім механізмам Hazelcast (таким як атомарні операції `get` та `replaceIfSame`), усі три клієнти змогли успішно інкрементувати значення, не втративши жодного оновлення. Це підтверджує можливість досягнення високої пропускної здатності при паралельній роботі з даними в Hazelcast.

7. Порівняйте результати кожного з запусків

Порівняння результатів:

- **Без блокувань:** Втрата даних (кінцеве значення 13122 замість 30000), час виконання 44.45 с.
- **Песимістичне блокування:** Дані цілісні (кінцеве значення 30000), дуже повільний час виконання (4 хв 31.66 с).
- **Оптимістичне блокування:** Дані цілісні (кінцеве значення 30000), швидкий час виконання (43.65 с).

Швидкість:

Оптимістичний підхід (43.65с) значно швидший за песимістичний (4хв 31.66с).

8. Робота з Bounded queue

Для того що творити Bounded queue потрібно у конфігураційному файлі hazelcast-docker.xml встановити значення розміру queue до 10:

```
cat hazelcast-docker.xml | grep -A 8 "queue name"
    <queue name="default">
      <!--
        Maximum size of the queue. When a JVM's local queue size reaches
the maximum,
        all put/offer operations will get blocked until the queue size
        of the JVM goes down below the maximum.
        Any integer between 0 and Integer.MAX_VALUE. 0 means
        Integer.MAX_VALUE. Default is 0.
      -->
      <max-size>10</max-size>
```

Далі запусимо код, який запускає одного записника та два клієнти черги:

```
Черга 'default' успішно отримана.
Виробник 0: Додано '1' до черги.
Споживач 1: Отримано '1' з черги.
Виробник 0: Додано '2' до черги.
Споживач 2: Отримано '2' з черги.
Виробник 0: Додано '3' до черги.
...
Виробник 0: Додано '98' до черги.
Споживач 2: Отримано '97' з черги.
Споживач 1: Отримано '98' з черги.
Виробник 0: Додано '99' до черги.
Виробник 0: Додано '100' до черги.
Споживач 2: Отримано '99' з черги.
Споживач 1: Отримано '100' з черги.
Виробник 0 завершив додавання 100 елементів.
Споживач 2: Черга порожня, завершую роботу.
Споживач 1: Черга порожня, завершую роботу.
Всі клієнти завершили роботу з 'default'.
```


Як можна побачити вище, значення з queue вичитувались по черзі за принципом FIFO. Отже перший клієнт, який був готовий вичитати значення вичитував його зразу як воно з'являлось у черзі. Тепер переглянемо що буде, якщо заповнити чергу та не читати її:

```
Демонстрація поведінки при заповненій черзі 'default' без читачів...
Додано 'item-1' до черги 'default'.
Додано 'item-2' до черги 'default'.
Додано 'item-3' до черги 'default'.
```

Додано 'item-4' до черги 'default'.
Додано 'item-5' до черги 'default'.
Додано 'item-6' до черги 'default'.
Додано 'item-7' до черги 'default'.
Додано 'item-8' до черги 'default'.
Додано 'item-9' до черги 'default'.
Додано 'item-10' до черги 'default'.
Додано 'item-11' до черги 'default'.
Додано 'item-12' до черги 'default'.

Queues 

Name ^	Persistence ^	Items ^	Backups ^
default	Disabled	10	10

1 - 1 of 1 Rows 10 

Як можна побачити за результатами вище, код виконував спробу запису 12 елементів, а фактично у черзі записано тільки 10.

Висновки: У ході роботи було успішно розгорнуто та сконфігуровано кластер Hazelcast із трьома нодами за допомогою Docker Compose, що дозволило дослідити роботу розподілених структур даних, таких як Distributed Map та Bounded Queue. Продемонстровано механізми реплікації та відмовостійкості: дані автоматично перерозподілялися між нодами при відключенні чи поверненні інстансів, хоча без належної конфігурації втрата двох нод одночасно призводила до часткової втрати даних. Експерименти з Distributed Map показали, що без синхронізації виникають race conditions, песимістичне блокування забезпечує цілісність даних, але значно уповільнює виконання, тоді як оптимістичне блокування поєднує точність і високу продуктивність. Робота з Bounded Queue підтвердила її FIFO-природу та обмеження розміру, блокуючи подальший запис при заповненні без читачів. Загалом, Hazelcast виявився потужним інструментом для роботи з розподіленими даними, де правильний вибір стратегії синхронізації критично впливає на продуктивність і надійність системи. Для забезпечення повної відмовостійкості необхідна додаткова конфігурація резервного копіювання.