

Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

“Київський політехнічний інститут”

Фізико-Технічний інститут

# **Лабораторна робота № 1 з семестрового курсу**

# **“Проектування високонавантажених систем”**

Виконала:

Студентка групи ФІ-03

Швець Катерина

Київ, 2023

**Реалізація каунтера з використанням Hazelcast**

**Завдання:**

1. Встановити і налаштувати Hazelcast <http://hazelcast.org/download/>
2. Сконфігурувати і запустити 3 ноди (інстанси) об'єднані в кластер або як частину Java-застосування, або як окремі застосування <https://hazelcast.org/getting-started-with-hazelcast/>

У справжній системі кожна нода має запускатись на окремому сервері.

1. Далі, на основі прикладу з Distributed Map, напишіть код який буде емулювати інкремент значення для одного й того самого ключа у циклі до 10К. Це необхідно робити у 10 потоках.

*for (t in 1..10) {*

*thread {*

*for (i in 1..10\_000) {*

*increment\_counter()*

*// distribMap.put(key, counter + 1)*

*}*

*}*

*}*

Виходячи з того, шо 10 потоків інкрементують каунтер 10К разів кожен, то остаточне значення каунтера має бути 10\*10\_000 = 100\_000

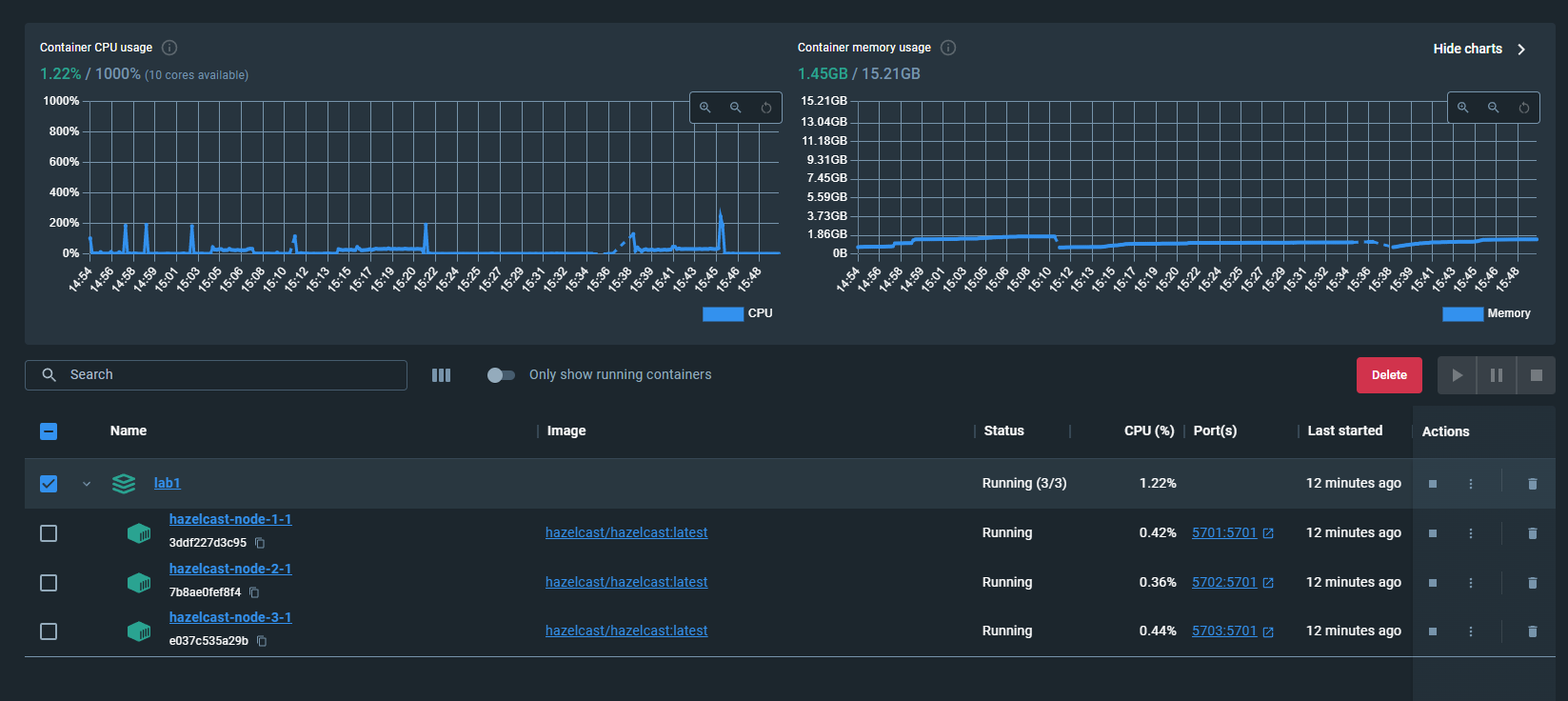
Для імплементації спочатку скористаємось Distributed Map у Hazelcast

1. На основі прикладу <https://docs.hazelcast.com/imdg/latest/data-structures/map#locking-maps> реалізуйте каунтер без блокувань. Поміряйте час виконання, та подивиться чи коректне кінцеве значення каунтера ви отримаєте.
2. На основі прикладу <https://docs.hazelcast.com/imdg/latest/data-structures/map#pessimistic-locking> реалізуйте каунтер з використанням песимістичного блокування. Поміряйте час виконання, та подивиться чи коректне кінцеве значення каунтера ви отримаєте.
3. На основі прикладу <https://docs.hazelcast.com/imdg/latest/data-structures/map#optimistic-locking> реалізуйте каунтер з використанням оптимістичного блокування. Поміряйте час виконання, та подивиться чи коректне кінцеве значення каунтера ви отримаєте.
4. На справді, в загальному випадку, Distributed Map у Hazelcast не гарантує, що у випадку падіння ноди на інші ноди встигне реплікуватись (скопіюватись) значення ключа у Distributed Map.  
   Тому використаємо іншу структуру даних, яка дає більші гарантії - *IAtomicLong*   
   та використовує так званий протокол консенсусу Raft: <https://docs.hazelcast.com/hazelcast/5.1/data-structures/iatomiclong>

Реалізуйте каунтер з використанням *IAtomicLong* та увімкнувши підтимку *CP Sysbsystem* (<https://docs.hazelcast.com/hazelcast/5.1/cp-subsystem/configuration#quickstart>) на основі трьох нод.

Реалізовано 4 метода: каунер без блокувань, каунтер з використанням песимістичного та оптимістичним блокуванням, каунтер з використанням IAtomicLong.

Встановила Hazelcast та Docker. Сконфігурувала та запустила 3 ноди як окремі застосування(показані на скриншоті знизу).



Далі згідно прикладів у завданні реалізовала 4 каунтери. Зробила вивід часу виконання. Результати:

|  |  |  |
| --- | --- | --- |
| Каунтер | Кінцеве значення | Час |
| без блокувань | 16904 | 30.02s |
| з використанням песимістичного блокуванням | 100000 | 230.06s |
| з використанням оптимістичним блокуванням | 100000 | 145.06s |
| з використанням IAtomicLong | 100000 | 15.02s |

Результат виконання збережено у powershell\_log.txt