**Task 2**

**Реалізація каунтера з використанням Hazelcast**

Необхідно імплементувати дуже простий функціонал, який є частиную соціальних мереж, відео-платформ, сервісів фото та відео шарингу, тощо. Це є лічильник (каунтер) лайків/переглядів/ретвітів …, що збільшується на один при відповідній дії.

Необхідною умовою є:

* щоб через одночасний доступ (*race condition*) не було втрати проміжних значень;
* щоб данні не втрачались при раптовому перезавантаженні сервера.

Останній пункт може бути релізований за допомогою запису кожного оновлення на жорсткий диск, або за допомогою реплікації значення у пам’ять іншого сервера.

У данній роботі вам необхідно буде реалізувати другий підхід за допомогою різних структур даних Hazelcast.

Hazelcast є розподіленим сховище даних в оперативній пам'яті. "Розподілене" означає те, що на кожній з нод (серверів) системи запускається свій екземпляр Hazelcast, які потім об'єднуються в загальний кластер. В рамках цього кластера, через API можна створювати різні розподілені структури даних: Map, Queue, Topic, Lock, ...

Кластер одночасно дозволяє підвищити відмовостійкість системи зберігання (за рахунок того, що данні будуть репликуватись на інші ноди) та розподілити навантаження (за рахунок масштабування в ширину).

<https://docs.hazelcast.com/hazelcast/5.5/deploy/choosing-a-deployment-option>

**Запуск Hazelcast з Java-застосування**

Ноду можна створювати та запускати напряму з Java-додатку, який буде працювати на тому же сервері де запущений екземпляр Hazelcast. Додаток матиме доступ через API доступ до цих розподіленим структурам даних, і зможе писати/читати в/з них.

<https://docs.hazelcast.com/hazelcast/5.5/getting-started/get-started-java>

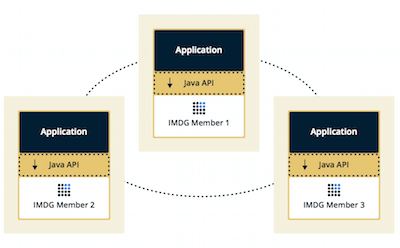
HazelcastInstance hzInstance = Hazelcast.newHazelcastInstance();

Map<String, String> distributedMap = hzInstance.getMap( "capitals" );

capitalcities.put( "1", "Tokyo" );

capitalcities.put( "2", "Paris" );

При цьому, інші додатки підключені до інших нод кластеру Hazelcast будуть також бачити зміни в розподіленим структурам даних, і також можуть писати/читати в/з них.



**Запуск нод Hazelcast окремо**

Іншим способом є запуск нод кластеру, як окремих застосувань через командний рядок та підключення до нього за допомгою клієнтів, які існують під різні мови програмування

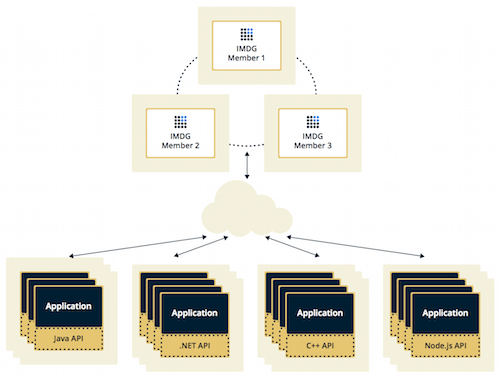
<https://docs.hazelcast.com/hazelcast/5.5/getting-started/get-started-binary>

HazelcastInstance hzInstance = Hazelcast.newHazelcastClient();

Map<String, String> distributedMap = hzInstance.getMap( "capitals" );

capitalcities.put( "1", "Tokyo" );

capitalcities.put( "2", "Paris" );



**Завдання:**

1. Встановити і налаштувати **Hazelcast 5.4.x** (у новіших версіях частина необхідного для виконання завдань функціоналу є платною)

<https://docs.hazelcast.com/hazelcast/5.4/getting-started/install-hazelcast> <https://hazelcast.com/community-edition-projects/downloads/archives/#hazelcast-platform-5-4-0>

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

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/5.4/data-structures/iatomiclong>

Реалізуйте каунтер з використанням *IAtomicLong* та увімкнувши підтимку *CP Sysbsystem* (<https://docs.hazelcast.com/hazelcast/5.4/cp-subsystem/configuration>) на основі трьох нод.   
**УВАГА! Без *CP Sysbsystem* не гарантується коректність результату** (у протоколі мають бути логи Hazelcast з яких видно, що CP Subsystem активована та складається з 3-х нод)

Поміряйте час виконання, та подивиться чи коректне кінцеве значення каунтера ви отримаєте.

**Вимоги до звіту та реалізації**

* Мова реалізації будь-яка
* Має бути надано код програми/скрипта та результати виконання
* Приведно лог, який видають ноди Hazelcast, де буде видно що кластер складається з 3-х нод і що активована CP Subsystem

**Literature**

Hazelcast Python Client <https://hazelcast.com/clients/python/>

Hazelcast Go Client <https://pkg.go.dev/github.com/hazelcast/hazelcast-go-client>

Hazelcast in Docker <https://docs.hazelcast.com/hazelcast/5.4/getting-started/get-started-docker>