**Task 2**

**Розгортання і робота з distributed in-memory data structures на основі Hazelcast:**

**Distributed Map**

Hazelcast є розподіленою платформою для зберігання та обробки даних в реальному часі 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, Data-stream processing pipeline,...*

**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: <https://docs.hazelcast.com/hazelcast/5.3/getting-started/get-started-java>

Додаток матиме доступ через API до цих розподіленим структурам даних, і зможе писати/читати в/з них.

HazelcastInstance hzInstance = Hazelcast.newHazelcastInstance();

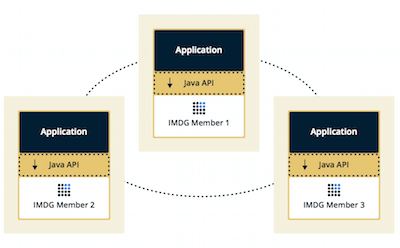
Map<String, String> distributedMap = hzInstance.getMap( "capitals" );

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.

<https://docs.hazelcast.com/hazelcast/5.3/deploy/choosing-a-deployment-option>



**Запуск нод Hazelcast окремо**

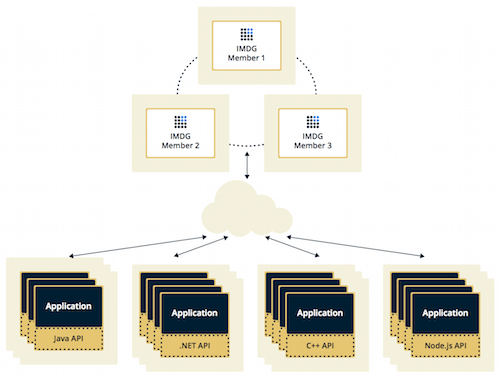
Іншим способом є запуск нод кластеру, як окремих застосувань через командний рядок (<https://docs.hazelcast.com/hazelcast/5.3/getting-started/get-started-binary>) або як Docker-контейнер (<https://docs.hazelcast.com/hazelcast/5.3/getting-started/get-started-docker>) та підключення до нього за допомогою клієнтів, які існують під різні мови програмування

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 <https://hazelcast.com/open-source-projects/downloads/>
2. Сконфігурувати і запустити 3 ноди (інстанси) об'єднані в кластер або як частину Java-застосування, або як окремі застосування <https://docs.hazelcast.com/hazelcast/5.3/getting-started/get-started-binary#step-6-scale-your-cluster>
3. Продемонструйте роботу Distributed Map

<https://docs.hazelcast.com/hazelcast/5.3/data-structures/creating-a-map>

* використовуючи API створіть Distributed Map
* запишіть в неї 1000 значень з ключем від 0 до 1к
* за допомогою Management Center (<https://docs.hazelcast.com/management-center/5.3/getting-started/install#before-you-begin>) подивиться на розподіл значень по нодах
* подивитись як зміниться розподіл даних по нодах:
  + якщо відключити одну ноду
  + відключити дві ноди.
  + Чи буде втрата даних?
  + Яким чином зробити щоб не було втрати даних?

1. Продемонструйте роботу з Topic <https://docs.hazelcast.com/hazelcast/5.3/data-structures/topic>

* запустіть одного клієнта який буде писати в Topic значення 1..100, а двох інших які будуть читати з Topic
* яким чином будуть вичитуватись значення з Topic двома клієнтами?
* якщо один з читачів буде певний час неактивний, чи отримає він повідомлення які він пропустив?

1. Робота з Bounded queue

* на основі Distributed Queue (<https://docs.hazelcast.com/hazelcast/5.3/data-structures/queue#creating-an-example-queue>) налаштуйте Bounded queue на 10 елементів (<https://docs.hazelcast.com/hazelcast/5.3/data-structures/queue#configuring-queue>)
* запустіть одного клієнта який буде писати в чергу значення 1..100, а двох інших які будуть читати з черги
* яким чином будуть вичитуватись значення з черги двома клієнтами?
* перевірте яка буде поведінка на запис якщо відсутнє читання, і черга заповнена

**Вимоги до протоколу**

У протоколі має міститись:

* посилання на GitHub з кодом
* вміст консолі з виводом результатів виконання коду

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

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