Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Фізико-технічний інститут

Лабораторна робота №1

з дисципліни

«Проектування високонавантажених систем»

на тему:   
«Реалізація каунтера з використанням Hazelcast»

Виконав:

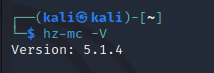
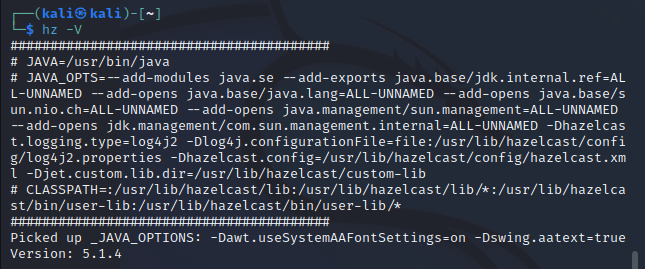
студент групи ФБ-21мп

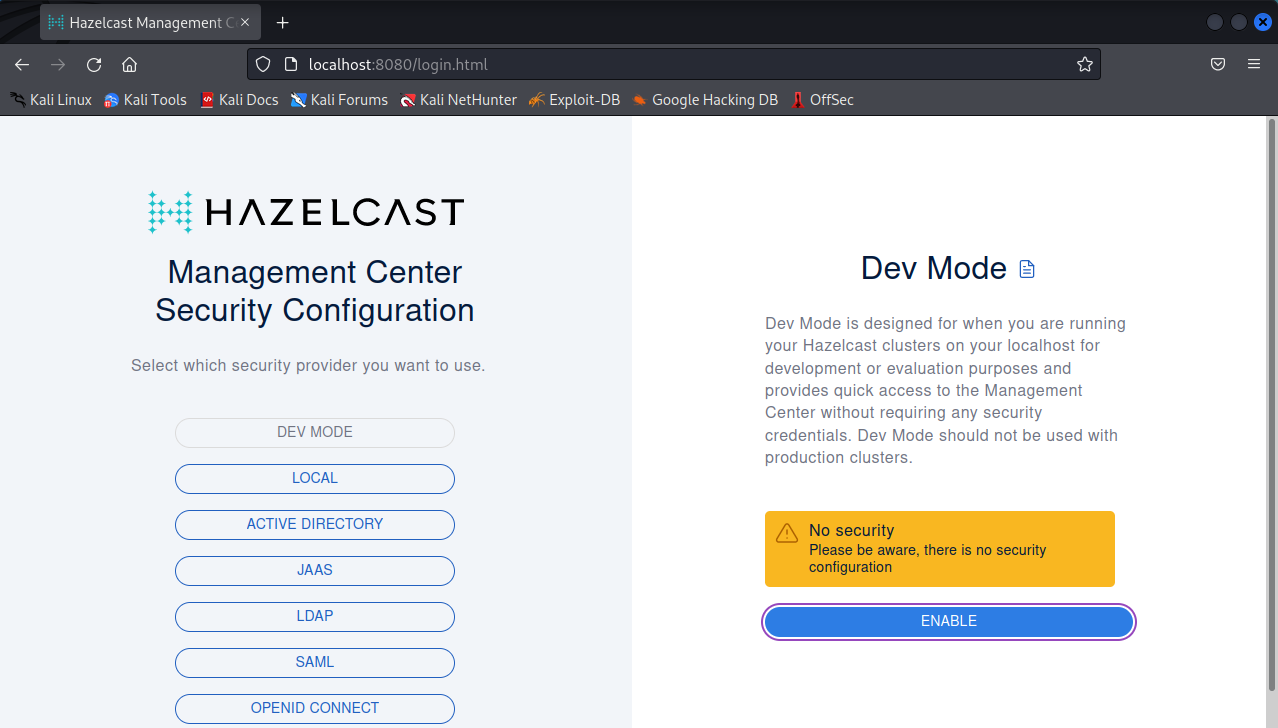
Тущенко Денис

Перевірив:

Родіонов А. М.

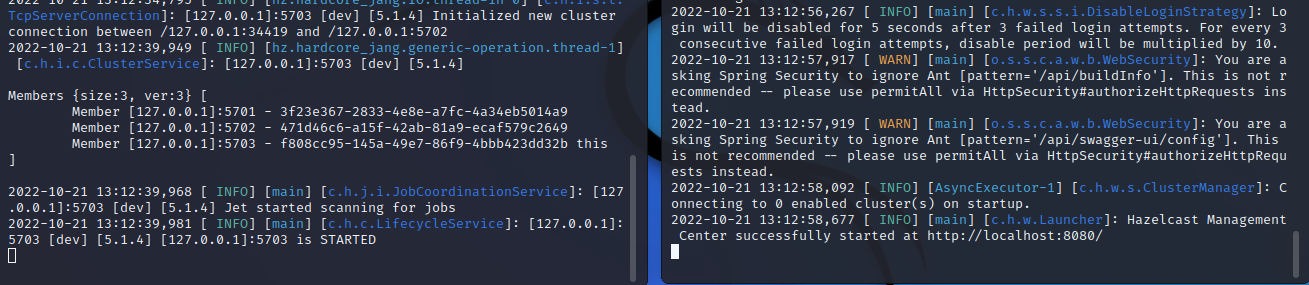
1. Встановити і налаштувати Hazelcast <http://hazelcast.org/download/>

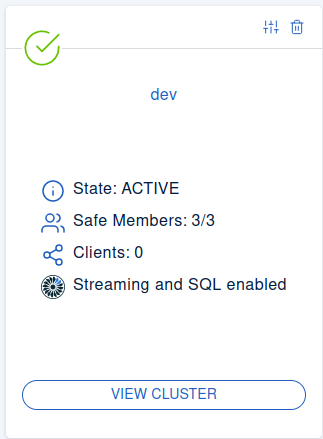




1. Сконфігурувати і запустити 3 ноди (інстанси) об'єднані в кластер або як частину Java-застосування, або як окремі застосування <https://hazelcast.org/getting-started-with-hazelcast/>

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.
2. 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.
3. На основі прикладу [https://docs.hazelcast.com/imdg/latest/data-structures/map#pessimistic-locking](https://docs.hazelcast.com/imdg/latest/data-structures/map" \l "pessimistic-locking)  реалізуйте каунтер з використанням песимістичного блокування. Поміряйте час виконання, та подивиться чи коректне кінцеве значення каунтера ви отримаєте.
4. На основі прикладу [https://docs.hazelcast.com/imdg/latest/data-structures/map#optimistic-locking](https://docs.hazelcast.com/imdg/latest/data-structures/map" \l "optimistic-locking)  реалізуйте каунтер з використанням оптимістичного блокування. Поміряйте час виконання, та подивиться чи коректне кінцеве значення каунтера ви отримаєте.
5. 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>

Код:

from threading import Thread

from threading import Lock

import time

import hazelcast

import yaml

def parse\_yaml():

with open("hazelcast.yaml", "r") as stream:

try:

yaml.safe\_load(stream)

except yaml.YAMLError as exc:

print(exc)

map\_name = "my-distributed-map"

class ThreadCounter():

# constructor

def \_\_init\_\_(self):

# initialize counter

self.\_counter = 0

# initialize lock

self.\_lock = Lock()

# increment the counter

def increment(self):

self.\_counter += 1

def third\_task(counter):

hz = hazelcast.HazelcastClient()

key = "Task3"

map = hz.get\_map("my-distributed-map").blocking()

for i in range(10000):

with counter.\_lock:

counter.increment()

map.put(key, counter.\_counter)

def forth\_task(counter):

hz = hazelcast.HazelcastClient()

key = "Task4"

map = hz.get\_map("my-distributed-map").blocking()

for i in range(10000):

counter.\_counter = map.get(key)

counter.\_counter += 1

map.put(key, counter.\_counter)

def fifth\_task():

hz = hazelcast.HazelcastClient()

key = "Task5"

map = hz.get\_map("my-distributed-map").blocking()

for i in range(10000):

map.lock(key)

try:

value = map.get(key)

value += 1

map.put(key, value)

#print(map.get(key))

finally:

map.unlock(key)

def sixth\_task(counter):

hz = hazelcast.HazelcastClient()

key = "Task6"

map = hz.get\_map("my-distributed-map").blocking()

loop = True

for i in range(10000):

while loop is not False:

oldValue = map.get(key)

newValue = oldValue

newValue += 1

if (map.replace\_if\_same(key, oldValue, newValue)):

break

def seventh\_task(counter):

hz = hazelcast.HazelcastClient()

key = "Task7"

map = hz.get\_map("my-distributed-map").blocking()

count = hz.cp\_subsystem.get\_atomic\_long("counter").blocking()

value = 0

for i in range(10000):

value = count.add\_and\_get(1)

#print(value)

map.put(key, value)

def output(param):

hz = hazelcast.HazelcastClient()

key = param

map = hz.get\_map("my-distributed-map").blocking()

print("Finished! Result = ", map.get(key))

if \_\_name\_\_ == "\_\_main\_\_":

parse\_yaml()

counter = ThreadCounter()

print("=======TASK 3=======")

start\_time = time.time()

threads = [Thread(target=third\_task, args=(counter,)) for \_ in range(10)]

for thread in threads:

thread.start()

for thread in threads:

thread.join()

output("Task3")

print("--- %s seconds ---" % (time.time() - start\_time))

print("\n\n=======TASK 4=======")

hz = hazelcast.HazelcastClient()

map = hz.get\_map("my-distributed-map").blocking()

key = "Task4"

map.put(key, 0)

start\_time = time.time()

threads = [Thread(target=forth\_task, args=(counter,)) for \_ in range(10)]

for thread in threads:

thread.start()

for thread in threads:

thread.join()

output("Task4")

print("--- %s seconds ---" % (time.time() - start\_time))

print("\n\n=======TASK 5=======")

key = "Task5"

map.put(key, 0)

start\_time = time.time()

threads = [Thread(target=fifth\_task) for \_ in range(10)]

for thread in threads:

thread.start()

for thread in threads:

thread.join()

output("Task5")

print("--- %s seconds ---" % (time.time() - start\_time))

print("\n\n=======TASK 6=======")

key = "Task6"

map.put(key, 0)

start\_time = time.time()

threads = [Thread(target=sixth\_task, args=(counter,)) for \_ in range(10)]

for thread in threads:

thread.start()

for thread in threads:

thread.join()

output("Task6")

#output("Task6\_1")

print("--- %s seconds ---" % (time.time() - start\_time))

print("\n\n=======TASK 7=======")

key = "Task7"

count = hz.cp\_subsystem.get\_atomic\_long("counter").blocking()

count.set(0)

start\_time = time.time()

threads = [Thread(target=seventh\_task, args=(counter,)) for \_ in range(10)]

for thread in threads:

thread.start()

for thread in threads:

thread.join()

output("Task7")

print("--- %s seconds ---" % (time.time() - start\_time))

Results:

└─$ python file.py

=======TASK 3=======

Finished! Result = 100000

--- 0.3244013786315918 seconds ---

=======TASK 4=======

Finished! Result = 18032

--- 123.84115409851074 seconds ---

=======TASK 5=======

Finished! Result = 100000

--- 352.23577094078064 seconds ---

=======TASK 6=======

Finished! Result = 100000

--- 493.15377020835876 seconds ---

=======TASK 7=======

Finished! Result = 100000

--- 83.88951992988586 seconds ---