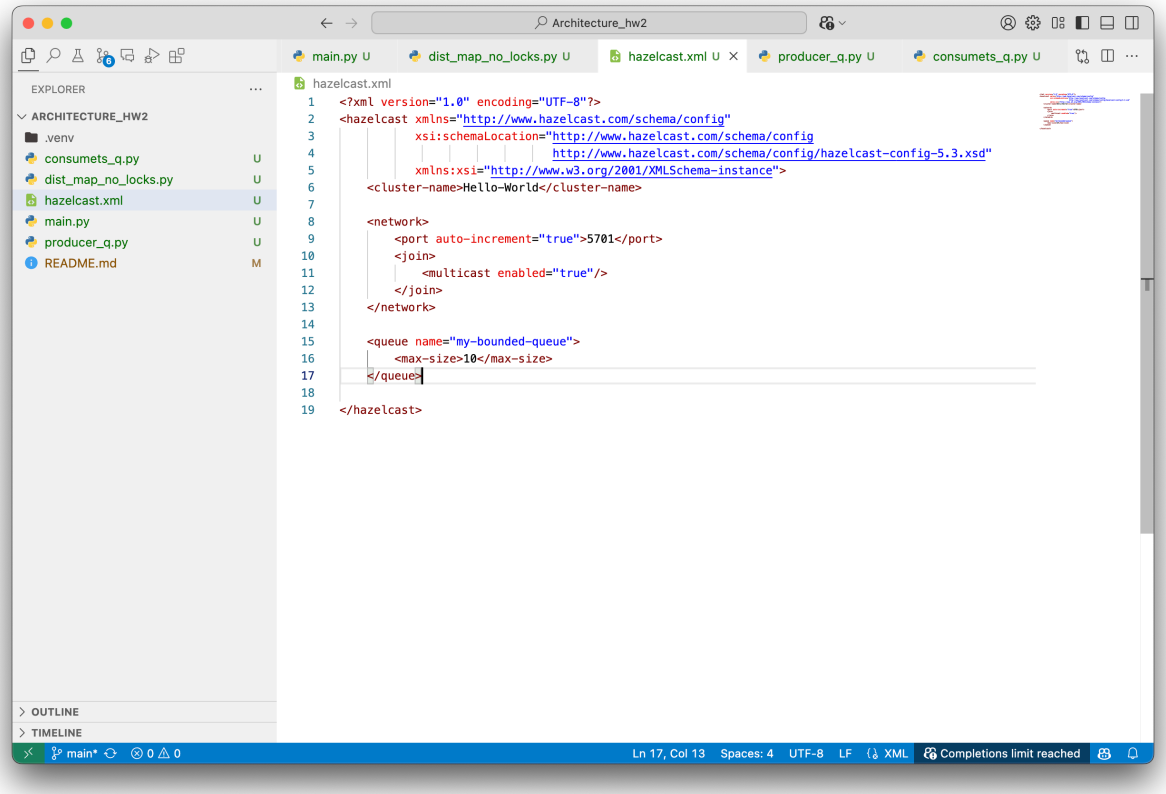
Report HW2 Hazelcast

Author: Bohdan Hashchuk

GitHub: https://github.com/gashchukk/Architecture_hw2.git

1.Встановити і налаштувати Hazelcast

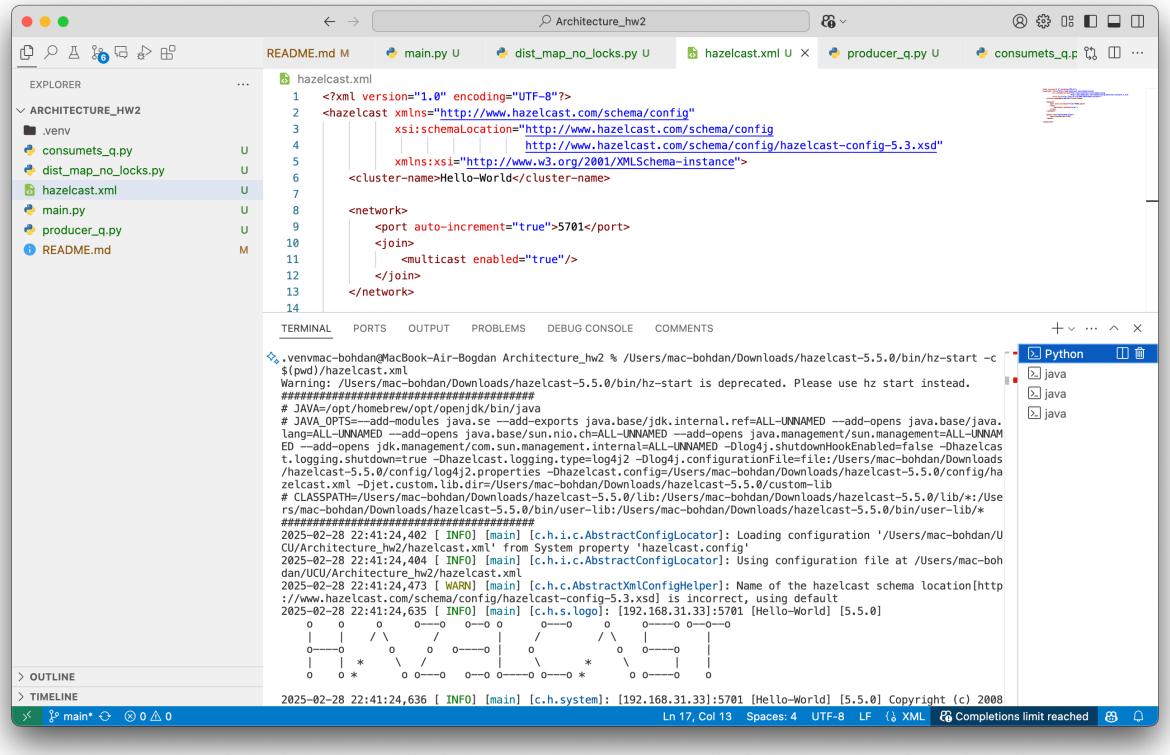


2.Сконфігурувати і запустити 3 ноди (інстанси) об'єднані в кластер

Використовував команду:

```
/Users/mac-bohdan/Downloads/hazelcast-5.5.0/bin/hz-start -c $(pwd)/hazelcas
```

Так як не додав бінарник hazelcast в PATH



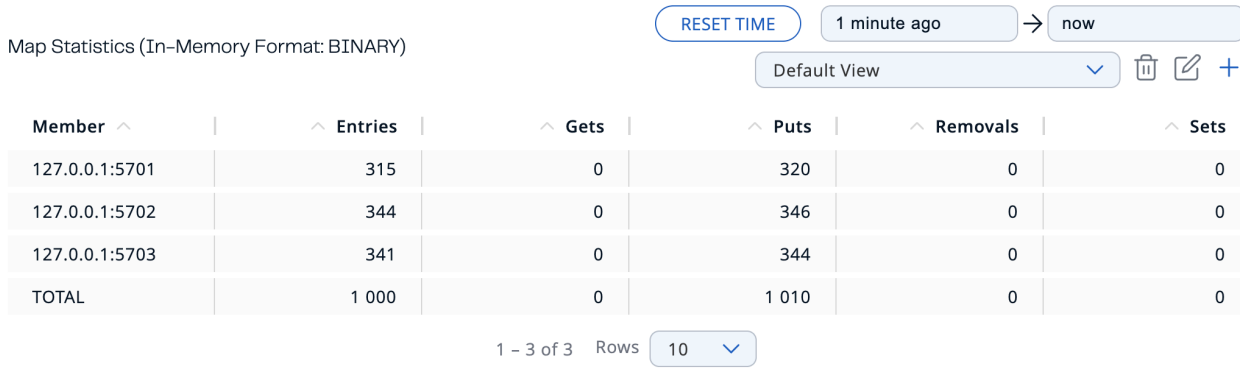
Продемонструйте роботу Distributed Map

Використовуючи API на Python створіть Distributed Map. Запишіть в неї 1000 значень з ключами від 0 до 1000. За допомогою Management Center подивиться на розподіл ключів по нодах. Подивитись як зміниться розподіл даних по нодах:

- якщо відключити одну ноду (результати мають бути у протоколі)
- відключити послідовно дві ноди (результати мають бути у протоколі)
- відключити одночасно дві ноди (емулюючи “падіння” серверів, чи використовуючи команду kill -9) (результати мають бути у протоколі)
- Чи буде втрата даних?
- Яким чином зробити щоб не було втрати даних?

```
import hazelcast

client = hazelcast.HazelcastClient(
    cluster_name="Hello-World",
    cluster_members=["127.0.0.1:5701"]
)

my_map = client.get_map("my-distributed-map").blocking()

for i in range(1000):
    my_map.put(i, f"value- {i}")

for key, val in my_map.entry_set():
    print(key, val)

client.shutdown()
```

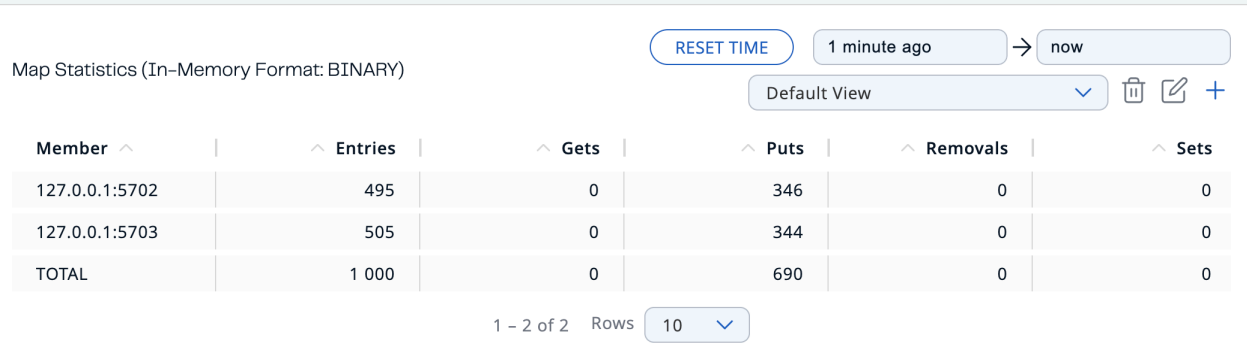
Розподіл по нодах



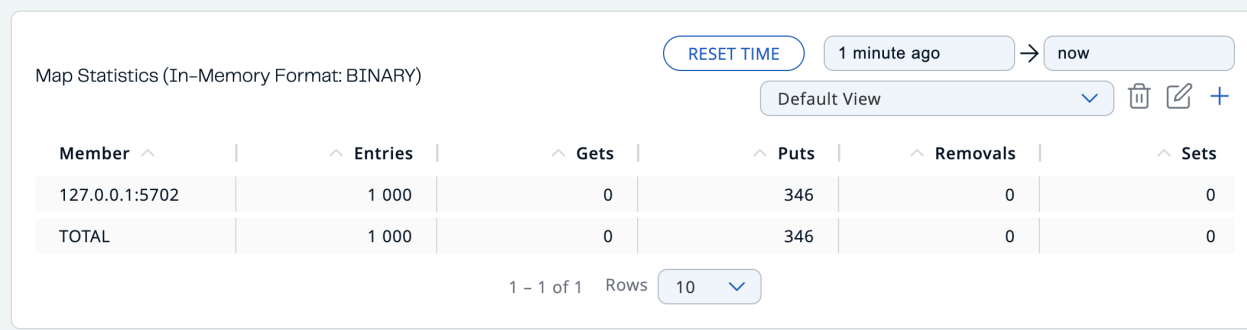
Member	Entries	Gets	Puts	Removals	Sets
127.0.0.1:5701	315	0	320	0	0
127.0.0.1:5702	344	0	346	0	0
127.0.0.1:5703	341	0	344	0	0
TOTAL	1 000	0	1 010	0	0

1 – 3 of 3 Rows 10

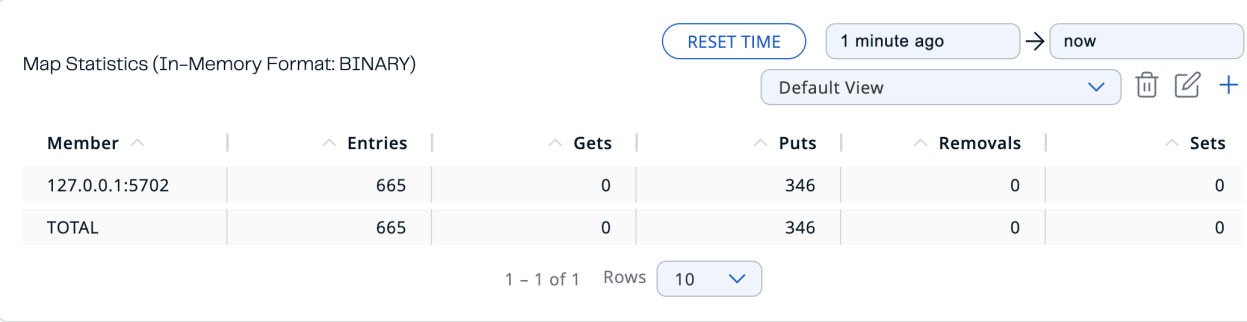
Якщо відключити одну ноду



Якщо відключити дві ноди послідовно



Якщо відключити дві ноди одночасно. Емулюючи падіння



Як можна побачити, у нас є чимала втрата даних. Щоб запобігти цьому, у нашому випадку можна просто не додавати `-9` прапорець до команди `kill` .

Якщо ж по хорошому, то краще зробити synchronization або використовувати transactions.

Продемонструйте роботу Distributed Map without locks

Використовуючи 3 клієнта, на кожному з них одночасно запустіть інкремент значення для одного й того самого ключа в циклі на 10K ітерацій:

```
def no_locks_task(client_id):
    """Increment key 10K times without using locks"""
    client = hazelcast.HazelcastClient(
    cluster_name="Hello-World",
    cluster_members=["127.0.0.1:5701","127.0.0.1:5702","127.0.0.1:5703"]
    )
    distributed_map = client.get_map("distributed-map").blocking()

    for _ in range(10000):
        value = distributed_map.get("key")
        value += 1
        distributed_map.put("key", value)

    print(f"No locks - Client {client_id} finished.")
    client.shutdown()



with ThreadPoolExecutor(max_workers=3) as executor:
        for i in range(3):
          executor.submit(no_locks_task, i+1)
```

Подивиться яке кінцеве значення для ключа “key” буде отримано (чи вийде 30K?)

```
--- Starting no_locks test ---
No locks - Client 2 finished.
No locks - Client 3 finished.
No locks - Client 1 finished.
no_locks - Final value: 13204
no_locks - Expected value (30,000): No
no_locks - Execution time: 5.182 seconds
```

```
--- Comparison Results ---
Mode              Final Value     Time (seconds)
-------------------------------------------------
no_locks          13204           5.182
```

Як бачимо, маємо чималі втрати якщо використовувати no locks. Давайте глянемо на pessimistic та optimistic locks

Зробіть те саме з використанням песимістичним блокування та поміряйте час:

```
def pessimistic_locks_task(client_id):
    """Increment key 10K times using pessimistic locking"""
    client = hazelcast.HazelcastClient(
    cluster_name="Hello-World",
    cluster_members=["127.0.0.1:5701","127.0.0.1:5702","127.0.0.1:5703"]
    )
    distributed_map = client.get_map("distributed-map").blocking()

    for _ in range(10000):
        distributed_map.lock("key")
        try:
            value = distributed_map.get("key")
            value += 1
            distributed_map.put("key", value)
        finally:
            distributed_map.unlock("key")

    print(f"Pessimistic locks - Client {client_id} finished.")
    client.shutdown()

start_time = time.time()

if mode == "no_locks":
```

```
    task_func = no_locks_task
elif mode == "pessimistic_locks":
    task_func = pessimistic_locks_task

with ThreadPoolExecutor(max_workers=3) as executor:
    for i in range(3):
        executor.submit(task_func, i+1)

execution_time = time.time() - start_time
```

Результат:

```
--- Starting no_locks test ---
No locks - Client 2 finished.
No locks - Client 3 finished.
No locks - Client 1 finished.
no_locks - Final value: 13239
no_locks - Expected value (30,000): No
no_locks - Execution time: 4.524 seconds

--- Starting pessimistic_locks test ---
Pessimistic locks - Client 3 finished.
Pessimistic locks - Client 1 finished.
Pessimistic locks - Client 2 finished.
pessimistic_locks - Final value: 30000
pessimistic_locks - Expected value (30,000): Yes
pessimistic_locks - Execution time: 10.393 seconds

--- Comparison Results ---
Mode              Final Value     Time (seconds)
-------------------------------------------------
no_locks           13239          4.524
pessimistic_locks    30000          10.393
```

Як бачимо, pessimistic lock має правильне фінальне значення, але ціною збільшення часу виконання більше ніж у 2 рази.

Зробіть те саме з використанням оптимістичним блокуванням та поміряйте час:

```
def optimistic_locks_task(client_id):
    """Increment key 10K times using optimistic locking"""
    client = hazelcast.HazelcastClient(
    cluster_name="Hello-World",
    cluster_members=["127.0.0.1:5701","127.0.0.1:5702","127.0.0.1:5703"]
    )
    distributed_map = client.get_map("distributed-map").blocking()

    for _ in range(10000):
        while True:
            old_value = distributed_map.get("key")
            new_value = old_value + 1

            if distributed_map.replace_if_same("key", old_value, new_value):
                break

    print(f"Optimistic locks - Client {client_id} finished.")
    client.shutdown()

if mode == "no_locks":
    task_func = no_locks_task
elif mode == "pessimistic_locks":
    task_func = pessimistic_locks_task
else:
    task_func = optimistic_locks_task

with ThreadPoolExecutor(max_workers=3) as executor:
    for i in range(3):
        executor.submit(task_func, i+1)
```

```
execution_time = time.time() - start_time
```

Результат

```
--- Starting no_locks test ---
No locks - Client 3 finished.
No locks - Client 2 finished.
No locks - Client 1 finished.
no_locks - Final value: 13272
no_locks - Expected value (30,000): No
no_locks - Execution time: 4.946 seconds

--- Starting pessimistic_locks test ---
Pessimistic locks - Client 1 finished.
Pessimistic locks - Client 2 finished.
Pessimistic locks - Client 3 finished.
pessimistic_locks - Final value: 30000
pessimistic_locks - Expected value (30,000): Yes
pessimistic_locks - Execution time: 11.408 seconds

--- Starting optimistic_locks test ---
Optimistic locks - Client 3 finished.
Optimistic locks - Client 2 finished.
Optimistic locks - Client 1 finished.
optimistic_locks - Final value: 30000
optimistic_locks - Expected value (30,000): Yes
optimistic_locks - Execution time: 10.083 seconds

--- Comparison Results ---
Mode              Final Value     Time (seconds)
-------------------------------------------------
no_locks           13272          4.946
```

```
pessimistic_locks    30000          11.408
optimistic_locks     30000          10.083
```

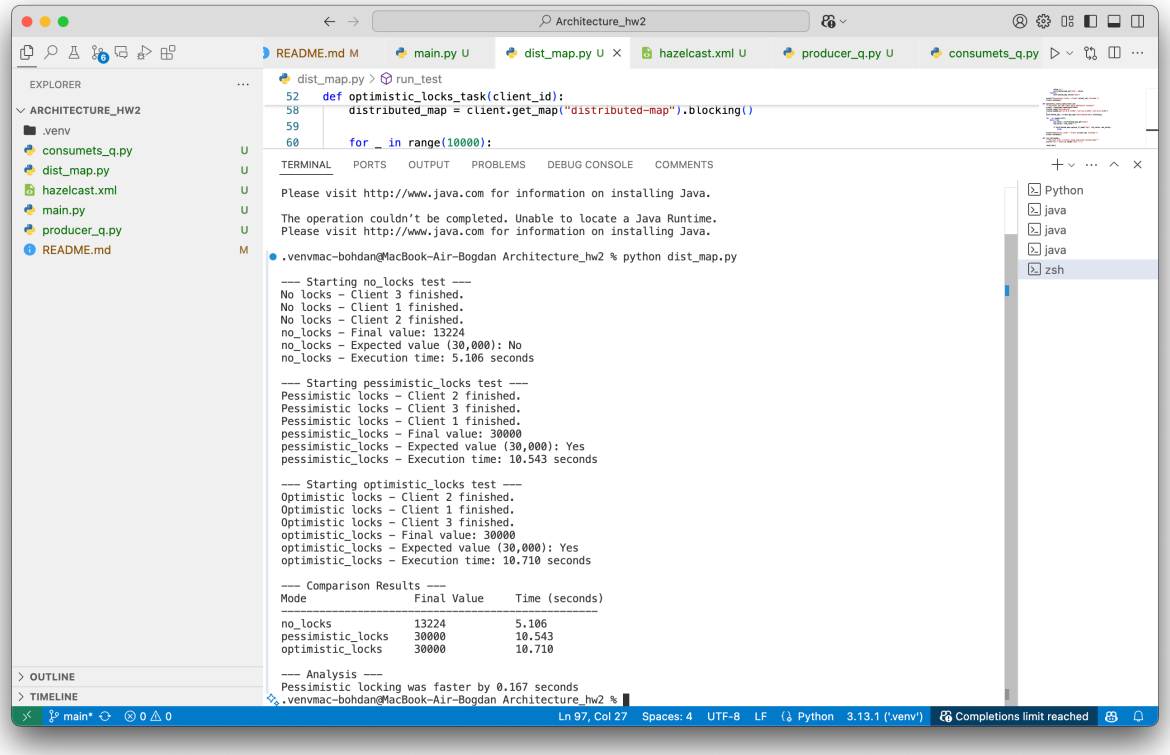
Порівняйте результати кожного з запусків. Песимістичний чи оптимістичний підхід працює швидше?

```
--- Starting no_locks test ---
No locks - Client 3 finished.
No locks - Client 2 finished.
No locks - Client 1 finished.
no_locks - Final value: 13272
no_locks - Expected value (30,000): No
no_locks - Execution time: 4.946 seconds

--- Starting pessimistic_locks test ---
Pessimistic locks - Client 1 finished.
Pessimistic locks - Client 2 finished.
Pessimistic locks - Client 3 finished.
pessimistic_locks - Final value: 30000
pessimistic_locks - Expected value (30,000): Yes
pessimistic_locks - Execution time: 11.408 seconds

--- Starting optimistic_locks test ---
Optimistic locks - Client 3 finished.
Optimistic locks - Client 2 finished.
Optimistic locks - Client 1 finished.
optimistic_locks - Final value: 30000
optimistic_locks - Expected value (30,000): Yes
optimistic_locks - Execution time: 10.083 seconds

--- Comparison Results ---
Mode              Final Value     Time (seconds)
------------------------------------------------
```

```
no_locks            13272          4.946
pessimistic_locks   30000          11.408
optimistic_locks    30000          10.083

--- Analysis ---
Optimistic locking was faster by 1.326 seconds
```

Як бачимо, для `no locks` можна спостерігати втрату даних, але дуже швидке виконання.

`pessimistic lock` та `optimistic lock` вже коректно обраховують результат, проте час виконання був збільшений удвічі, зокрема через блокування.



Робота з Bounded queue

На основі Distributed Queue налаштуйте Bounded queue на 10 елементів

```
<?xml version="1.0" encoding="UTF-8"?>
<hazelcast xmlns="http://www.hazelcast.com/schema/config"
        xsi:schemaLocation="http://www.hazelcast.com/schema/config
                    http://www.hazelcast.com/schema/config/hazelcast-config-5.
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <cluster-name>Hello-World</cluster-name>

   <network>
      <port auto-increment="true">5701</port>
      <join>
         <multicast enabled="true"/>
      </join>
   </network>

   <queue name="my-bounded-queue">
      <max-size>10</max-size>
   </queue>

</hazelcast>
```

Запустіть одного клієнта який буде писати в чергу значення 1..100, а двох інших які будуть читати з черги

- під час вичитування, кожне повідомлення має вичитуватись одразу

Producer

```
import hazelcast

client = hazelcast.HazelcastClient(
   cluster_name="Hello-World",
   cluster_members=["127.0.0.1:5701"]
)
queue = client.get_queue("my-bounded-queue").blocking()

print("\n--- Started writing (1 - 100) ---")
```

```
for i in range(1, 101):
    queue.put(i)
    print(f" - Wrote: {i}")
print("\n--- Writing Finished ---")
client.shutdown()
```

```
.venvmac-bohdan@MacBook-Air-Bogdan Architecture_hw2 % python producer_q.py

--- Started writing (1 - 100) ---
 - Wrote: 1
 - Wrote: 2
 - Wrote: 3
 - Wrote: 4
 - Wrote: 5
 - Wrote: 6
 - Wrote: 7
 - Wrote: 8
 - Wrote: 9
 - Wrote: 10
^[[A - Wrote: 11
 - Wrote: 12
 - Wrote: 13
 - Wrote: 14
 - Wrote: 15
 - Wrote: 16
 - Wrote: 17
 - Wrote: 18
 - Wrote: 19
 - Wrote: 20
 - Wrote: 21
 - Wrote: 22
 - Wrote: 23
```

Consumer:

```
import hazelcast
import time

client = hazelcast.HazelcastClient(
    cluster_name="Hello-World",
    cluster_members=["127.0.0.1:5701"]
)

queue = client.get_queue("my-bounded-queue").blocking()

print("\n--- Started listening for values ---")
while True:
```

```
    item = queue.take()
    print(f" - Read: {item}")
    time.sleep(1)

client.shutdown()
```

```
.venvmac-bohdan@MacBook-Air-Bogdan Architecture_hw2 % python consumets_q.py

--- Started listening for values ---
 - Read: 1
 - Read: 2
 - Read: 3
 - Read: 5
 - Read: 7
 - Read: 9
 - Read: 11
 - Read: 13
 - Read: 15
 - Read: 17
 - Read: 19
 - Read: 21
 - Read: 23
 - Read: 25
 - Read: 27
 - Read: 29
 - Read: 31
 - Read: 33
 - Read: 35
 - Read: 37
 - Read: 39
 - Read: 41
 - Read: 43
 - Read: 45
```

```
(.venvmac-bohdan@MacBook-Air-Bogdan Architecture_hw2 % python consumets_q.py

--- Started listening for values ---
 - Read: 4
 - Read: 6
 - Read: 8
 - Read: 10
 - Read: 12
 - Read: 14
 - Read: 16
 - Read: 18
 - Read: 20
 - Read: 22
 - Read: 24
 - Read: 26
 - Read: 28
 - Read: 30
 - Read: 32
 - Read: 34
 - Read: 36
 - Read: 38
 - Read: 40
 - Read: 42
 - Read: 44
 - Read: 46
 - Read: 48
 - Read: 50
 - Read: 52
```

Яким чином будуть вичитуватись значення з черги двома клієнтами?

Так як Hazelcast використовує модель *distributed queue* , то значення в черзі можуть бути вичитані лише одним клієнтом — після того як клієнт вичитав елемент, він зникає з черги.

Отже, якщо у нас два клієнти-споживачі, вони будуть по черзі витягувати елементи:

- Перший клієнт прочитає наприклад: 1, 3, 5, ...
- Другий клієнт: 2, 4, 6, ...

Перевірте яка буде поведінка на запис якщо відсутнє читання, і черга заповнена

Так як ми виставили `max-size = 10` ,то коли *queqe* досягне цього розміру, операція `put()` заблокується і програма зависне поки не звільниться місце