# Item Rest Service Report

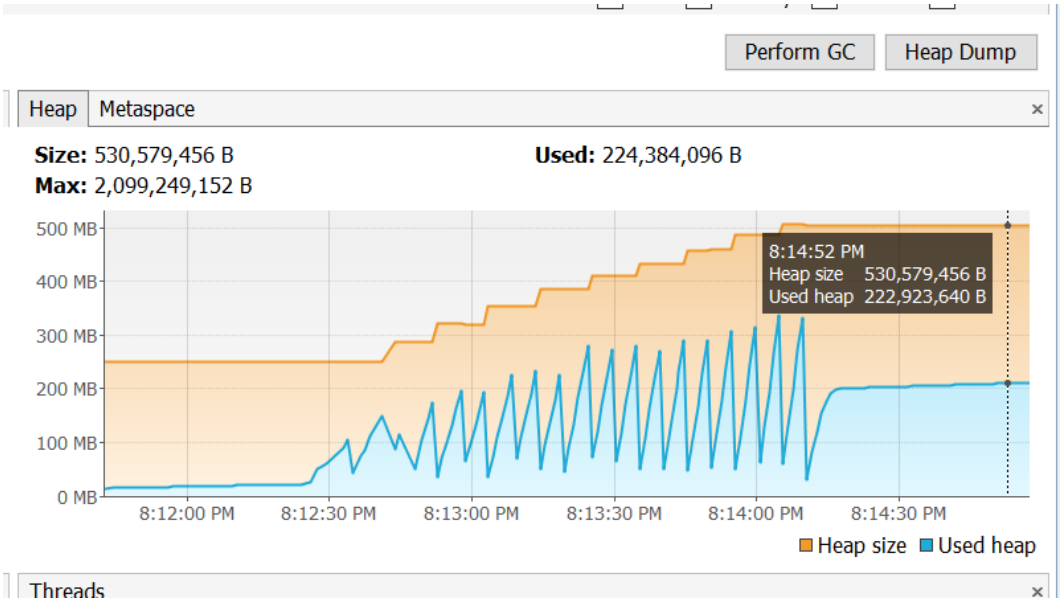
**Task**: Create a REST service that enables user to POST and GET item objects as JSON strings.

**Objectives**:

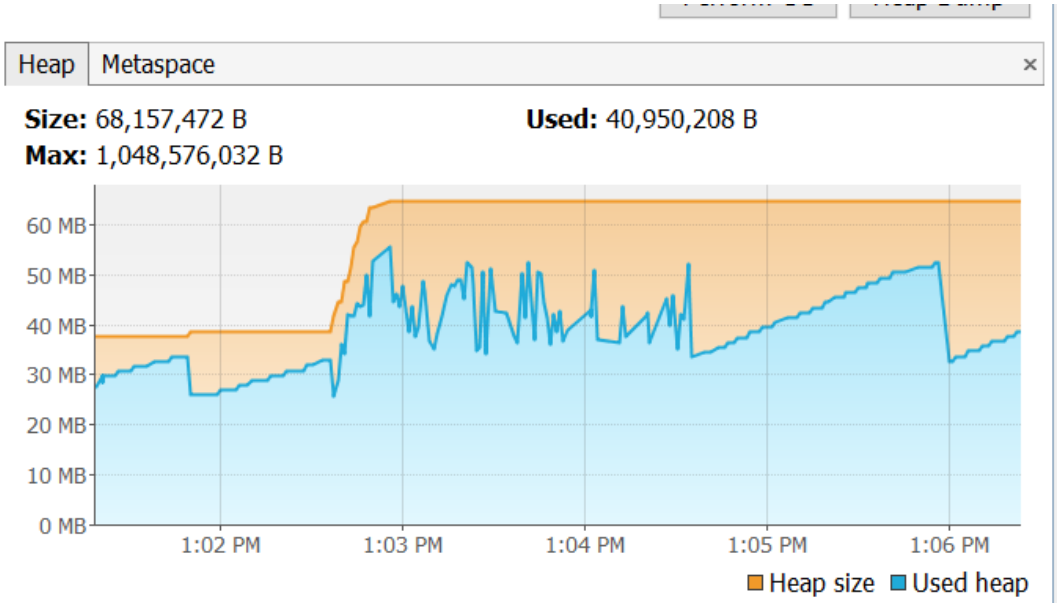
1. Service should have maximum throughput and minimum latency
2. Service should minimize heap memory footprint as much as possible

**Design choices to achieve these objectives**:

1. Maximum throughput and minimum latency
    a. Use of Spring Boot to create a service:
        i. It's very easy and fast to construct a REST service with Spring Boot. Minimum configuration is needed due to annotation – based interface of Spring Boot
        ii. Spring Boot manages the incoming traffic to the REST service via multiple threads. So, performance in terms of throughput and latency is very good.
        iii. I tried developing the service with and without Spring Boot and found that there is not much difference between memory consumption (Spring Boot memory consumption not significantly higher).
2. Minimizing heap memory footprint
    a. Cleaning out unnecessary objects:
        i. Use of ConcurrentSkipListMap to store incoming objects. The reasons are as follows:
            1. Map – because we need to store the time the objects were created. This is important since the problem requirements state that get request should return either the objects created in last 2 seconds or latest 100 objects. Hence the map of <Timestamp, Object>
            2. Concurrent Map – This is because Spring boot uses multiple threads to operate the web service. Hence under heavy load, multiple threads are trying to access the map and get 500 internal server error if concurrency is not maintained.
            3. Skip List Map – It maintains natural order of keys. In our case the key is the timestamp.
        ii. If more than 100 objects are created in last 2 seconds, then we don't need objects older than 2 seconds. Hence, I am deleting those objects from the map. This helps keeping map size under control.
    b. Using G1 garbage collector as opposed to the default garbage collector.
        i. According to Oracle documentation, G1 is specifically designed for multi – threaded server environments like this service that:
            1. Can operate concurrently with applications threads like the CMS collector
            2. Compact free space without lengthy GC induced pause times.
            3. Need more predictable GC pause durations.
            4. Do not want to sacrifice a lot of throughput performance.
            5. Do not require a much larger Java heap
        ii. Here is comparison of Heap consumption under default CMS gc vs G1 gc under load –




Heap footprint using default garbage collector




Heap footprint using G1 garbage collector

3. Performance testing –
   I tested this service performance using Gatling load testing tool. I sent total of 40000 POST and GET requests in span of about 2 minutes to the application and monitored its performance. Even the heap footprint graphs you see above are during those load tests. The Load testing simulation is included in the repository as “LoadTestingSimulation.scala” Here are outputs of the load test (Screenshot on next page):
   a. Request count: 40000
   b. Latency – Mean response time: 5 ms
   c. Throughput – Mean requests per second: 357.143

```
================================================================================

Simulation computerdatabase.LoadTestingSimulation completed in 111 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

================================================================================
---- Global Information --------------------------------------------------------
> request count                                      40000 (OK=40000  KO=0     )
> min response time                                      0 (OK=0      KO=-     )
> max response time                                   1236 (OK=1236   KO=-     )
> mean response time                                     5 (OK=5      KO=-     )
> std deviation                                         35 (OK=35     KO=-     )
> response time 50th percentile                          2 (OK=2      KO=-     )
> response time 75th percentile                          3 (OK=3      KO=-     )
> response time 95th percentile                          6 (OK=6      KO=-     )
> response time 99th percentile                         43 (OK=42     KO=-     )
> mean requests/sec                                357.143 (OK=357.143 KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                         39984 (100%)
> 800 ms < t < 1200 ms                                  15 (  0%)
> t > 1200 ms                                            1 (  0%)
> failed                                                 0 (  0%)
================================================================================

Reports generated in 6s.
Please open the following file: D:\gatling-charts-highcharts-bundle-3.0.0-RC4\results\loadtestingsimulation-2018
Press any key to continue . . .
```