Performance Testing

Testing the performance of the service of the Care web application

Name: Victoria C. A. Fong

Year: 2022

Class: S-A-RB-CMK 4

Introduction

This document summarizes scalability testing done on the Care web application using the K6 scalability tool. The scalability testing is based on testing each microservice (medicine-inventory-service & user-medicine-inventory-service) to evaluate how much users the services can handle. The testing consists of smoke, load, stress and soak testing.

These tests have been conducted on each service running on Azure Kubernetes to examine the performance of the application.

Table of Contents

K6 Scalability Tool

Grafana k6 is an open-source load testing tool to test the performance and reliability of a system and catch performance regressions and problems earlier. The goal of k6 is to help developers build resilient and performant applications that scale.
K6 is used for testing the performance and reliability of API’s, microservices and websites.
Common k6 uses are Load testing (including smoke, load, stress and soak testing), browser testing, chaos and resilience testing, performance and synthetic monitoring.

Types of Performance Testing

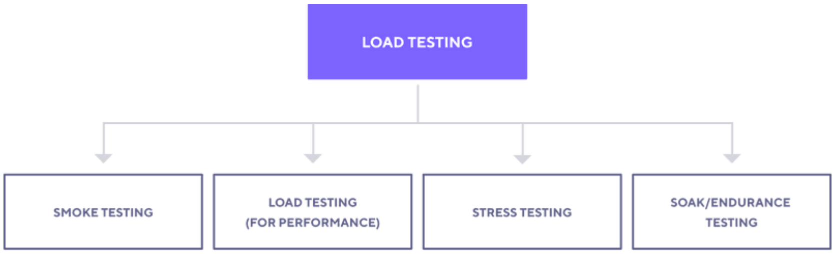


Image 1: "Load Test Layout of what tests it consists of"

Context

The k6 scalability testing tool as indicated in section "k6 Scalability Tool" runs load testing. This testing includes multiple types of testing such as smoke, load, stress, and soak testing. Further in this section you can read more of the different type of testing and their purpose of usage.

Smoke Testing

Context

Smoke tests are to verify that the system can handle minimal load without any issues. Once the smoke test displays zero errors, is the next to carry out a load test.

Load Testing

Context

Load tests evaluate system performance with regards to concurrent users or requests per seconds. The goal of this test is to examine if the system is meeting the set performance goals.
The load test is used establish a system's behavior under both standard and peak state.

Stress Testing

Context

Stress tests estimate the limits and stability of the system under utmost conditions. Stress testing is a kind of load testing used to find out the limits of the system. With the aid of the stress test, we can determine how the system will act under extreme constrain, what the maximal capacity of the system is in terms of users, the breaking point of the system and its failure mode and lastly, if the system will recover without manual intervention after the stress test is over.

Soak Testing

Context

Soak tests determine the reliability and performance of the system over a long duration. The purpose of the soak test is to detect performance and reliability issues derived from the system being under pressure for an extended period.

Smoke Testing

Context

This smoke test is based on looping 1 user for 1 minute to verify that the service does not throw any errors when under minimal load.

Virtual Users (VU) chart

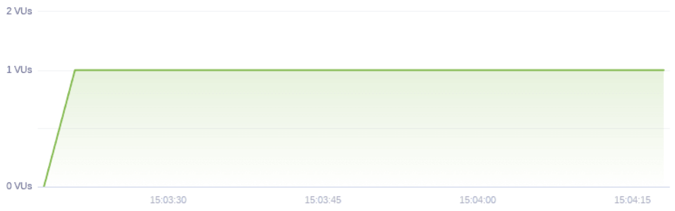


Image 2: "Virtual User Chart of the Smoke Test"

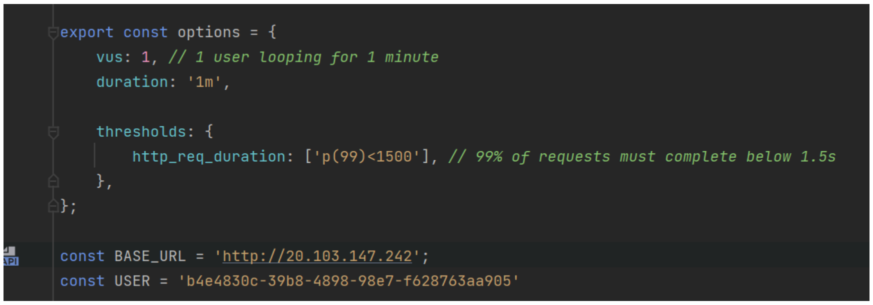
The VU chart of a smoke tests that is used for the service should look similar to image 2 when using 1 user.

We will be creating and testing two smoke tests on the medicine inventory and user medicine inventory service of the care web application.

User Medicine Service

For the user medicine inventory service, we will be testing the minimal amount of users the service can handle under a (GET) request.

Scenario

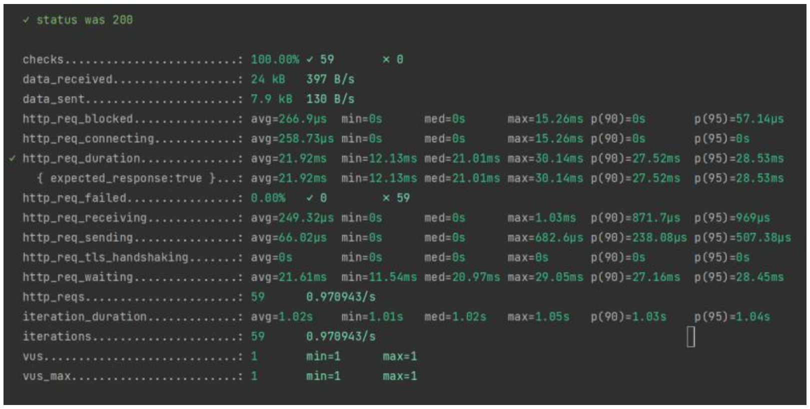


```
export const options = {
    vus: 1, // 1 user looping for 1 minute
    duration: '1m',

    thresholds: {
        http_req_duration: ['p(99)<1500'], // 99% of requests must complete below 1.5s
    },
};

const BASE_URL = 'http://20.103.147.242';
const USER = 'b4e4830c-39b8-4898-98e7-f628763aa905'
```

Image 3: "set criteria for the smoke test for the user medicine inventory service"

The test is based on how long one user can perform a get request to see what medicines are appointed to them. We will be using the get request of this URL http://20.103.147.242/prescription/user/b4e4830c-39b8-4898-98e7-f628763aa905 for the user with the id of b4e4830c-39b8-4898-98e7-f628763aa905 running on Azure Kubernetes.
The prerequisite for the test to pass is that the duration of the request from sending to retrieving the data should take 1.5 s of 99% of the requests sent.

Summary



```
  ✓ status was 200

  checks.........................: 100.00% ✓ 59       × 0
  data_received..................: 24 kB   397 B/s
  data_sent......................: 7.9 kB  130 B/s
  http_req_blocked...............: avg=266.9µs  min=0s      med=0s      max=15.26ms p(90)=0s       p(95)=57.14µs
  http_req_connecting............: avg=258.73µs min=0s      med=0s      max=15.26ms p(90)=0s       p(95)=0s
✓ http_req_duration..............: avg=21.92ms  min=12.13ms med=21.01ms max=30.14ms p(90)=27.52ms  p(95)=28.53ms
    { expected_response:true }...: avg=21.92ms  min=12.13ms med=21.01ms max=30.14ms p(90)=27.52ms  p(95)=28.53ms
  http_req_failed................: 0.00%   ✓ 0        × 59
  http_req_receiving.............: avg=249.32µs min=0s      med=0s      max=1.03ms  p(90)=871.7µs  p(95)=969µs
  http_req_sending...............: avg=66.02µs  min=0s      med=0s      max=682.6µs p(90)=238.08µs p(95)=507.38µs
  http_req_tls_handshaking.......: avg=0s       min=0s      med=0s      max=0s      p(90)=0s       p(95)=0s
  http_req_waiting...............: avg=21.61ms  min=11.54ms med=20.97ms max=29.05ms p(90)=27.16ms  p(95)=28.45ms
  http_reqs......................: 59      0.970943/s
  iteration_duration.............: avg=1.02s    min=1.01s   med=1.02s   max=1.05s   p(90)=1.03s    p(95)=1.04s
  iterations.....................: 59      0.970943/s
  vus............................: 1       min=1      max=1
  vus_max........................: 1       min=1      max=1
```

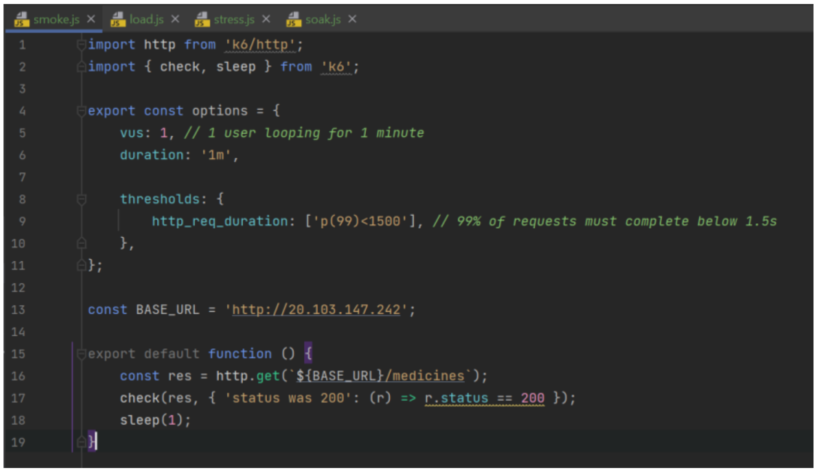
Image 4: "results of the smoke test for the user medicine inventory service"

As you can see from the above photo, the user medicine inventory service passes the minimal amount of users it can handle based on the set criteria.

Medicine Inventory Service

For the medicine inventory service, we will be testing the minimal amount of users the service can handle under a (GET) request.

Scenario



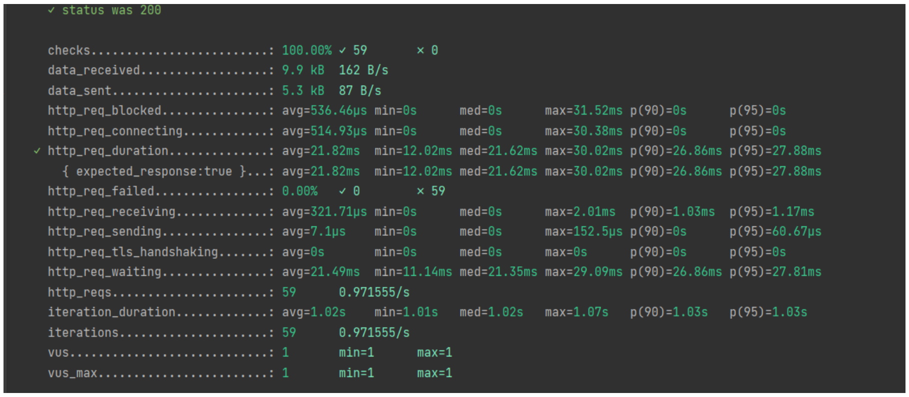
```
import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
    vus: 1, // 1 user looping for 1 minute
    duration: '1m',

    thresholds: {
        http_req_duration: ['p(99)<1500'], // 99% of requests must complete below 1.5s
    },
};

const BASE_URL = 'http://20.103.147.242';

export default function () {
    const res = http.get(`${BASE_URL}/medicines`);
    check(res, { 'status was 200': (r) => r.status == 200 });
    sleep(1);
}
```

Image 5: “set criteria for the smoke test for the medicine inventory service”

The test is based on how long a user can perform a GET request to retrieving all medicines registered in the application. We will be using the get request of this URL http://20.103.147.242/medicines running on Azure Kubernetes.

The prerequisite for the test to pass is that the duration of the request from sending to retrieving the data should take 1.5 s of 99% of the requests sent.

Summary



```
     ✓ status was 200

     checks.........................: 100.00% ✓ 59       × 0
     data_received..................: 9.9 kB  162 B/s
     data_sent......................: 5.3 kB  87 B/s
     http_req_blocked...............: avg=536.46µs min=0s      med=0s      max=31.52ms p(90)=0s      p(95)=0s
     http_req_connecting............: avg=514.93µs min=0s      med=0s      max=30.38ms p(90)=0s      p(95)=0s
   ✓ http_req_duration..............: avg=21.82ms  min=12.02ms med=21.62ms max=30.02ms p(90)=26.86ms p(95)=27.88ms
       { expected_response:true }...: avg=21.82ms  min=12.02ms med=21.62ms max=30.02ms p(90)=26.86ms p(95)=27.88ms
     http_req_failed................: 0.00%   ✓ 0        × 59
     http_req_receiving.............: avg=321.71µs min=0s      med=0s      max=2.01ms  p(90)=1.03ms  p(95)=1.17ms
     http_req_sending...............: avg=7.1µs    min=0s      med=0s      max=152.5µs p(90)=0s      p(95)=60.67µs
     http_req_tls_handshaking.......: avg=0s       min=0s      med=0s      max=0s      p(90)=0s      p(95)=0s
     http_req_waiting...............: avg=21.49ms  min=11.14ms med=21.35ms max=29.09ms p(90)=26.86ms p(95)=27.81ms
     http_reqs......................: 59      0.971555/s
     iteration_duration.............: avg=1.02s    min=1.01s   med=1.02s   max=1.07s   p(90)=1.03s   p(95)=1.03s
     iterations.....................: 59      0.971555/s
     vus............................: 1       min=1      max=1
     vus_max........................: 1       min=1      max=1
```

Image 6: “results of the smoke test for the medicine inventory service”

As you can see from the above photo, the medicine inventory service passes the minimal amount of users it can handle based on the set criteria.

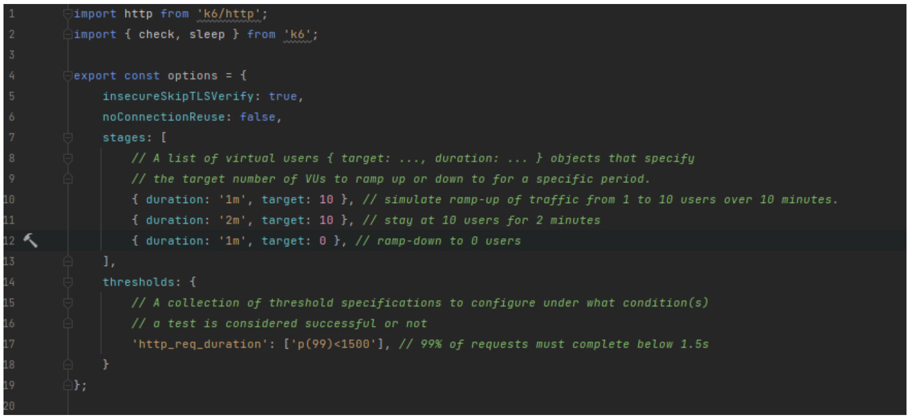
Load Testing

Criteria

We will set stages to ramping up the number of users for the service and observing the performance to how well the service can handle the amount of users for a duration of a specific time.
We will be trying out 3 stages of ramping.

The expectation of each of the stages are:

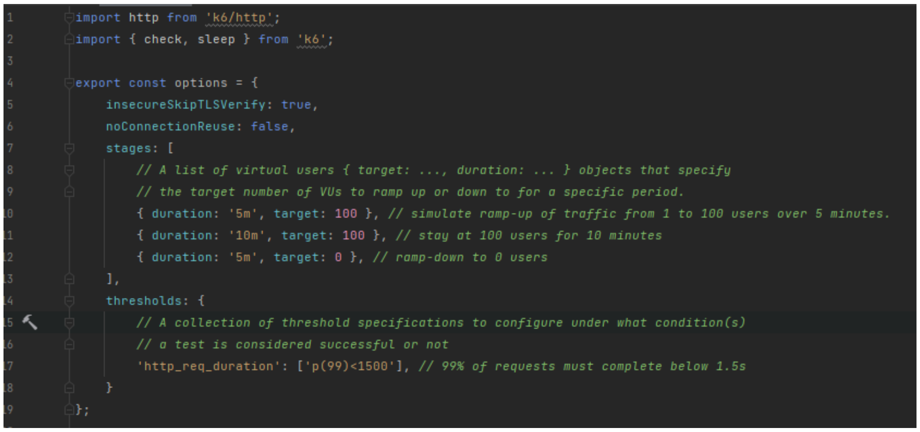
- 99% of requests should finish within 5 seconds.
- 95% of requests should finish within 1 second.



```javascript
import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
    insecureSkipTLSVerify: true,
    noConnectionReuse: false,
    stages: [
        // A list of virtual users { target: ..., duration: ... } objects that specify
        // the target number of VUs to ramp up or down to for a specific period.
        { duration: '1m', target: 10 }, // simulate ramp-up of traffic from 1 to 10 users over 10 minutes.
        { duration: '2m', target: 10 }, // stay at 10 users for 2 minutes
        { duration: '1m', target: 0 }, // ramp-down to 0 users
    ],
    thresholds: {
        // A collection of threshold specifications to configure under what condition(s)
        // a test is considered successful or not
        'http_req_duration': ['p(99)<1500'], // 99% of requests must complete below 1.5s
    }
};
```

Image 7: "Load Test stage 1 set targets"

Stage 1 will consist of ramping up the number of users to 10 in 1 minute, staying at 10 users for 2 minutes and then ramping down to 0 users in a span of 1 min. as shown in image 7. The purpose of this stage is to see if the system can handle a small amount of load.



```javascript
import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
    insecureSkipTLSVerify: true,
    noConnectionReuse: false,
    stages: [
        // A list of virtual users { target: ..., duration: ... } objects that specify
        // the target number of VUs to ramp up or down to for a specific period.
        { duration: '5m', target: 100 }, // simulate ramp-up of traffic from 1 to 100 users over 5 minutes.
        { duration: '10m', target: 100 }, // stay at 100 users for 10 minutes
        { duration: '5m', target: 0 }, // ramp-down to 0 users
    ],
    thresholds: {
        // A collection of threshold specifications to configure under what condition(s)
        // a test is considered successful or not
        'http_req_duration': ['p(99)<1500'], // 99% of requests must complete below 1.5s
    }
};
```

Image 8: "Load Test stage 2 set targets"

Stage 2 will consist of ramping up the number of users to 100 in 5 minutes, staying at 100 users for 10 minutes and then ramping down to 0 users in a span of 5 min. as shown in image 8. Where 100 users are our peak hours which we will be assessing. The purpose of this stage is to see if the system can handle the normal amount of load of users they receive on an average.

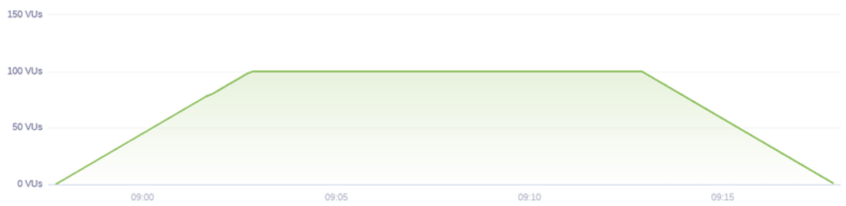
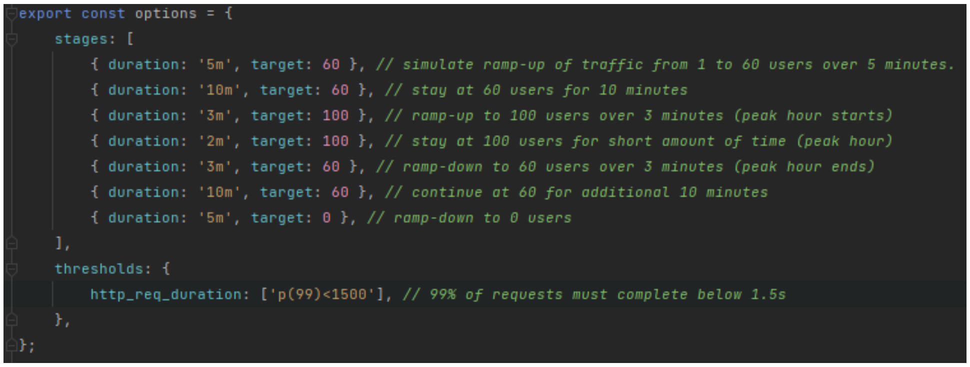


Image 9: “Virtual User Chart of stage 2 Load Test”

The VU chart of stage 2 can look similar to image 9.



```
export const options = {
    stages: [
        { duration: '5m', target: 60 }, // simulate ramp-up of traffic from 1 to 60 users over 5 minutes.
        { duration: '10m', target: 60 }, // stay at 60 users for 10 minutes
        { duration: '3m', target: 100 }, // ramp-up to 100 users over 3 minutes (peak hour starts)
        { duration: '2m', target: 100 }, // stay at 100 users for short amount of time (peak hour)
        { duration: '3m', target: 60 }, // ramp-down to 60 users over 3 minutes (peak hour ends)
        { duration: '10m', target: 60 }, // continue at 60 for additional 10 minutes
        { duration: '5m', target: 0 }, // ramp-down to 0 users
    ],
    thresholds: {
        http_req_duration: ['p(99)<1500'], // 99% of requests must complete below 1.5s
    },
};
```

Image 10: “Load Test stage 3 set targets”

Stage 3 is a step further of step 2. Stage 3 is to resemble a normal and peak conditions normally for the services. Here we will configure a load test to stay at 60 users for most of the day which is the average amount of users which have been determined to visit the site. And then ramp up to 100 users during the peak hours and then ramp back down to normal load. The purpose of this stage is to see if the system can handle an average amount of users they receive on an average including peak hours.

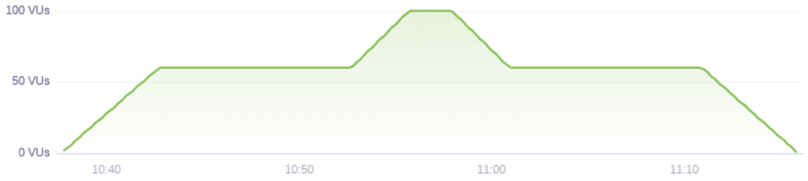


Image 11: “Virtual User Chart of stage 3 Load Test”

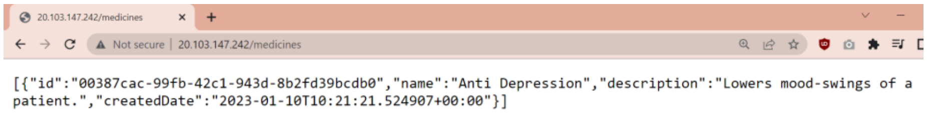
The VU chart of stage 3 can look similar to image 11.

We will be creating and testing the 3 set stages of load tests on the medicine inventory service and not the user medicine inventory service of the care web application due time limitation.

Performance Testing

Medicine Service

Request View

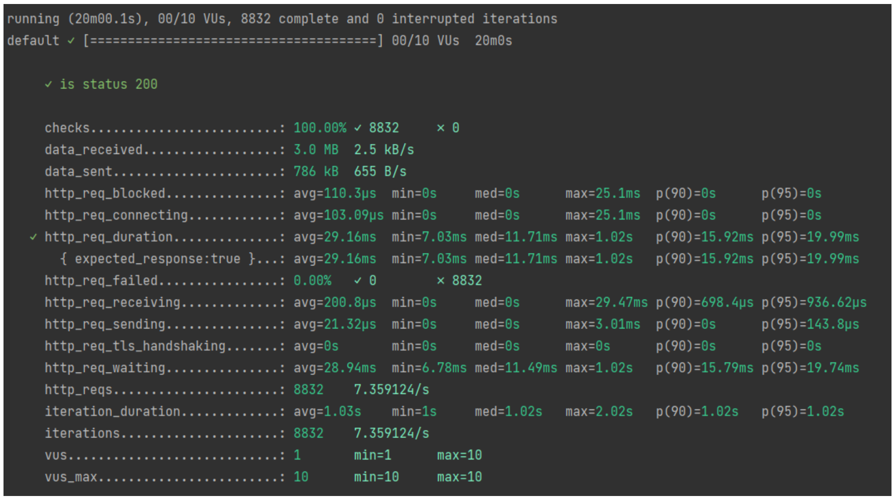


```
[{"id":"00387cac-99fb-42c1-943d-8b2fd39bcdb0","name":"Anti Depression","description":"Lowers mood-swings of a
patient.","createdDate":"2023-01-10T10:21:21.524907+00:00"}]
```

Image 12: “Results of stage 2 load testing on the medicine inventory service”

The request that will be tested on the medicine service for the load test is a get request of one registered medicine that is shown in image 12.

Stage 1 results



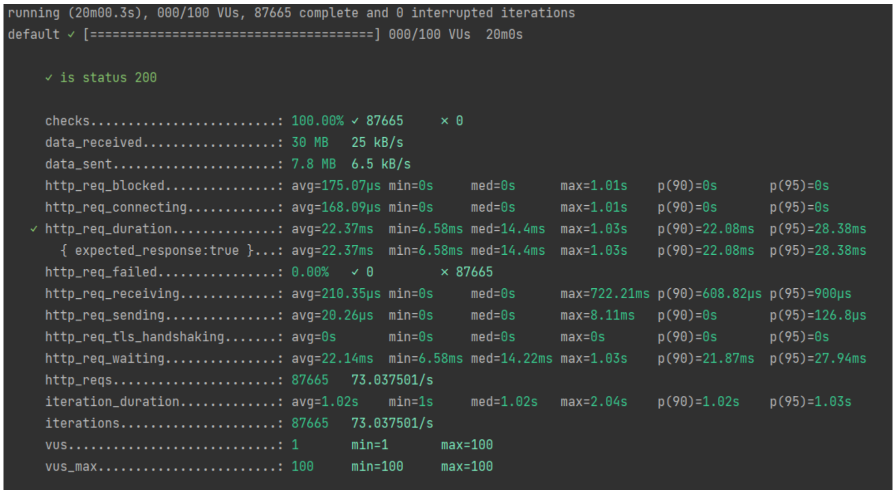
```
running (20m00.1s), 00/10 VUs, 8832 complete and 0 interrupted iterations
default ✓ [======================================] 00/10 VUs  20m0s


     ✓ is status 200


     checks.........................: 100.00% ✓ 8832      × 0
     data_received..................: 3.0 MB  2.5 kB/s
     data_sent......................: 786 kB  655 B/s
     http_req_blocked...............: avg=110.3µs  min=0s      med=0s       max=25.1ms   p(90)=0s       p(95)=0s
     http_req_connecting............: avg=103.09µs min=0s      med=0s       max=25.1ms   p(90)=0s       p(95)=0s
   ✓ http_req_duration..............: avg=29.16ms  min=7.03ms  med=11.71ms  max=1.02s    p(90)=15.92ms  p(95)=19.99ms
       { expected_response:true }...: avg=29.16ms  min=7.03ms  med=11.71ms  max=1.02s    p(90)=15.92ms  p(95)=19.99ms
     http_req_failed................: 0.00%   ✓ 0         × 8832
     http_req_receiving.............: avg=200.8µs  min=0s      med=0s       max=29.47ms  p(90)=698.4µs  p(95)=936.62µs
     http_req_sending...............: avg=21.32µs  min=0s      med=0s       max=3.01ms   p(90)=0s       p(95)=143.8µs
     http_req_tls_handshaking.......: avg=0s       min=0s      med=0s       max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=28.94ms  min=6.78ms  med=11.49ms  max=1.02s    p(90)=15.79ms  p(95)=19.74ms
     http_reqs......................: 8832    7.359124/s
     iteration_duration.............: avg=1.03s    min=1s      med=1.02s    max=2.02s    p(90)=1.02s    p(95)=1.02s
     iterations.....................: 8832    7.359124/s
     vus............................: 1       min=1       max=10
     vus_max........................: 10      min=10      max=10
```

Image 13: “Results of stage 1 load testing on the medicine inventory service”

Summary

As you can see from the above photo, the user medicine service passes the stage 1 load test of handling a ramping to 10 users with an average of 29.16 ms for the duration of handling requests.

Stage 2 results



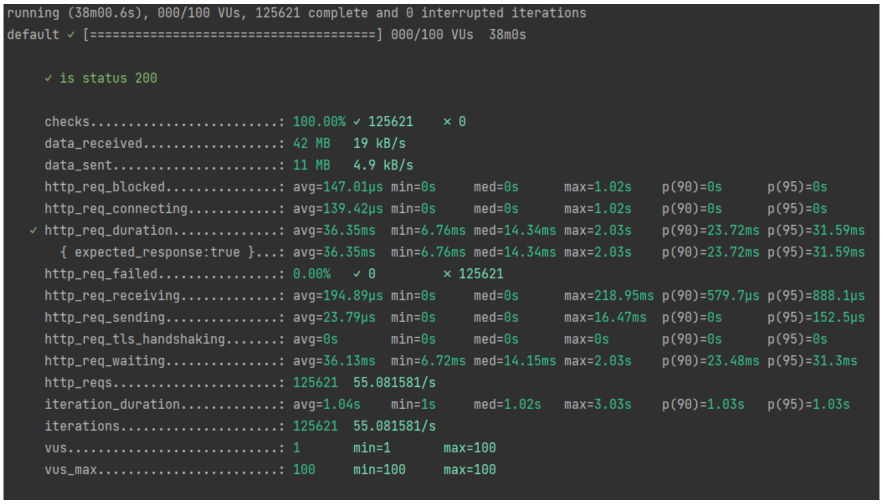
```
running (20m00.3s), 000/100 VUs, 87665 complete and 0 interrupted iterations
default ✓ [======================================] 000/100 VUs  20m0s

     ✓ is status 200

     checks.........................: 100.00% ✓ 87665     × 0
     data_received..................: 30 MB   25 kB/s
     data_sent......................: 7.8 MB  6.5 kB/s
     http_req_blocked...............: avg=175.07µs min=0s     med=0s      max=1.01s    p(90)=0s       p(95)=0s
     http_req_connecting............: avg=168.09µs min=0s     med=0s      max=1.01s    p(90)=0s       p(95)=0s
   ✓ http_req_duration..............: avg=22.37ms  min=6.58ms med=14.4ms  max=1.03s    p(90)=22.08ms  p(95)=28.38ms
       { expected_response:true }...: avg=22.37ms  min=6.58ms med=14.4ms  max=1.03s    p(90)=22.08ms  p(95)=28.38ms
     http_req_failed................: 0.00%   ✓ 0         × 87665
     http_req_receiving.............: avg=210.35µs min=0s     med=0s      max=722.21ms p(90)=608.82µs p(95)=900µs
     http_req_sending...............: avg=20.26µs  min=0s     med=0s      max=8.11ms   p(90)=0s       p(95)=126.8µs
     http_req_tls_handshaking.......: avg=0s       min=0s     med=0s      max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=22.14ms  min=6.58ms med=14.22ms max=1.03s    p(90)=21.87ms  p(95)=27.94ms
     http_reqs......................: 87665   73.037501/s
     iteration_duration.............: avg=1.02s    min=1s     med=1.02s   max=2.04s    p(90)=1.02s    p(95)=1.03s
     iterations.....................: 87665   73.037501/s
     vus............................: 1       min=1       max=100
     vus_max........................: 100     min=100     max=100
```

Image 14: “Results of stage 2 load testing on the medicine inventory service”

Summary

As can be seen from the results from the image 14. The service passed all checks of handling requests under the threshold that was set at 1.5 s with an average of 22.37 ms for the duration of handling requests.

Stage 3 results



```
running (38m00.6s), 000/100 VUs, 125621 complete and 0 interrupted iterations
default ✓ [======================================] 000/100 VUs  38m0s

     ✓ is status 200

     checks.........................: 100.00% ✓ 125621    × 0
     data_received..................: 42 MB   19 kB/s
     data_sent......................: 11 MB   4.9 kB/s
     http_req_blocked...............: avg=147.01µs min=0s     med=0s      max=1.02s    p(90)=0s       p(95)=0s
     http_req_connecting............: avg=139.42µs min=0s     med=0s      max=1.02s    p(90)=0s       p(95)=0s
   ✓ http_req_duration..............: avg=36.35ms  min=6.76ms med=14.34ms max=2.03s    p(90)=23.72ms  p(95)=31.59ms
       { expected_response:true }...: avg=36.35ms  min=6.76ms med=14.34ms max=2.03s    p(90)=23.72ms  p(95)=31.59ms
     http_req_failed................: 0.00%   ✓ 0         × 125621
     http_req_receiving.............: avg=194.89µs min=0s     med=0s      max=218.95ms p(90)=579.7µs  p(95)=888.1µs
     http_req_sending...............: avg=23.79µs  min=0s     med=0s      max=16.47ms  p(90)=0s       p(95)=152.5µs
     http_req_tls_handshaking.......: avg=0s       min=0s     med=0s      max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=36.13ms  min=6.72ms med=14.15ms max=2.03s    p(90)=23.48ms  p(95)=31.3ms
     http_reqs......................: 125621  55.081581/s
     iteration_duration.............: avg=1.04s    min=1s     med=1.02s   max=3.03s    p(90)=1.03s    p(95)=1.03s
     iterations.....................: 125621  55.081581/s
     vus............................: 1       min=1       max=100
     vus_max........................: 100     min=100     max=100
```

Image 15: “Results of stage 3 load testing on the medicine inventory service”

Summary

As can be seen from the results from the image 15. The service passed all checks of handling requests under the threshold that was set at 1.5 s with an average of 36.35 ms for the duration of handling requests.

Stress Testing

Context

Here we will be using stress testing to assess the availability and stability of the system under heavy load.

The purpose of performing the stress test is to find out what happens when pushing the performance limits of the system. Further, it is to help/prove if they system is configured to auto-scale when the load increases and to look for any failures during scaling events.

Scenario

A scenario to where a stress can occur to the system is for instance an outbreak to where medicine would have or can be tracked by multiple users.

We will be increasing the load of users by 100 every 2 minutes and will at this each level for 5 min. further, a recovery stage has been included to where users gradually decreases to 0.

Virtual Users (VU) chart

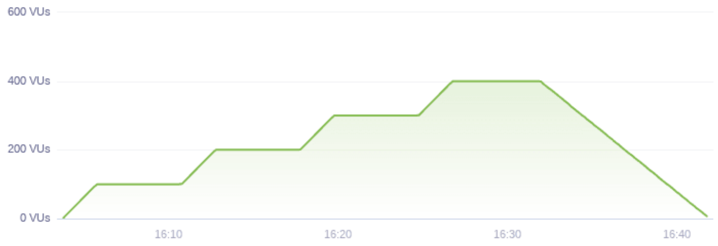


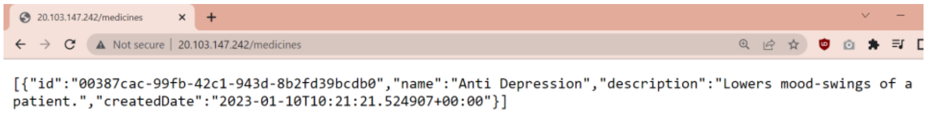
Image 16: “Virtual User Chart of the Stress Test”

The VU chart of the stress test that will be used for the service should look similar to image 16.

We will be creating and testing the 3 set stages of load tests on the medicine inventory service and not the user medicine inventory service of the care web application due time limitation.

Medicine Service

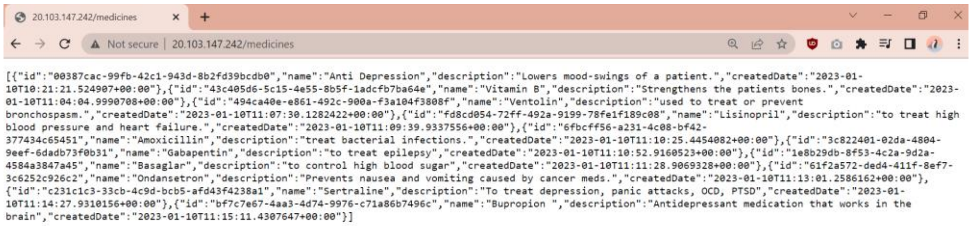
For the medicine service we will be testing how the service handles a high amount of requests under a short amount of time.



```
[{"id":"00387cac-99fb-42c1-943d-8b2fd39bcdb0","name":"Anti Depression","description":"Lowers mood-swings of a
patient.","createdDate":"2023-01-10T10:21:21.524907+00:00"}]
```

Image 17: “Results of stage 2 load testing on the medicine inventory service”

We will be testing the service twice to compare the difference between how big the request is. The first test will test a request with one medicine created as shown in image 17.



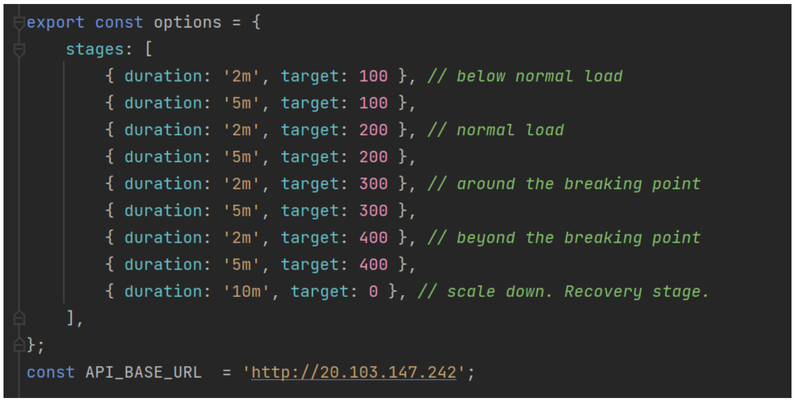
```
[{"id":"00387cac-99fb-42c1-943d-8b2fd39bcdb0","name":"Anti Depression","description":"Lowers mood-swings of a patient.","createdDate":"2023-01-
10T10:21:21.524907+00:00"},{"id":"43c405d6-5c15-4e55-8b5f-1adcfb7ba64e","name":"Vitamin B","description":"Strengthens the patients bones.","createdDate":"2023-
01-10T11:04:04.9990708+00:00"},{"id":"494ca40e-e861-492c-900a-f3a104f3808f","name":"Ventolin","description":"used to treat or prevent
bronchospasm.","createdDate":"2023-01-10T11:07:30.1282422+00:00"},{"id":"fd8cd054-72ff-492a-9199-78fe1f189c08","name":"Lisinopril","description":"to treat high
blood pressure and heart failure.","createdDate":"2023-01-10T11:09:39.9337556+00:00"},{"id":"6fbcff56-a231-4c08-bf42-
377434c65451","name":"Amoxicillin","description":"treat bacterial infections.","createdDate":"2023-01-10T11:10:25.4454082+00:00"},{"id":"3c822401-02da-4804-
9eef-6dadb73f0b31","name":"Gabapentin","description":"to treat epilepsy","createdDate":"2023-01-10T11:10:52.9160523+00:00"},{"id":"1e8b29db-8f53-4c2a-9d2a-
4584a3847a45","name":"Basaglar","description":"to control high blood sugar","createdDate":"2023-01-10T11:11:28.9069328+00:00"},{"id":"61f2a572-ded4-411f-8ef7-
3c6252c926c2","name":"Ondansetron","description":"Prevents nausea and vomiting caused by cancer meds.","createdDate":"2023-01-10T11:13:01.2586162+00:00"},
{"id":"c231c1c3-33cb-4c9d-bcb5-afd43f4238a1","name":"Sertraline","description":"To treat depression, panic attacks, OCD, PTSD","createdDate":"2023-01-
10T11:14:27.9310156+00:00"},{"id":"bf7c7e67-4aa3-4d74-9976-c71a86b7496c","name":"Bupropion ","description":"Antidepressant medication that works in the
brain","createdDate":"2023-01-10T11:15:11.4307647+00:00"}]
```

Image 18: “Results of stage 2 load testing on the medicine inventory service”

And the second test will test a request holding 10 registered medicines as shown in image 18.

Scenario

The test is based on how the service can handle a heavy amount of request under a short amount of time. We will be using the get request of this endpoint http://20.103.147.242/medicines to retrieve all registered medicines in the system.



```
export const options = {
    stages: [
        { duration: '2m', target: 100 }, // below normal load
        { duration: '5m', target: 100 },
        { duration: '2m', target: 200 }, // normal load
        { duration: '5m', target: 200 },
        { duration: '2m', target: 300 }, // around the breaking point
        { duration: '5m', target: 300 },
        { duration: '2m', target: 400 }, // beyond the breaking point
        { duration: '5m', target: 400 },
        { duration: '10m', target: 0 }, // scale down. Recovery stage.
    ],
};
const API_BASE_URL  = 'http://20.103.147.242';
```

Image 19: “stress test that will be tested on the medicine inventory service”

The set criteria can be seen in image 19.

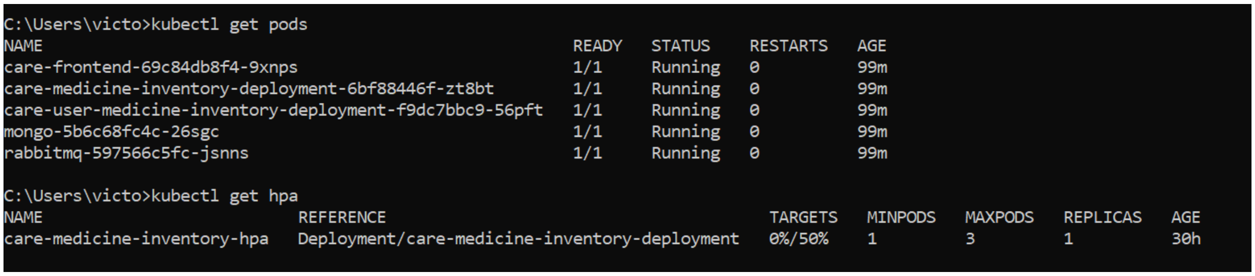
1. Below normal load: increasing to 100 users in 2 minutes and holding this for 5 minutes.
2. Normal load: increase the amount of users to 200 users in 2 minutes and hold this for another 5 minutes.
3. Around the breaking point: increase the amount of users to 300 users and holding this for 5 minutes.
4. Beyond the breaking point: increasing the amount of users to 400 users and holding for 5 minutes.
5. Scaling down / Recovery stage: scaling the amount of users gracefully to 0 users in a span of 10 minutes.

Summary / Results of Scaling for a request holding one registered medicine

We will be examining if the service scales when the amount of CPU increases over the set CPU usage for one pod. We will be assessing each target of virtual users to examine the CPU usage of the service pod.

Start

The start of the test we can verify that the there is nothing running in the portal and that the CPU usage is at 0%.



```
C:\Users\victo>kubectl get pods
NAME                                                READY   STATUS    RESTARTS   AGE
care-frontend-69c84db8f4-9xnps                      1/1     Running   0          99m
care-medicine-inventory-deployment-6bf88446f-zt8bt  1/1     Running   0          99m
care-user-medicine-inventory-deployment-f9dc7bbc9-56pft  1/1     Running   0          99m
mongo-5b6c68fc4c-26sgc                              1/1     Running   0          99m
rabbitmq-597566c5fc-jsnns                           1/1     Running   0          99m

C:\Users\victo>kubectl get hpa
NAME                        REFERENCE                                       TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
care-medicine-inventory-hpa Deployment/care-medicine-inventory-deployment   0%/50%    1         3         1          30h
```

Image 20: “CPU usage and pods running at the start of the stress test”

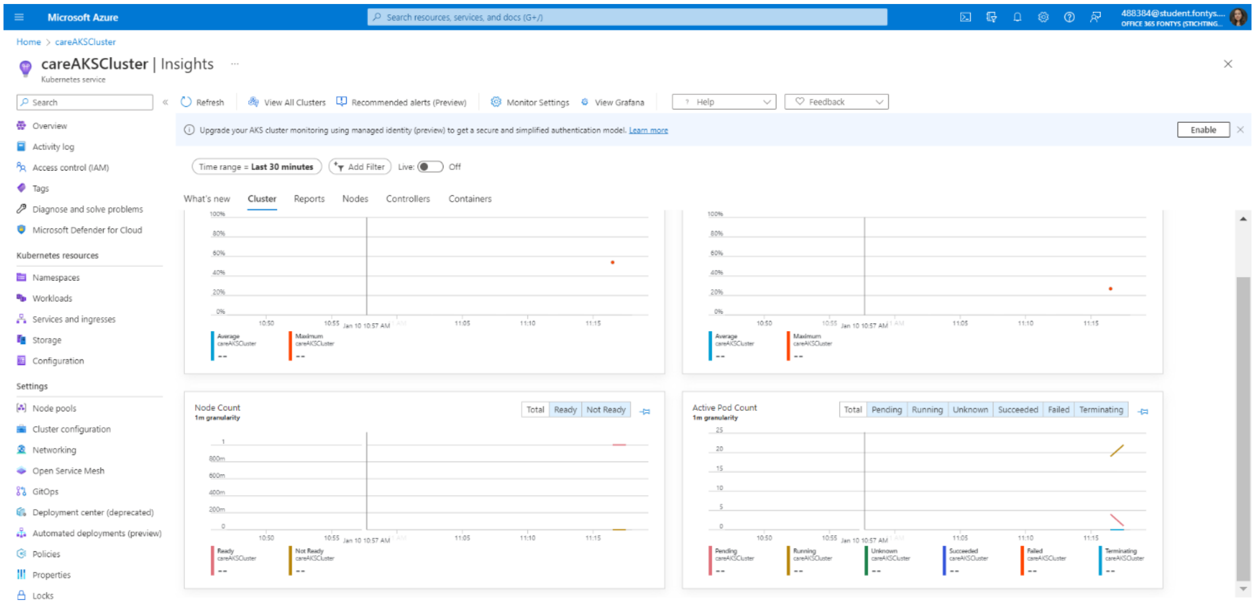
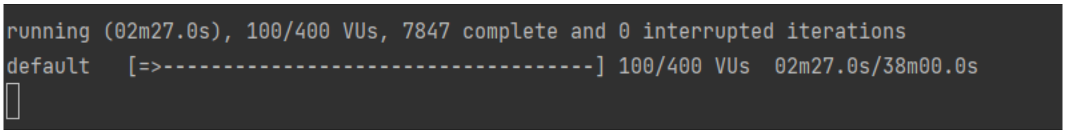


Image 21: “Azure portal insight into before running the stress test”

Performance Testing

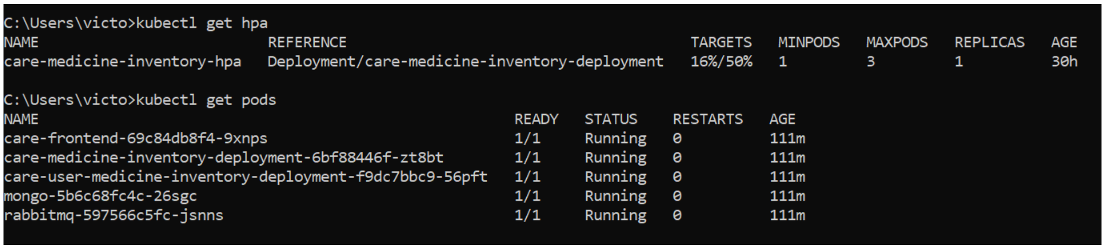
100 users target



```
running (02m27.0s), 100/400 VUs, 7847 complete and 0 interrupted iterations
default   [=>-----------------------------------] 100/400 VUs  02m27.0s/38m00.0s
```

Image 22: “current amount of virtual users created”

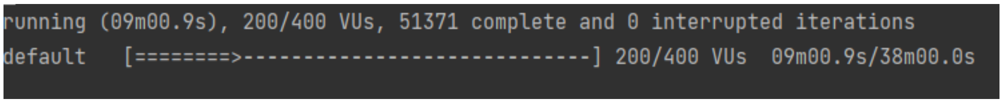
We can see that when the amount of virtual users is at 100 that the CPU usage is at 16% usage of 1 pod of the medicine inventory service. Where one pod can handle 100 virtual users.



```
C:\Users\victo>kubectl get hpa
NAME                       REFERENCE                                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
care-medicine-inventory-hpa   Deployment/care-medicine-inventory-deployment   16%/50%   1         3         1          30h

C:\Users\victo>kubectl get pods
NAME                                                   READY   STATUS    RESTARTS   AGE
care-frontend-69c84db8f4-9xnps                         1/1     Running   0          111m
care-medicine-inventory-deployment-6bf88446f-zt8bt     1/1     Running   0          111m
care-user-medicine-inventory-deployment-f9dc7bbc9-56pft   1/1     Running   0          111m
mongo-5b6c68fc4c-26sgc                                 1/1     Running   0          111m
rabbitmq-597566c5fc-jsnns                              1/1     Running   0          111m
```

Image 23: “CPU usage and pods running at 100 virtual users of the stress test”

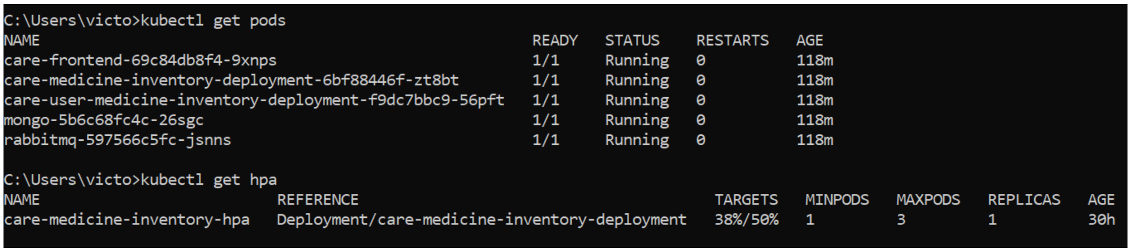
200 users target



```
running (09m00.9s), 200/400 VUs, 51371 complete and 0 interrupted iterations
default   [========>----------------------------] 200/400 VUs  09m00.9s/38m00.0s
```

Image 24: “current amount of virtual users created”

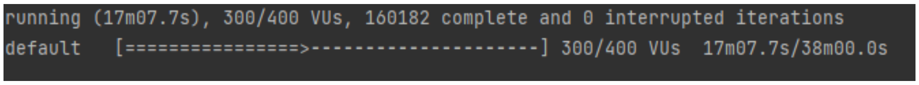
We can see that when the amount of virtual users is at 200 that the CPU usage is at 38% usage of 1 pod of the medicine inventory service. Which is double the amount from the CPU usage at 200 users. We can also conclude that one pod of the service can handle 200 virtual users for this request.



```
C:\Users\victo>kubectl get pods
NAME                                                   READY   STATUS    RESTARTS   AGE
care-frontend-69c84db8f4-9xnps                         1/1     Running   0          118m
care-medicine-inventory-deployment-6bf88446f-zt8bt     1/1     Running   0          118m
care-user-medicine-inventory-deployment-f9dc7bbc9-56pft   1/1     Running   0          118m
mongo-5b6c68fc4c-26sgc                                 1/1     Running   0          118m
rabbitmq-597566c5fc-jsnns                              1/1     Running   0          118m

C:\Users\victo>kubectl get hpa
NAME                       REFERENCE                                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
care-medicine-inventory-hpa   Deployment/care-medicine-inventory-deployment   38%/50%   1         3         1          30h
```

Image 25: “CPU usage and pods running at 200 virtual users of the stress test”

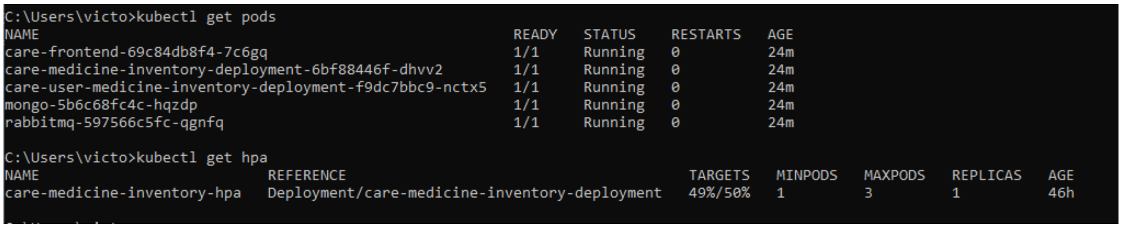
300 users target



```
running (17m07.7s), 300/400 VUs, 160182 complete and 0 interrupted iterations
default   [================>---------------------] 300/400 VUs  17m07.7s/38m00.0s
```

Image 26: “current amount of virtual users created”

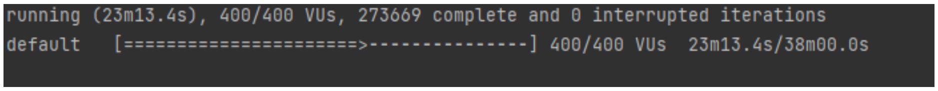
We can see that when the amount of virtual users is at 300 that the CPU usage is at 49% usage of 1 pod of the medicine inventory service. At this point we can say that the service can handle around 300 users for this certain request based on the CPU usage shown in image 27. We can predict that for more users that the pod should scale to handle more users.



```
C:\Users\victo>kubectl get pods
NAME                                                READY   STATUS    RESTARTS   AGE
care-frontend-69c84db8f4-7c6gq                      1/1     Running   0          24m
care-medicine-inventory-deployment-6bf88446f-dhvv2  1/1     Running   0          24m
care-user-medicine-inventory-deployment-f9dc7bbc9-nctx5   1/1     Running   0          24m
mongo-5b6c68fc4c-hqzdp                              1/1     Running   0          24m
rabbitmq-597566c5fc-qgnfq                           1/1     Running   0          24m

C:\Users\victo>kubectl get hpa
NAME                         REFERENCE                                       TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
care-medicine-inventory-hpa  Deployment/care-medicine-inventory-deployment   49%/50%   1         3         1          46h
```

Image 27: “CPU usage and pods running at 300 virtual users of the stress test”

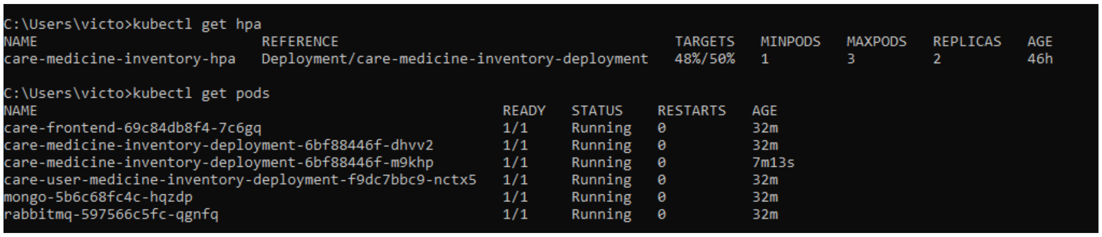
400 users target



```
running (23m13.4s), 400/400 VUs, 273669 complete and 0 interrupted iterations
default   [=====================>---------------] 400/400 VUs  23m13.4s/38m00.0s
```

Image 28: “current amount of virtual users created”

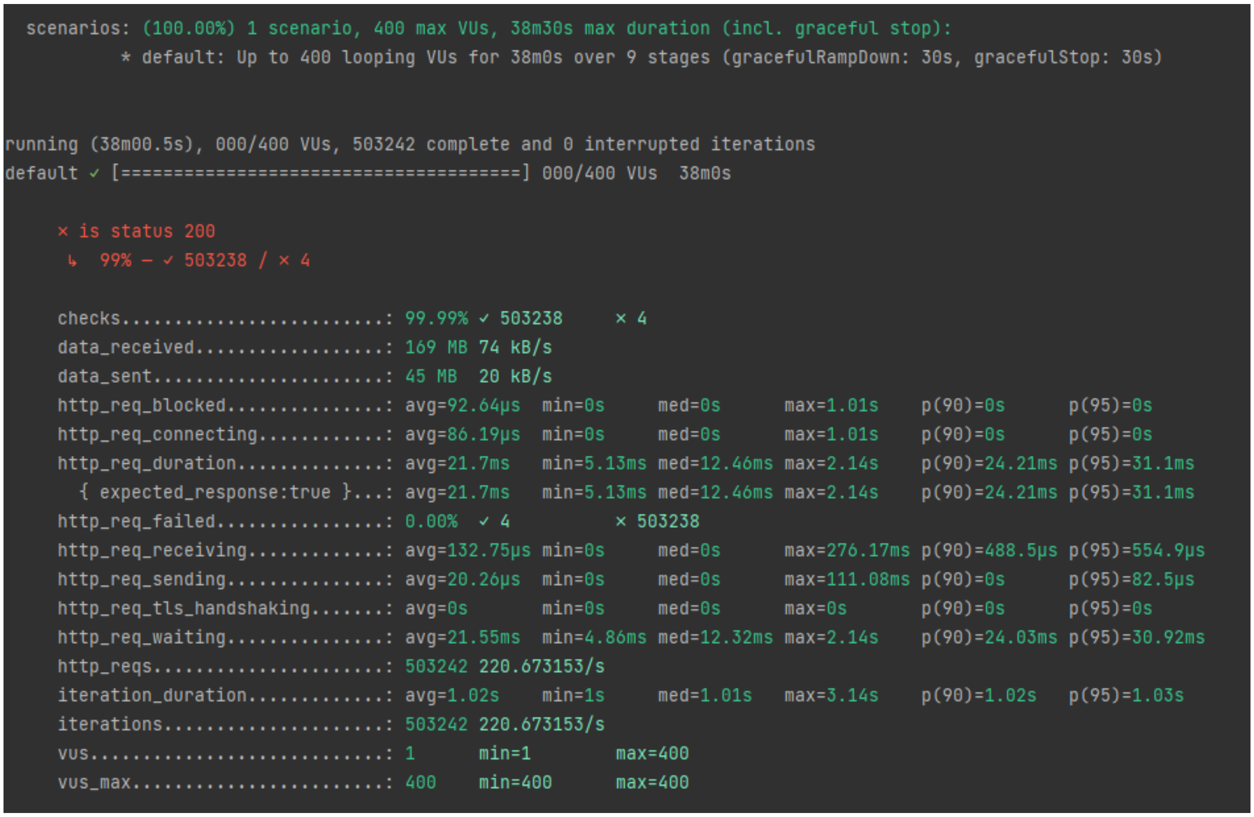
We can verify the prediction made by the target amount of users of 300 that the medicine inventory service pod has scaled to two pods to handle more virtual users. In image 29 we can see that there are two replicas of the pod and that they are both up and running.



```
C:\Users\victo>kubectl get hpa
NAME                         REFERENCE                                       TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
care-medicine-inventory-hpa  Deployment/care-medicine-inventory-deployment   48%/50%   1         3         2          46h

C:\Users\victo>kubectl get pods
NAME                                                READY   STATUS    RESTARTS   AGE
care-frontend-69c84db8f4-7c6gq                      1/1     Running   0          32m
care-medicine-inventory-deployment-6bf88446f-dhvv2  1/1     Running   0          32m
care-medicine-inventory-deployment-6bf88446f-m9khp  1/1     Running   0          7m13s
care-user-medicine-inventory-deployment-f9dc7bbc9-nctx5   1/1     Running   0          32m
mongo-5b6c68fc4c-hqzdp                              1/1     Running   0          32m
rabbitmq-597566c5fc-qgnfq                           1/1     Running   0          32m
```

Image 29: “CPU usage and pods running at 400 virtual users of the stress test”

Conclusion / End Results



```
  scenarios: (100.00%) 1 scenario, 400 max VUs, 38m30s max duration (incl. graceful stop):
           * default: Up to 400 looping VUs for 38m0s over 9 stages (gracefulRampDown: 30s, gracefulStop: 30s)


running (38m00.5s), 000/400 VUs, 503242 complete and 0 interrupted iterations
default ✓ [======================================] 000/400 VUs  38m0s

     × is status 200
      ↳  99% — ✓ 503238 / × 4

     checks.........................: 99.99% ✓ 503238     × 4
     data_received..................: 169 MB 74 kB/s
     data_sent......................: 45 MB  20 kB/s
     http_req_blocked...............: avg=92.64µs  min=0s     med=0s      max=1.01s    p(90)=0s      p(95)=0s
     http_req_connecting............: avg=86.19µs  min=0s     med=0s      max=1.01s    p(90)=0s      p(95)=0s
     http_req_duration..............: avg=21.7ms   min=5.13ms med=12.46ms max=2.14s    p(90)=24.21ms p(95)=31.1ms
       { expected_response:true }...: avg=21.7ms   min=5.13ms med=12.46ms max=2.14s    p(90)=24.21ms p(95)=31.1ms
     http_req_failed................: 0.00%  ✓ 4          × 503238
     http_req_receiving.............: avg=132.75µs min=0s     med=0s      max=276.17ms p(90)=488.5µs p(95)=554.9µs
     http_req_sending...............: avg=20.26µs  min=0s     med=0s      max=111.08ms p(90)=0s      p(95)=82.5µs
     http_req_tls_handshaking.......: avg=0s       min=0s     med=0s      max=0s       p(90)=0s      p(95)=0s
     http_req_waiting...............: avg=21.55ms  min=4.86ms med=12.32ms max=2.14s    p(90)=24.03ms p(95)=30.92ms
     http_reqs......................: 503242 220.673153/s
     iteration_duration.............: avg=1.02s    min=1s     med=1.01s   max=3.14s    p(90)=1.02s   p(95)=1.03s
     iterations.....................: 503242 220.673153/s
     vus............................: 1      min=1        max=400
     vus_max........................: 400    min=400      max=400
```

Image 30: “Results of the stress test”

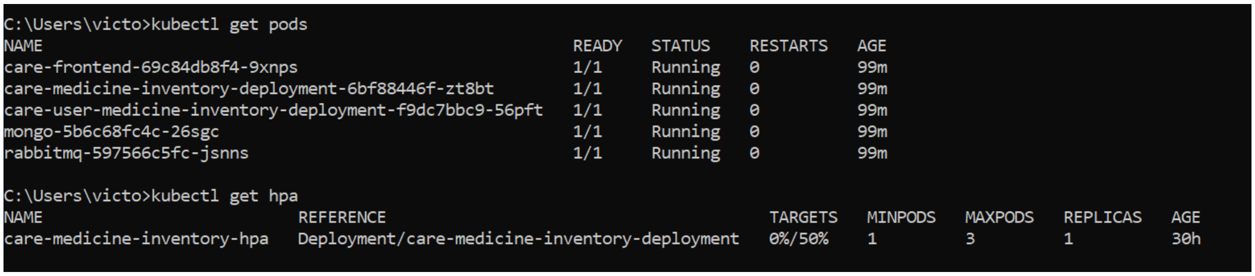
We can conclude that the medicine service inventory on Azure Kubernetes can scale based on the requests used for the maximum of 400 users.
Out of the 503248 requests made, did only 4 requests failed. Further was the average duration of a request 21.7 ms which is acceptable for the stress test.

Summary / Results of Scaling for a request holding ten registered medicine

We will be examining if the service scales when the amount of CPU increases over the set CPU usage for one pod and can handle a request with 10 register medicines. We will be assessing each target of virtual users to examine the CPU usage of the service pod.

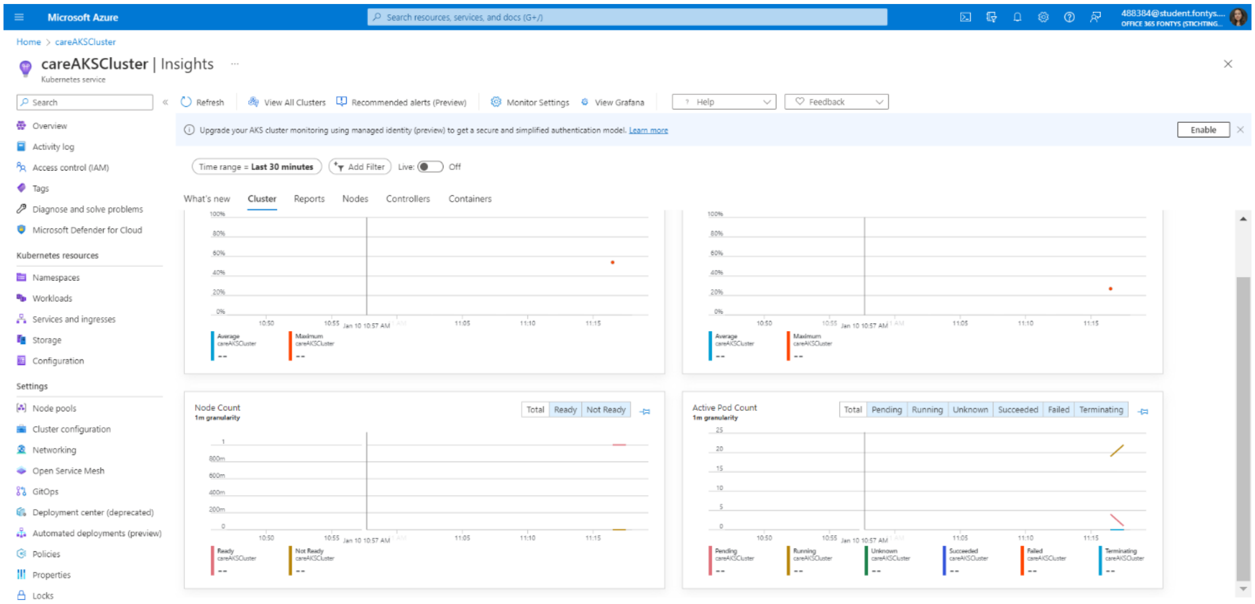
Start

The start of the test we can verify that the there is nothing running in the portal and that the CPU usage is at 0%.



```
C:\Users\victo>kubectl get pods
NAME                                                READY   STATUS    RESTARTS   AGE
care-frontend-69c84db8f4-9xnps                      1/1     Running   0          99m
care-medicine-inventory-deployment-6bf88446f-zt8bt  1/1     Running   0          99m
care-user-medicine-inventory-deployment-f9dc7bbc9-56pft  1/1     Running   0          99m
mongo-5b6c68fc4c-26sgc                              1/1     Running   0          99m
rabbitmq-597566c5fc-jsnns                           1/1     Running   0          99m

C:\Users\victo>kubectl get hpa
NAME                          REFERENCE                                       TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
care-medicine-inventory-hpa   Deployment/care-medicine-inventory-deployment   0%/50%    1         3         1          30h
```

Image 31: “CPU usage and pods running at the start of the stress test”

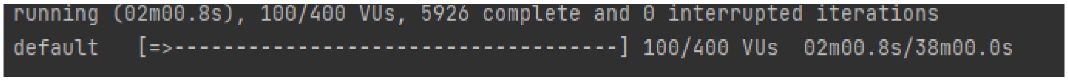


Upgrade your AKS cluster monitoring using managed identity (preview) to get a secure and simplified authentication model. Learn more

Time range = **Last 30 minutes**

Image 32: “Azure portal insight into before running the stress test”

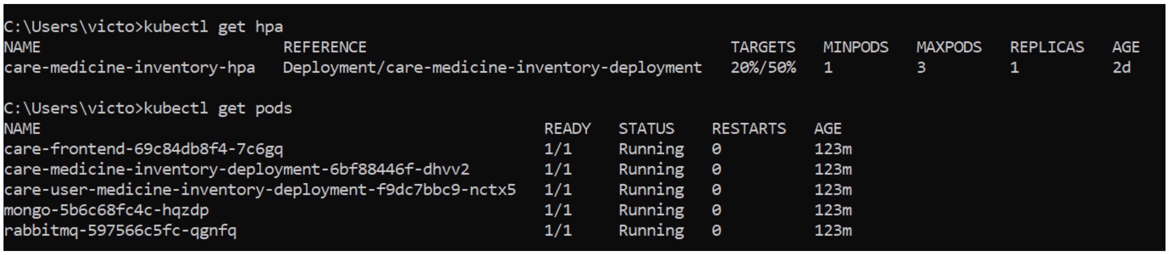
100 users target



```
running (02m00.8s), 100/400 VUs, 5926 complete and 0 interrupted iterations
default   [=>----------------------------------] 100/400 VUs  02m00.8s/38m00.0s
```

Image 33: “current amount of virtual users created”

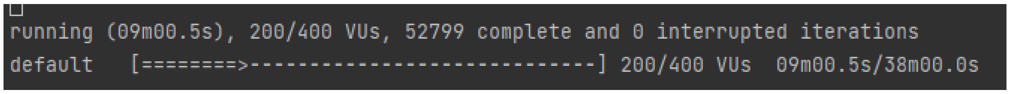
We can see that when the amount of virtual users is at 100 that the CPU usage is at 20% usage of 1 pod of the medicine inventory service. Where one pod can handle 100 virtual users with a request of 10 registered medicines.



```
C:\Users\victo>kubectl get hpa
NAME                        REFERENCE                                       TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
care-medicine-inventory-hpa   Deployment/care-medicine-inventory-deployment   20%/50%   1         3         1          2d

C:\Users\victo>kubectl get pods
NAME                                                 READY   STATUS    RESTARTS   AGE
care-frontend-69c84db8f4-7c6gq                       1/1     Running   0          123m
care-medicine-inventory-deployment-6bf88446f-dhvv2   1/1     Running   0          123m
care-user-medicine-inventory-deployment-f9dc7bbc9-nctx5   1/1     Running   0          123m
mongo-5b6c68fc4c-hqzdp                               1/1     Running   0          123m
rabbitmq-597566c5fc-qgnfq                            1/1     Running   0          123m
```

Image 34: “CPU usage and pods running at 100 virtual users of the stress test”

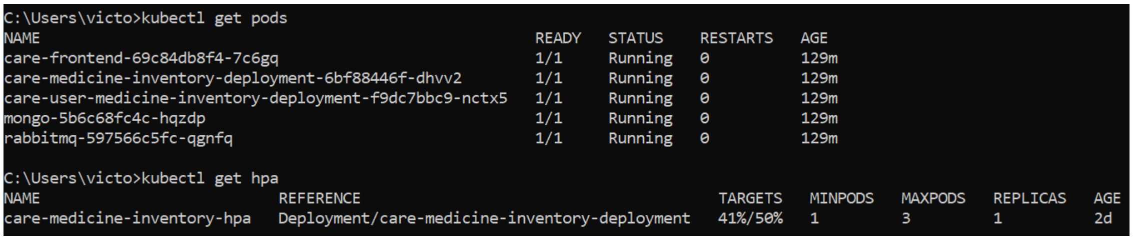
200 users target



```
running (09m00.5s), 200/400 VUs, 52799 complete and 0 interrupted iterations
default   [========>----------------------------] 200/400 VUs  09m00.5s/38m00.0s
```

Image 35: “current amount of virtual users created”

We can see that when the amount of virtual users is at 200 that the CPU usage is at 41% usage of 1 pod of the medicine inventory service. Which is more than the CPU usage for a request with 1 registered medicine. Nonetheless, we can also conclude that one pod of the service can handle 200 virtual users for this request.

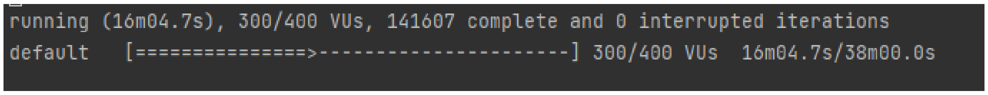


```
C:\Users\victo>kubectl get pods
NAME                                                      READY   STATUS    RESTARTS   AGE
care-frontend-69c84db8f4-7c6gq                            1/1     Running   0          129m
care-medicine-inventory-deployment-6bf88446f-dhvv2        1/1     Running   0          129m
care-user-medicine-inventory-deployment-f9dc7bbc9-nctx5   1/1     Running   0          129m
mongo-5b6c68fc4c-hqzdp                                    1/1     Running   0          129m
rabbitmq-597566c5fc-qgnfq                                 1/1     Running   0          129m

C:\Users\victo>kubectl get hpa
NAME                          REFERENCE                                       TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
care-medicine-inventory-hpa   Deployment/care-medicine-inventory-deployment   41%/50%   1         3         1          2d
```

Image 36: “CPU usage and pods running at 200 virtual users of the stress test”

Performance Testing

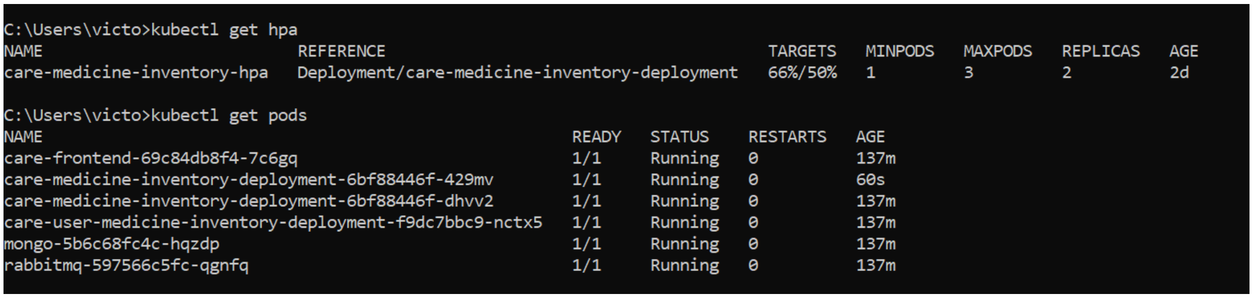
300 users target



```
running (16m04.7s), 300/400 VUs, 141607 complete and 0 interrupted iterations
default   [===============>----------------------] 300/400 VUs  16m04.7s/38m00.0s
```

Image 37: “current amount of virtual users created”

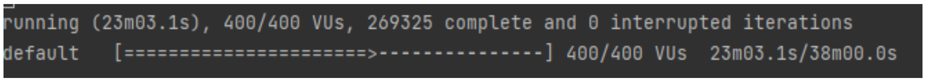
We can see that when the amount of virtual users is at 300 that the CPU usage is at 66% usage of 1 pod of the medicine inventory service which is over the registered amount of CPU usage per pod. Here can requests take longer than expected or fail. We will see the outcome at the end of the test.



```
C:\Users\victo>kubectl get hpa
NAME                         REFERENCE                                         TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
care-medicine-inventory-hpa   Deployment/care-medicine-inventory-deployment    66%/50%   1         3         2          2d

C:\Users\victo>kubectl get pods
NAME                                                      READY   STATUS    RESTARTS   AGE
care-frontend-69c84db8f4-7c6gq                            1/1     Running   0          137m
care-medicine-inventory-deployment-6bf88446f-429mv        1/1     Running   0          60s
care-medicine-inventory-deployment-6bf88446f-dhvv2        1/1     Running   0          137m
care-user-medicine-inventory-deployment-f9dc7bbc9-nctx5   1/1     Running   0          137m
mongo-5b6c68fc4c-hqzdp                                    1/1     Running   0          137m
rabbitmq-597566c5fc-qgnfq                                 1/1     Running   0          137m
```

Image 38: “CPU usage and pods running at 300 virtual users of the stress test”

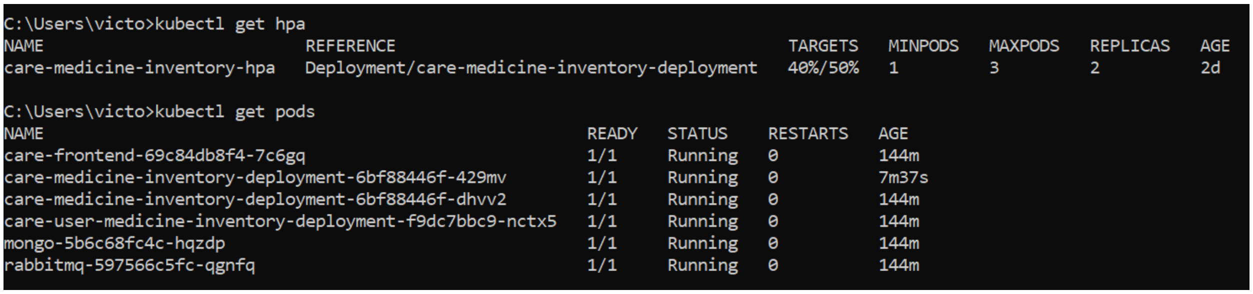
400 users target



```
running (23m03.1s), 400/400 VUs, 269325 complete and 0 interrupted iterations
default   [=====================>---------------] 400/400 VUs  23m03.1s/38m00.0s
```

Image 39: “current amount of virtual users created”

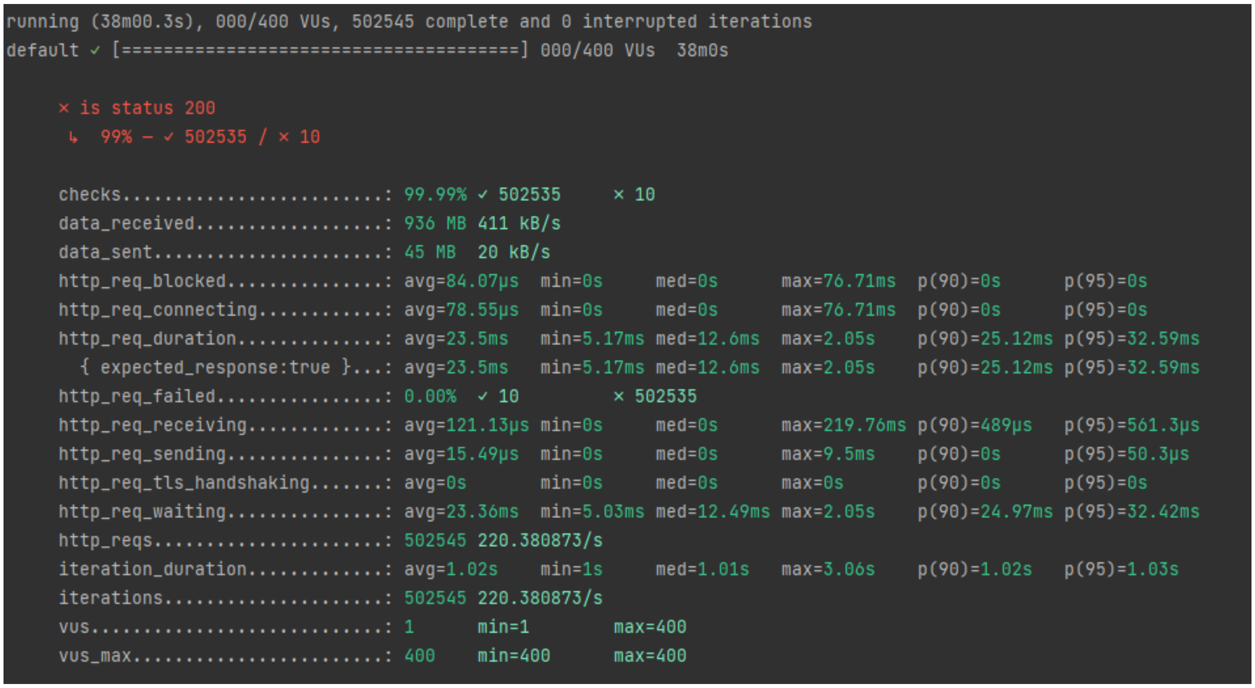
We can see that the amount of CPU usage has decreased with an added pod to the medicine inventory service.



```
C:\Users\victo>kubectl get hpa
NAME                         REFERENCE                                         TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
care-medicine-inventory-hpa   Deployment/care-medicine-inventory-deployment    40%/50%   1         3         2          2d

C:\Users\victo>kubectl get pods
NAME                                                      READY   STATUS    RESTARTS   AGE
care-frontend-69c84db8f4-7c6gq                            1/1     Running   0          144m
care-medicine-inventory-deployment-6bf88446f-429mv        1/1     Running   0          7m37s
care-medicine-inventory-deployment-6bf88446f-dhvv2        1/1     Running   0          144m
care-user-medicine-inventory-deployment-f9dc7bbc9-nctx5   1/1     Running   0          144m
mongo-5b6c68fc4c-hqzdp                                    1/1     Running   0          144m
rabbitmq-597566c5fc-qgnfq                                 1/1     Running   0          144m
```

Image 40: “CPU usage and pods running at 400 virtual users of the stress test”

Conclusion / End Results



```
running (38m00.3s), 000/400 VUs, 502545 complete and 0 interrupted iterations
default ✓ [======================================] 000/400 VUs  38m0s

     × is status 200
      ↳  99% — ✓ 502535 / × 10

     checks.........................: 99.99% ✓ 502535     × 10
     data_received..................: 936 MB 411 kB/s
     data_sent......................: 45 MB  20 kB/s
     http_req_blocked...............: avg=84.07µs  min=0s     med=0s      max=76.71ms  p(90)=0s      p(95)=0s
     http_req_connecting............: avg=78.55µs  min=0s     med=0s      max=76.71ms  p(90)=0s      p(95)=0s
     http_req_duration..............: avg=23.5ms   min=5.17ms med=12.6ms  max=2.05s    p(90)=25.12ms p(95)=32.59ms
       { expected_response:true }...: avg=23.5ms   min=5.17ms med=12.6ms  max=2.05s    p(90)=25.12ms p(95)=32.59ms
     http_req_failed................: 0.00%  ✓ 10         × 502535
     http_req_receiving.............: avg=121.13µs min=0s     med=0s      max=219.76ms p(90)=489µs   p(95)=561.3µs
     http_req_sending...............: avg=15.49µs  min=0s     med=0s      max=9.5ms    p(90)=0s      p(95)=50.3µs
     http_req_tls_handshaking.......: avg=0s       min=0s     med=0s      max=0s       p(90)=0s      p(95)=0s
     http_req_waiting...............: avg=23.36ms  min=5.03ms med=12.49ms max=2.05s    p(90)=24.97ms p(95)=32.42ms
     http_reqs......................: 502545 220.380873/s
     iteration_duration.............: avg=1.02s    min=1s     med=1.01s   max=3.06s    p(90)=1.02s   p(95)=1.03s
     iterations.....................: 502545 220.380873/s
     vus............................: 1      min=1       max=400
     vus_max........................: 400    min=400     max=400
```

Image 41: "Results of the stress test"

We can conclude that the medicine service inventory on Azure Kubernetes can scale based on the requests used for the maximum of 400 users.
Out of the 502535 requests made, did only 10 requests failed. Further was the average duration of a request 23.5 ms which is acceptable for the stress test.

Comparison between amount of registered medicines of a request

Comparing the stress test on a request with 1 registered medicine to a request with 10 registered medicines we can say that there are more fails with the request of 10 registered medicines with also a longer average duration.
Therefor, the make fact of the assumption made that more registered medicines will take longer and more CPU usage where how bigger the amount of storage, how more storage has to be made on Azure Kubernetes to handle larger amounts of requests.

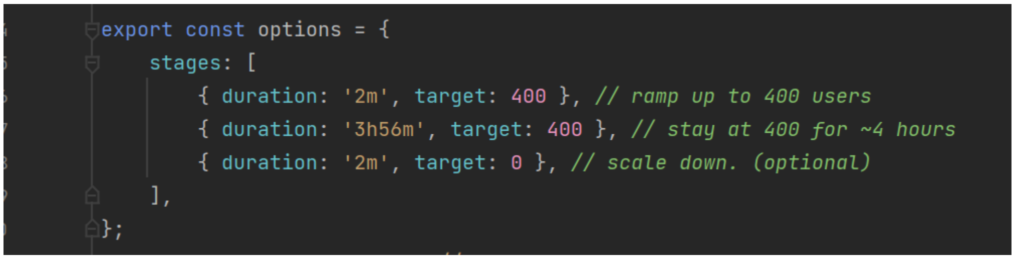
Soak Testing

Context

Here we will be using soak testing to assess the reliability of the system over a longer period of time. The purpose of performing the soak test is to find out performance and reliability issues stemming from the system from being under pressure for an extended period.
These issues typically include bugs, memory leaks, insufficient storage quotas, incorrect configuration or infrastructure failures.
With the help of a soak test, days' worth of traffic can be simulated into only a few hours.

Test Case

It is recommended to configure the soak test to about 80% capacity of the service. If the service can handle a maximum of 500 users, then would the configure amount of users be set to 400 users which is what we will be tested on the medicine inventory service.



```
export const options = {
    stages: [
        { duration: '2m', target: 400 }, // ramp up to 400 users
        { duration: '3h56m', target: 400 }, // stay at 400 for ~4 hours
        { duration: '2m', target: 0 }, // scale down. (optional)
    ],
};
```

Image 42: "Set target virtual users for the Soak Test"

The test will be started at 0 users and will increase to 400 users within a span of 2 minutes. The amount of users will stay at 400 for 4 hours to examine the performance of the service. Lastly, the service will scale down the amount of users to 0 in a span of 0 users.

Virtual Users (VU) chart

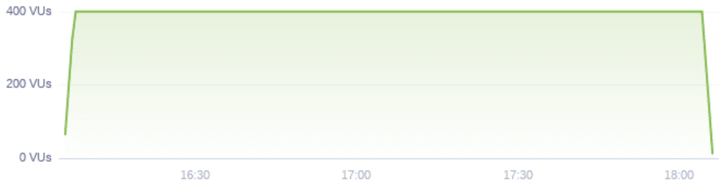


Image 43: "Virtual User Chart of the Soak Test"

The VU chart of the soak test that will be used for the service should look similar to image 43.

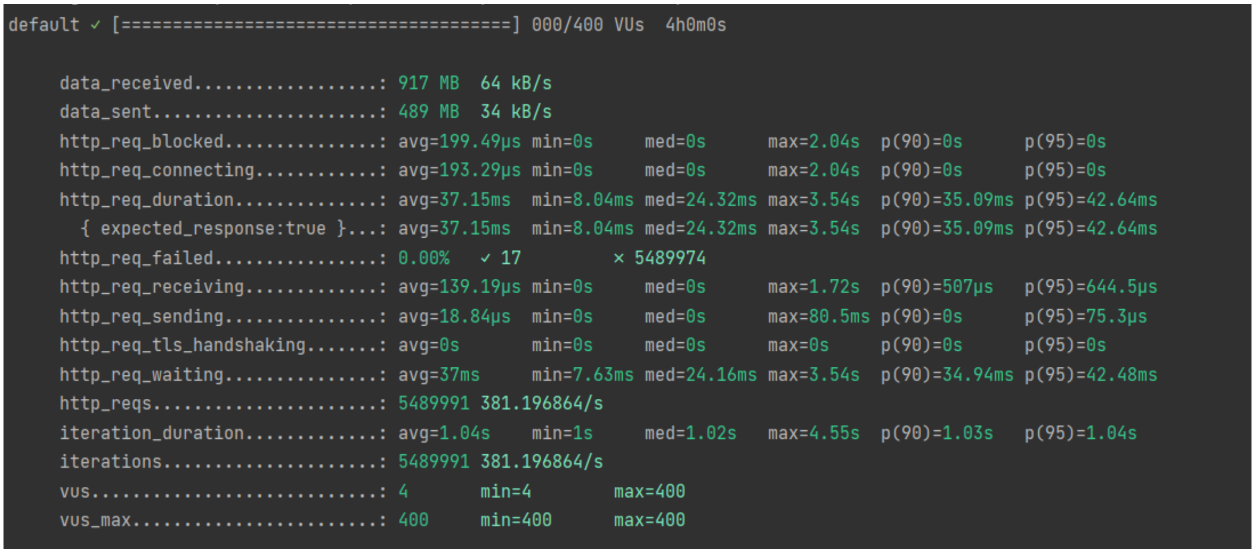
We will be creating and testing a soak test on the medicine inventory of the care web application for the purpose understanding and trying out the test. The reason there is no test for the user medicine inventory test is due to how long the test takes which will cost on Azure and to the priority of the other tests being completed and tested on the services.

Medicine Service

Scenario

The test is based on how the service can handle requests over a long period of time with a good amount of users. We will be using the get request of this endpoint http://20.103.147.242/medicines to retrieve all registered medicines.

Summary



```
default ✓ [======================================] 000/400 VUs  4h0m0s

     data_received..................: 917 MB  64 kB/s
     data_sent......................: 489 MB  34 kB/s
     http_req_blocked...............: avg=199.49µs min=0s     med=0s      max=2.04s  p(90)=0s      p(95)=0s
     http_req_connecting............: avg=193.29µs min=0s     med=0s      max=2.04s  p(90)=0s      p(95)=0s
     http_req_duration..............: avg=37.15ms  min=8.04ms med=24.32ms max=3.54s  p(90)=35.09ms p(95)=42.64ms
       { expected_response:true }...: avg=37.15ms  min=8.04ms med=24.32ms max=3.54s  p(90)=35.09ms p(95)=42.64ms
     http_req_failed................: 0.00%   ✓ 17        × 5489974
     http_req_receiving.............: avg=139.19µs min=0s     med=0s      max=1.72s  p(90)=507µs   p(95)=644.5µs
     http_req_sending...............: avg=18.84µs  min=0s     med=0s      max=80.5ms p(90)=0s      p(95)=75.3µs
     http_req_tls_handshaking.......: avg=0s       min=0s     med=0s      max=0s     p(90)=0s      p(95)=0s
     http_req_waiting...............: avg=37ms     min=7.63ms med=24.16ms max=3.54s  p(90)=34.94ms p(95)=42.48ms
     http_reqs......................: 5489991 381.196864/s
     iteration_duration.............: avg=1.04s    min=1s     med=1.02s   max=4.55s  p(90)=1.03s   p(95)=1.04s
     iterations.....................: 5489991 381.196864/s
     vus............................: 4       min=4       max=400
     vus_max........................: 400     min=400     max=400
```

Image 44: "Results of the Soak Test tested on the medicine inventory service"

To summarize, from running the soak test against the medicine inventory service we can see that from the 5489974 requests sent, 17 have failed (under http_req_failed).
During and after the process of running the test was the service still up, running and working.
To conclude from running the soak test can it be said that the service can handle a long period of time of users sending requests and that there were no interruptions made to the service.

Sources

k6 Documentation. (n.d.-a). https://k6.io/docs/

What is Smoke Testing? How to create a Smoke Test in k6. (n.d.). https://k6.io/docs/test-types/smoke-testing/

What is Load Testing? How to create a Load Test in k6. (n.d.). https://k6.io/docs/test-types/load-testing/

What is Stress Testing? How to create a Stress Test in k6. (n.d.). https://k6.io/docs/test-types/stress-testing/

What is Soak Testing? How to create a Soak Test in k6. (n.d.). https://k6.io/docs/test-types/soak-testing/

Paulo, M. (2022, February 5). *Load and stress testing in .NET 6 - Matias Paulo*. Medium. https://medium.com/@matias.paulo84/load-and-stress-testing-in-net-6-725ba346cf78

Metrics. (n.d.). https://k6.io/docs/using-k6/metrics/

Running k6. (n.d.). https://k6.io/docs/get-started/running-k6/

Thresholds. (n.d.). https://k6.io/docs/using-k6/thresholds/

Teixeira, B. (2022, January 4). *Using K6 test + Docker + Reports - Geek Culture*. Medium. https://medium.com/geekculture/using-k6-tests-docker-reports-53366512b5c5

Running large tests. (n.d.). https://k6.io/docs/testing-guides/running-large-tests/