Containers vs Virtual Machines

Containers and Virtual Machines (VMs) offer different ways to deploy applications to a web server. While VMs share only the hardware of the host system, Containers share both the hardware and the OS. To compare both approaches, I created a simple web application for Contact Management System using HTML, CSS, and JS for front-end and Python Flask and SQLite for back-end.

I successfully deployed the application on VMs using Vagrant and VirtualBox, and I did the same on Containers using Docker. One of the superficial differences observed between the two is that while the codebase remains the same, we are changing the supporting files for each use case. For deploying to VM, I required a **VagrantFile**, whereas for deploying to Containers, I used **Dockerfile**. Both these files essentially perform similar jobs which is to define & set up the environment, and deploy the application using the given commands.

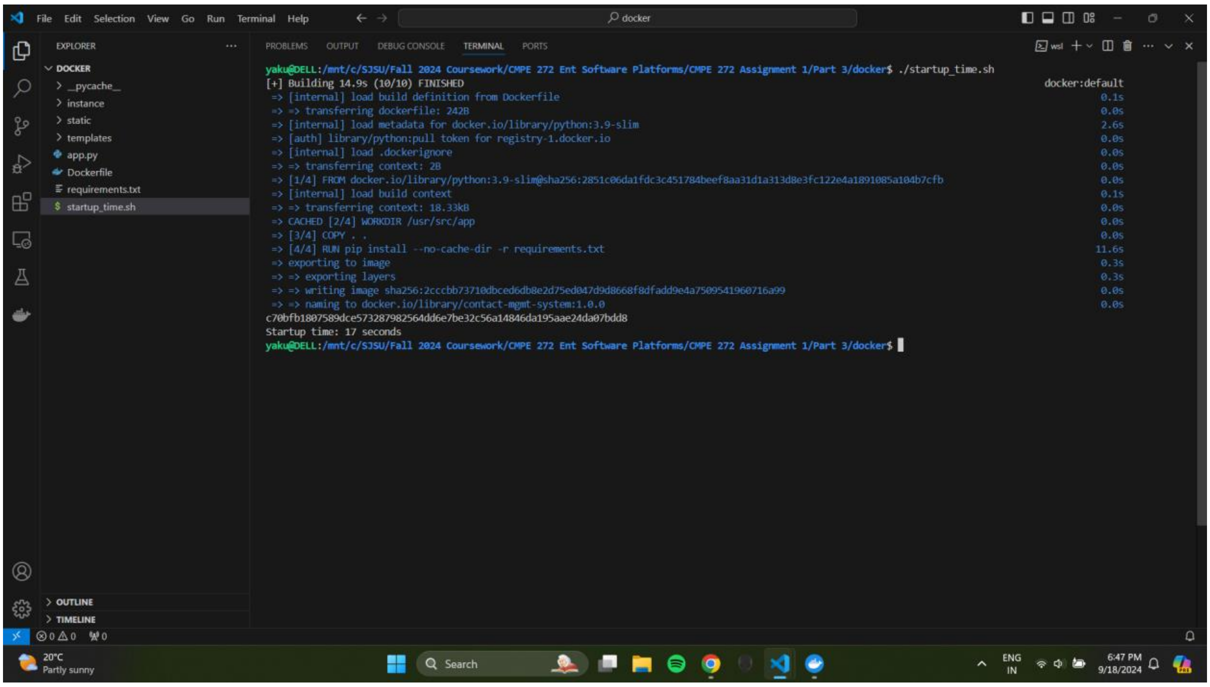
However, to statistically define which approach is better, we want to observe how their performancse over various metrics such as:

1. Startup time - The time it takes for an application to become fully operational from the moment it is started.
2. Memory usage - The amount of system memory (RAM) an application consumes while running.
3. CPU utilization - The percentage of CPU resources used by an application relative to the total available CPU resources.
4. Request throughput - The number of requests an application can process in a given time frame, usually measured in requests per second (RPS).
5. Response time - The amount of time taken by an application to respond to a request, typically measured from the moment a request is received until a response is sent back.
6. Garbage collection - The process of automatically freeing up memory by reclaiming space occupied by objects that are no longer in use by the application.

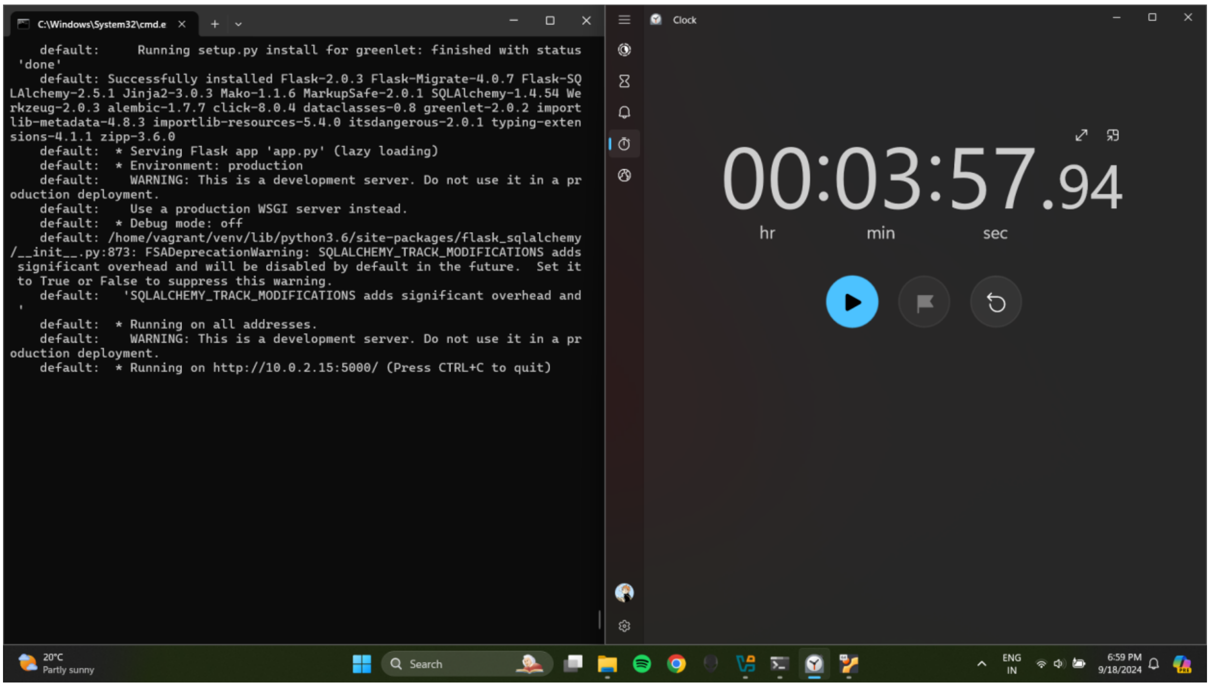
Startup Time

For containers, I wrote a shell script which encapsulated image creation and containerization commands and returned the startup time as soon as the application was deployed to the container. On the other hand, I manually calculated using a timer on my system for VMs. I observed that while the containers were up within 15-20 seconds, the VMs would take upto 4-5 minutes for the same. With respect to startup time, Containers have a clear edge over VMs.

The first image below represents the startup time for **Containers (17 secs)**, whereas the second image represents the same for **VMs (3 minute, 57 seconds)**.



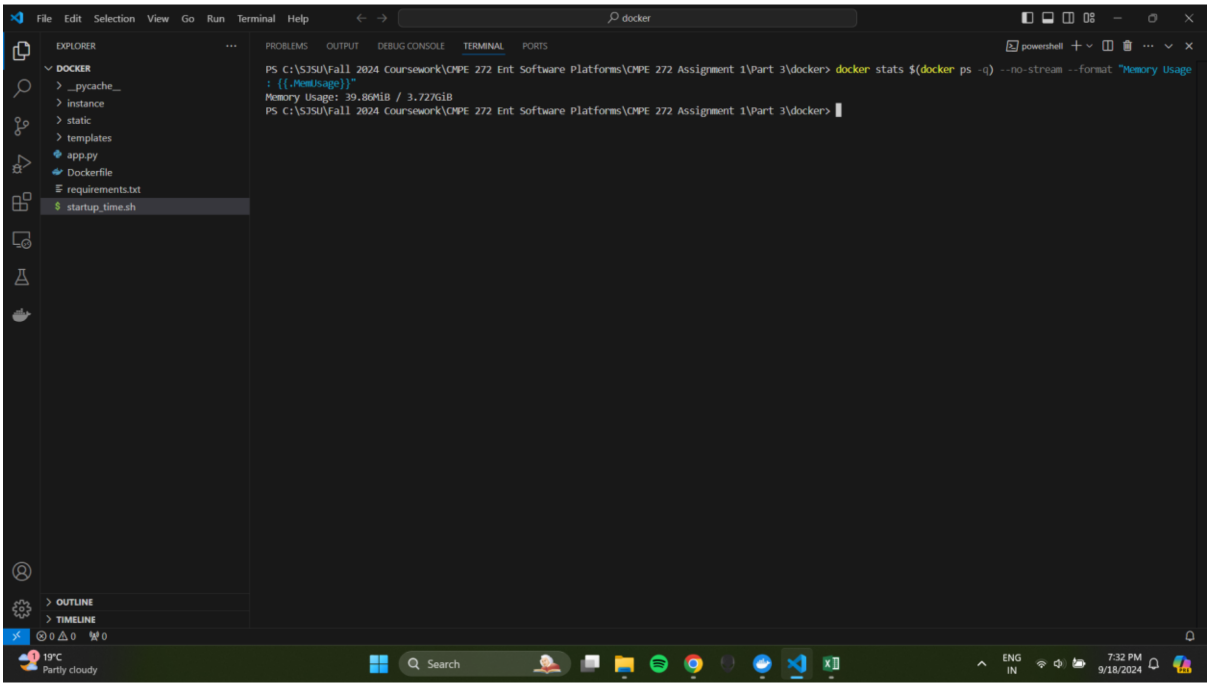
```
yaku@DELL:/mnt/c/SJSU/Fall 2024 Coursework/CMPE 272 Ent Software Platforms/CMPE 272 Assignment 1/Part 3/docker$ ./startup_time.sh
[+] Building 14.9s (10/10) FINISHED                                                                            docker:default
 => [internal] load build definition from Dockerfile                                                                     0.1s
 => => transferring dockerfile: 242B                                                                                     0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                                       2.6s
 => [auth] library/python:pull token for registry-1.docker.io                                                            0.0s
 => [internal] load .dockerignore                                                                                        0.0s
 => => transferring context: 2B                                                                                          0.0s
 => [1/4] FROM docker.io/library/python:3.9-slim@sha256:2851c06da1fdc3c451784beef8aa31d1a313d8e3fc122e4a1891085a104b7cfb  0.0s
 => [internal] load build context                                                                                        0.1s
 => => transferring context: 18.33kB                                                                                     0.0s
 => CACHED [2/4] WORKDIR /usr/src/app                                                                                    0.0s
 => [3/4] COPY . .                                                                                                       0.0s
 => [4/4] RUN pip install --no-cache-dir -r requirements.txt                                                            11.6s
 => exporting to image                                                                                                   0.3s
 => => exporting layers                                                                                                  0.3s
 => => writing image sha256:2cccbb73710dbced6db8e2d75ed047d9d8668f8dfadd9e4a7509541960716a99                             0.0s
 => => naming to docker.io/library/contact-mgmt-system:1.0.0                                                             0.0s
c70bfb1807589dce573287982564dd6e7be32c56a14846da195aae24da07bdd8
Startup time: 17 seconds
yaku@DELL:/mnt/c/SJSU/Fall 2024 Coursework/CMPE 272 Ent Software Platforms/CMPE 272 Assignment 1/Part 3/docker$
```



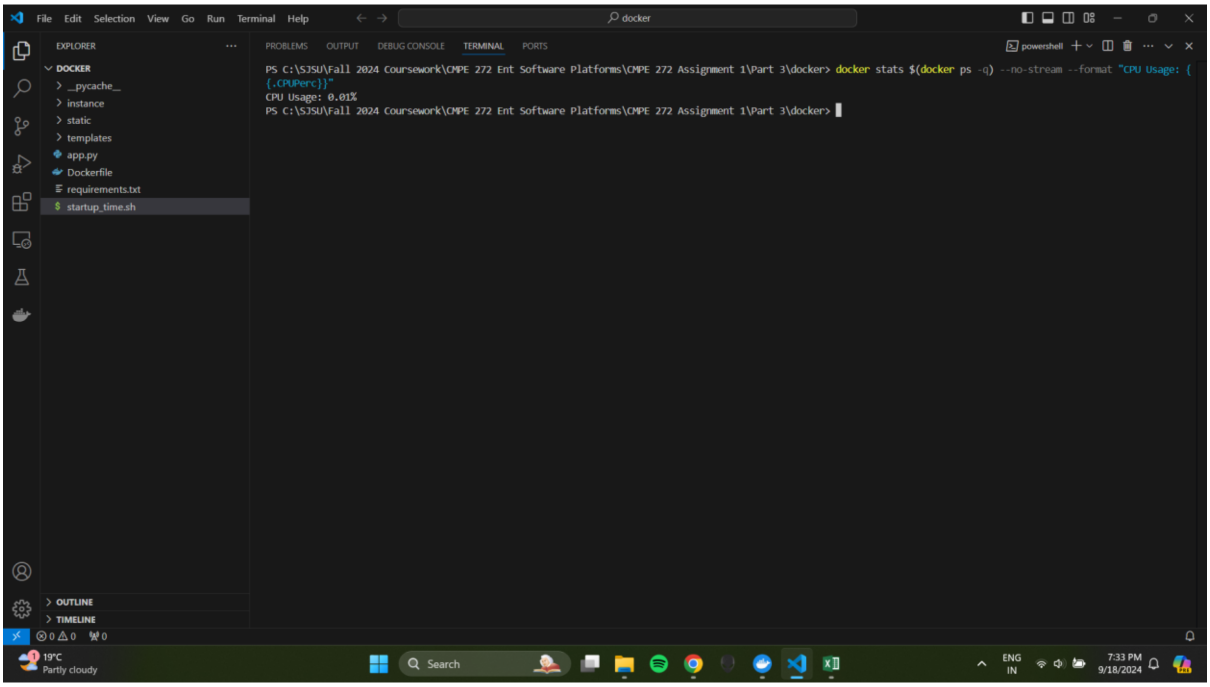
```
    default:     Running setup.py install for greenlet: finished with status
 'done'
    default: Successfully installed Flask-2.0.3 Flask-Migrate-4.0.7 Flask-SQ
LAlchemy-2.5.1 Jinja2-3.0.3 Mako-1.1.6 MarkupSafe-2.0.1 SQLAlchemy-1.4.54 We
rkzeug-2.0.3 alembic-1.7.7 click-8.0.4 dataclasses-0.8 greenlet-2.0.2 import
lib-metadata-4.8.3 importlib-resources-5.4.0 itsdangerous-2.0.1 typing-exten
sions-4.1.1 zipp-3.6.0
    default:  * Serving Flask app 'app.py' (lazy loading)
    default:  * Environment: production
    default:    WARNING: This is a development server. Do not use it in a pr
oduction deployment.
    default:    Use a production WSGI server instead.
    default:  * Debug mode: off
    default: /home/vagrant/venv/lib/python3.6/site-packages/flask_sqlalchemy
/__init__.py:873: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds
 significant overhead and will be disabled by default in the future.  Set it
 to True or False to suppress this warning.
    default:   'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and
 '
    default:  * Running on all addresses.
    default:    WARNING: This is a development server. Do not use it in a pr
oduction deployment.
    default:  * Running on http://10.0.2.15:5000/ (Press CTRL+C to quit)
```

Memory Usage & CPU Utilization

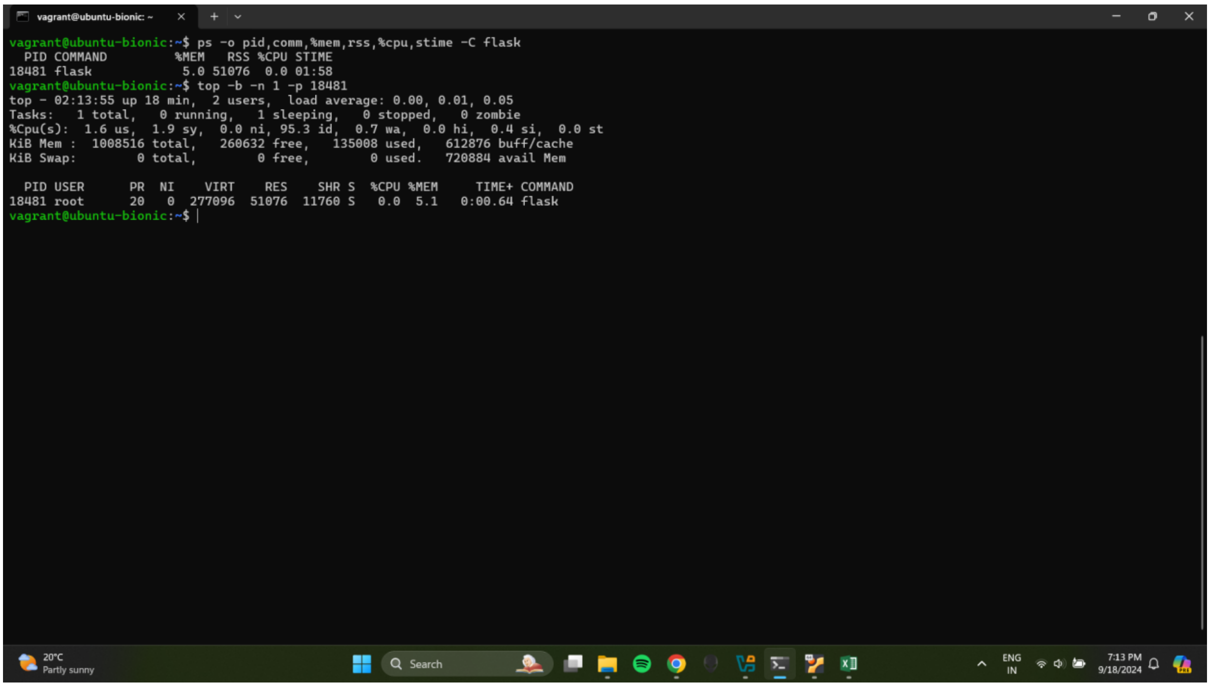
With respect to memory usage & cpu utilization, I used commands for getting the stats for both. For dockers, I was able to do so using the **docker stats** command. On the other hand, for VMs, I used the **ps** command. With respect to memory usage, I observed that the containers used significantly lesser memory for the same functionality. Where VMs used nearly 5% of the memory, Containers used closer to 1%. On the other hand, regarding CPU Utilization, it was a lot closer with Containers using 0.01% and VMs using 0.00%. VMs did use much lesser, but neither is using a lot of CPU.



```
PS C:\SJSU\Fall 2024 Coursework\CMPE 272 Ent Software Platforms\CMPE 272 Assignment 1\Part 3\docker> docker stats $(docker ps -q) --no-stream --format "Memory Usage
: {{.MemUsage}}"
Memory Usage: 39.86MiB / 3.727GiB
PS C:\SJSU\Fall 2024 Coursework\CMPE 272 Ent Software Platforms\CMPE 272 Assignment 1\Part 3\docker>
```



```
PS C:\SJSU\Fall 2024 Coursework\CMPE 272 Ent Software Platforms\CMPE 272 Assignment 1\Part 3\docker> docker stats $(docker ps -q) --no-stream --format "CPU Usage: {
{.CPUPerc}}"
CPU Usage: 0.01%
PS C:\SJSU\Fall 2024 Coursework\CMPE 272 Ent Software Platforms\CMPE 272 Assignment 1\Part 3\docker>
```



```
vagrant@ubuntu-bionic:~$ ps -o pid,comm,%mem,rss,%cpu,stime -C flask
  PID COMMAND         %MEM   RSS %CPU STIME
18481 flask            5.0 51076  0.0 01:58
vagrant@ubuntu-bionic:~$ top -b -n 1 -p 18481
top - 02:13:55 up 18 min,  2 users,  load average: 0.00, 0.01, 0.05
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.6 us,  1.9 sy,  0.0 ni, 95.3 id,  0.7 wa,  0.0 hi,  0.4 si,  0.0 st
KiB Mem :  1008516 total,   260632 free,   135008 used,   612876 buff/cache
KiB Swap:        0 total,        0 free,        0 used.   720884 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
18481 root      20   0  277096  51076  11760 S   0.0  5.1   0:00.64 flask
vagrant@ubuntu-bionic:~$ |
```

The first two images show the Memory and CPU usage in the Containers approach, whereas the third image shows the same for VMs. The table below summarizes the observed data as well.

Metric	VMs	Containers
Memory Usage %	5.10%	1.04%
CPU Utilization %	0.00%	0.01%

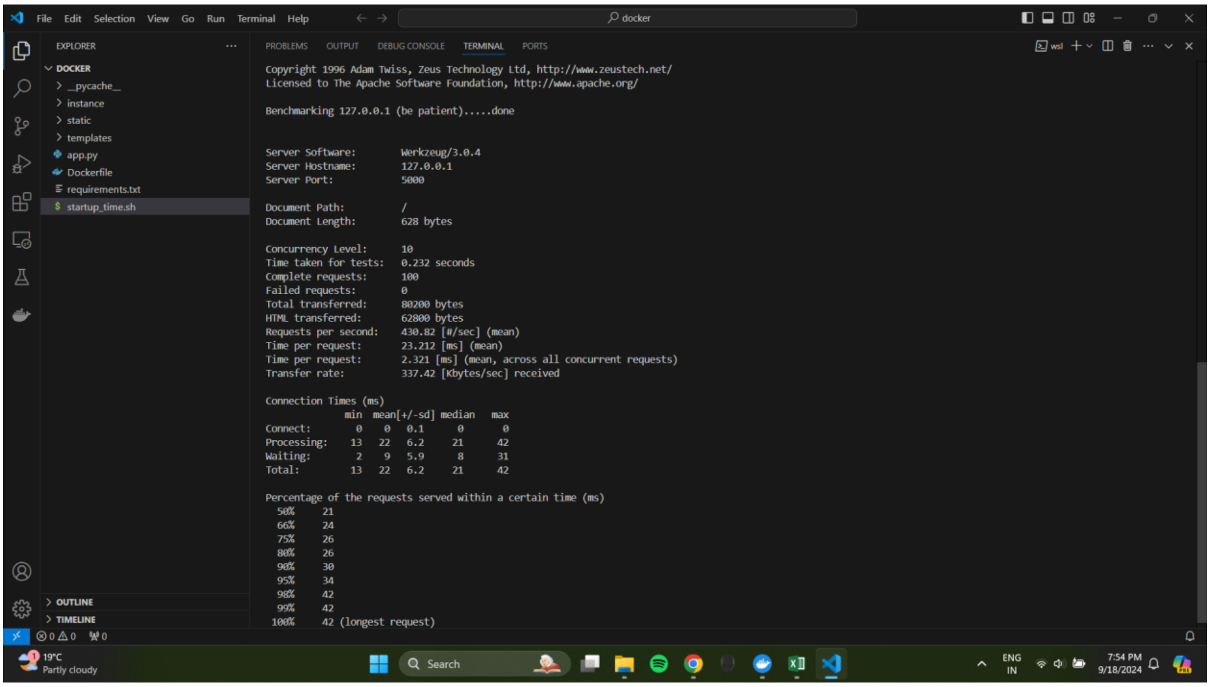
Request Throughput & Response Time

Request throughput and Response time go hand-in-hand, and hence, I used the Apache Benchmarking tool to measure both these metrics. I tested both VMs and Containers in two different scenarios:

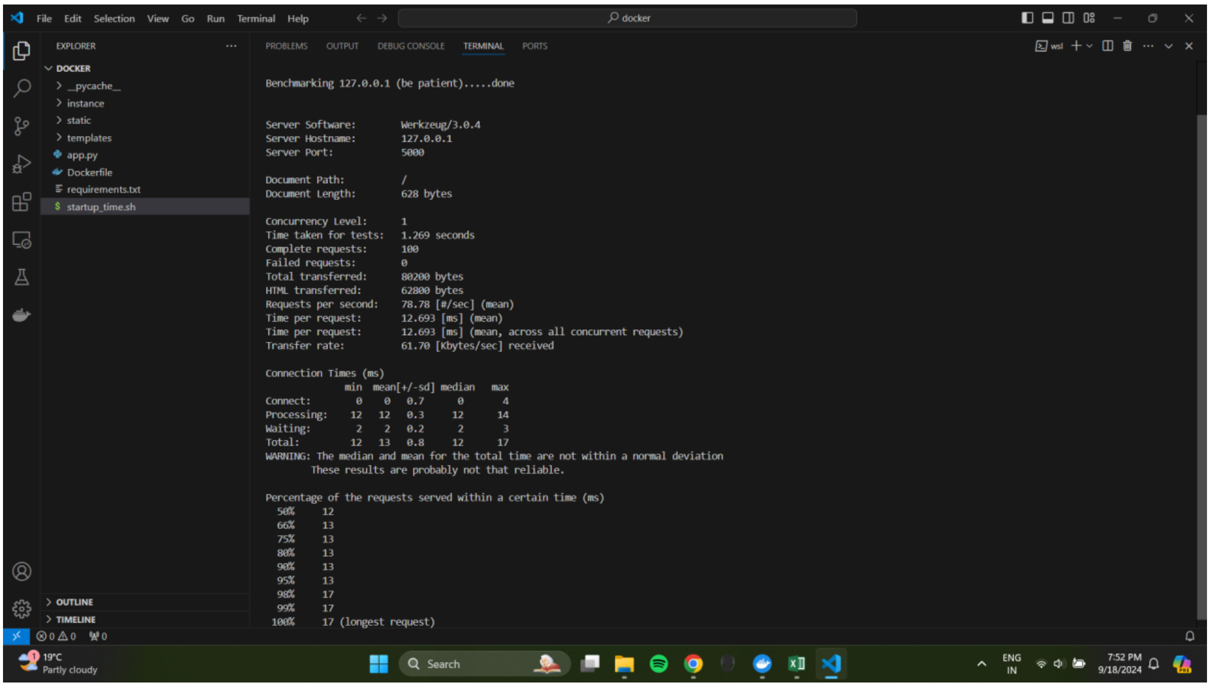
a. Firing 100 requests individually, one at a time.
b. Firing 100 requests concurrently, in a batch size of 10.

Surprisingly, I observed that for the first scenario, VMs were quicker than Containers to handle individual requests. On the other hand, Containers fared better with the second scenario. The metrics are represented in the table below.

Scenario	Metric	VMs	Containers
Individual	Request Throughput (Requests/sec)	217.63	78.78
	Median Response Time (ms)	4	12
	Max Response Time (ms)	7	17
Concurrent	Request Throughput (Requests/sec)	240.90	430.82
	Median Response Time (ms)	40	21
	Max Response Time (ms)	57	42

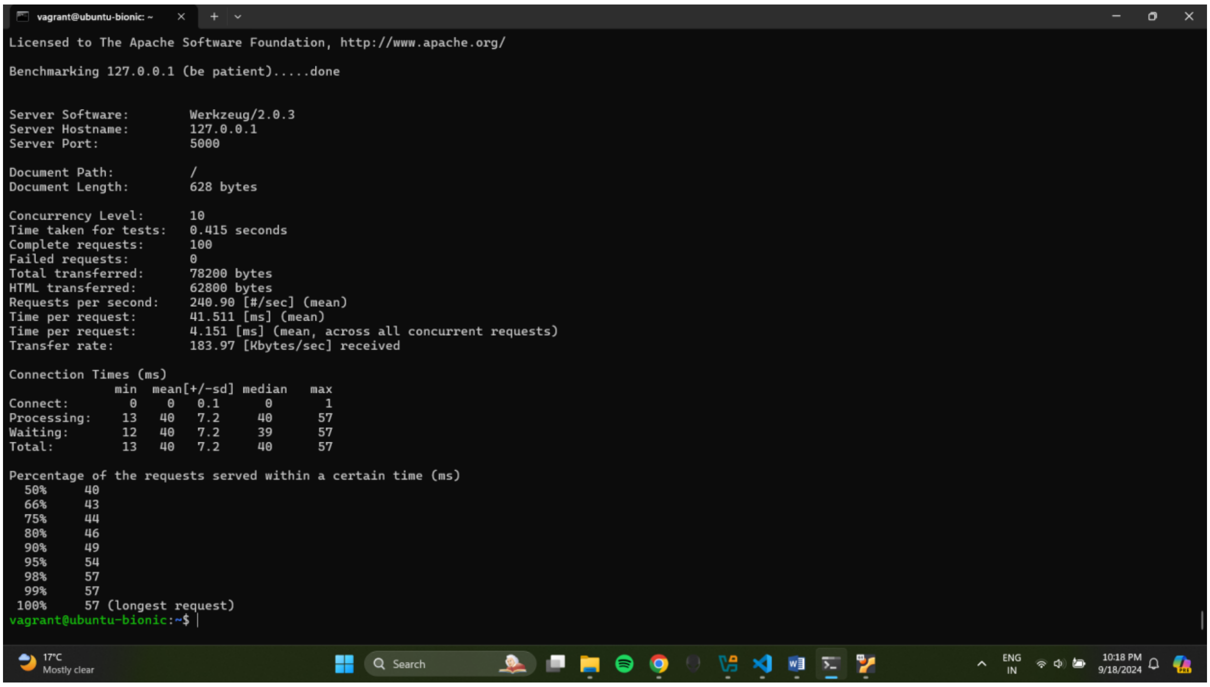


```
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient).....done


Server Software:        Werkzeug/3.0.4
Server Hostname:        127.0.0.1
Server Port:            5000

Document Path:          /
Document Length:        628 bytes

Concurrency Level:      10
Time taken for tests:   0.232 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      80200 bytes
HTML transferred:       62800 bytes
Requests per second:    430.82 [#/sec] (mean)
Time per request:       23.212 [ms] (mean)
Time per request:       2.321 [ms] (mean, across all concurrent requests)
Transfer rate:          337.42 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       0
Processing:    13   22   6.2     21      42
Waiting:        2    9   5.9      8      31
Total:         13   22   6.2     21      42

Percentage of the requests served within a certain time (ms)
  50%     21
  66%     24
  75%     26
  80%     26
  90%     30
  95%     34
  98%     42
  99%     42
 100%     42 (longest request)
```

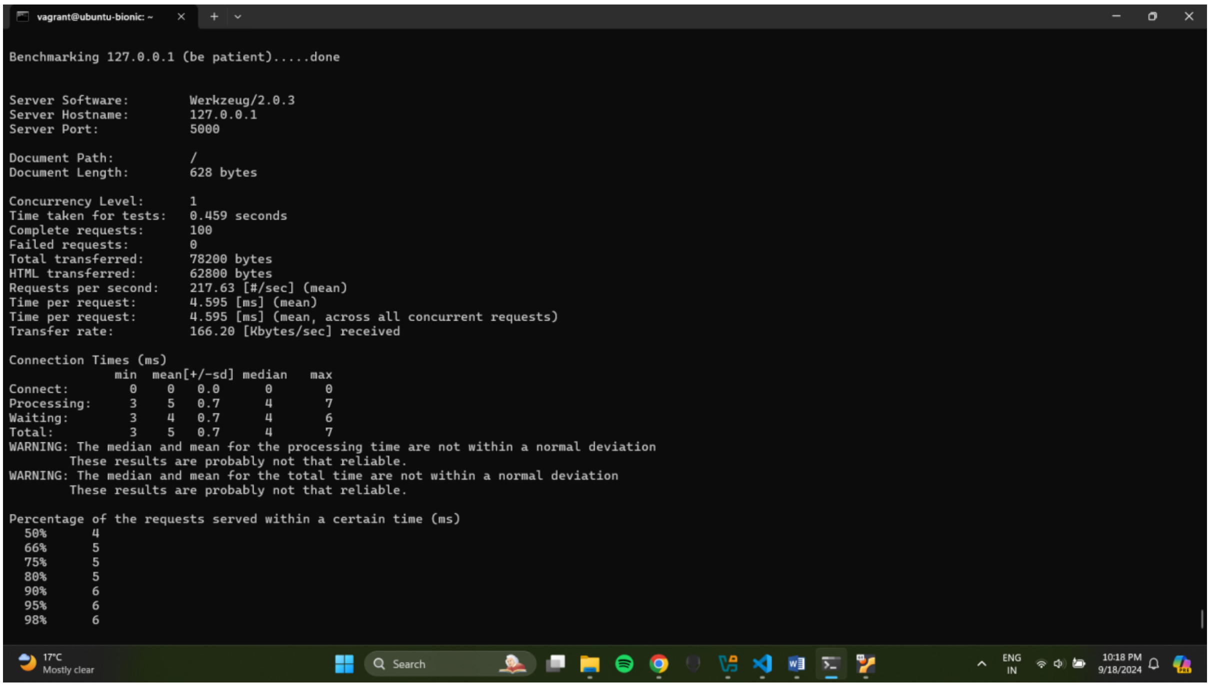


```
Benchmarking 127.0.0.1 (be patient).....done


Server Software:        Werkzeug/3.0.4
Server Hostname:        127.0.0.1
Server Port:            5000

Document Path:          /
Document Length:        628 bytes

Concurrency Level:      1
Time taken for tests:   1.269 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      80200 bytes
HTML transferred:       62800 bytes
Requests per second:    78.78 [#/sec] (mean)
Time per request:       12.693 [ms] (mean)
Time per request:       12.693 [ms] (mean, across all concurrent requests)
Transfer rate:          61.70 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.7      0       4
Processing:    12   12   0.3     12      14
Waiting:        2    2   0.2      2       3
Total:         12   13   0.8     12      17
WARNING: The median and mean for the total time are not within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%     12
  66%     13
  75%     13
  80%     13
  90%     13
  95%     13
  98%     17
  99%     17
 100%     17 (longest request)
```

These two images show the Apache Benchmark results for individual and concurrent requests scenarios in a Container environment respectively.



```
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient).....done


Server Software:        Werkzeug/2.0.3
Server Hostname:        127.0.0.1
Server Port:            5000

Document Path:          /
Document Length:        628 bytes

Concurrency Level:      10
Time taken for tests:   0.415 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      78200 bytes
HTML transferred:       62800 bytes
Requests per second:    240.90 [#/sec] (mean)
Time per request:       41.511 [ms] (mean)
Time per request:       4.151 [ms] (mean, across all concurrent requests)
Transfer rate:          183.97 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:    13   40   7.2     40      57
Waiting:       12   40   7.2     39      57
Total:         13   40   7.2     40      57

Percentage of the requests served within a certain time (ms)
  50%     40
  66%     43
  75%     44
  80%     46
  90%     49
  95%     54
  98%     57
  99%     57
 100%     57 (longest request)
vagrant@ubuntu-bionic:~$ 
```



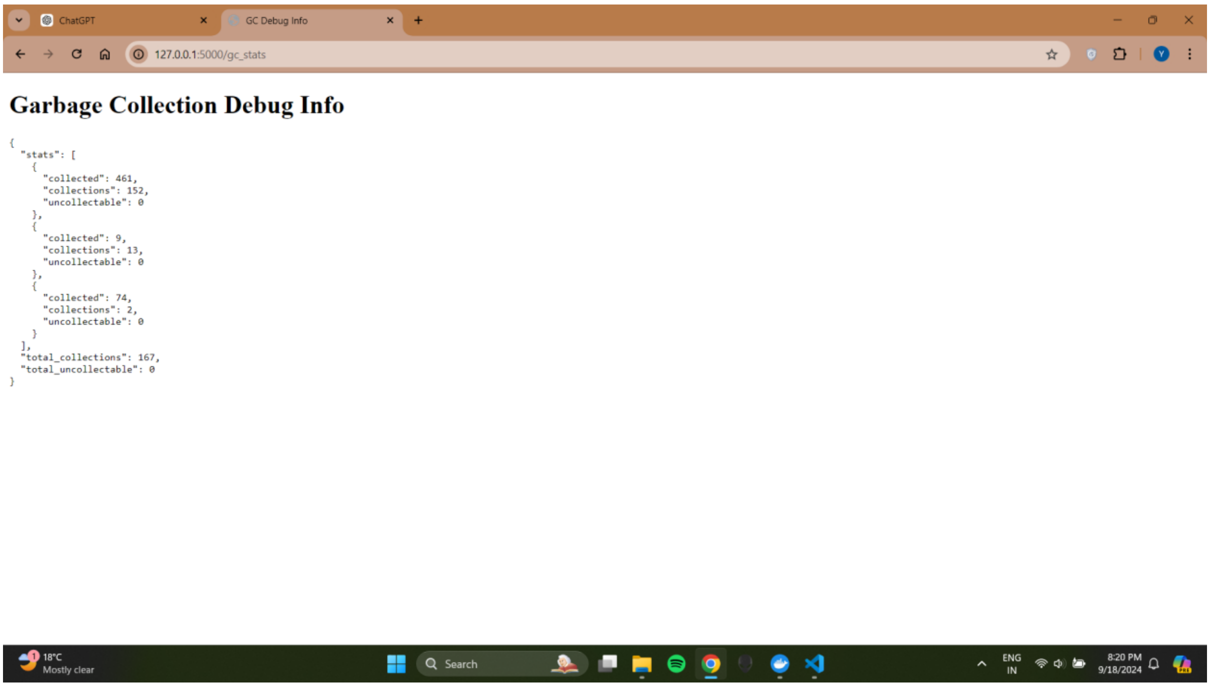
```
Benchmarking 127.0.0.1 (be patient).....done


Server Software:        Werkzeug/2.0.3
Server Hostname:        127.0.0.1
Server Port:            5000

Document Path:          /
Document Length:        628 bytes

Concurrency Level:      1
Time taken for tests:   0.459 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      78200 bytes
HTML transferred:       62800 bytes
Requests per second:    217.63 [#/sec] (mean)
Time per request:       4.595 [ms] (mean)
Time per request:       4.595 [ms] (mean, across all concurrent requests)
Transfer rate:          166.20 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:     3    5   0.7      4       7
Waiting:        3    4   0.7      4       6
Total:          3    5   0.7      4       7
WARNING: The median and mean for the processing time are not within a normal deviation
        These results are probably not that reliable.
WARNING: The median and mean for the total time are not within a normal deviation
        These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50%      4
  66%      5
  75%      5
  80%      5
  90%      6
  95%      6
  98%      6
```

These two images show the Apache Benchmark results for individual and concurrent requests scenarios in a VM environment respectively.

Garbage Collection

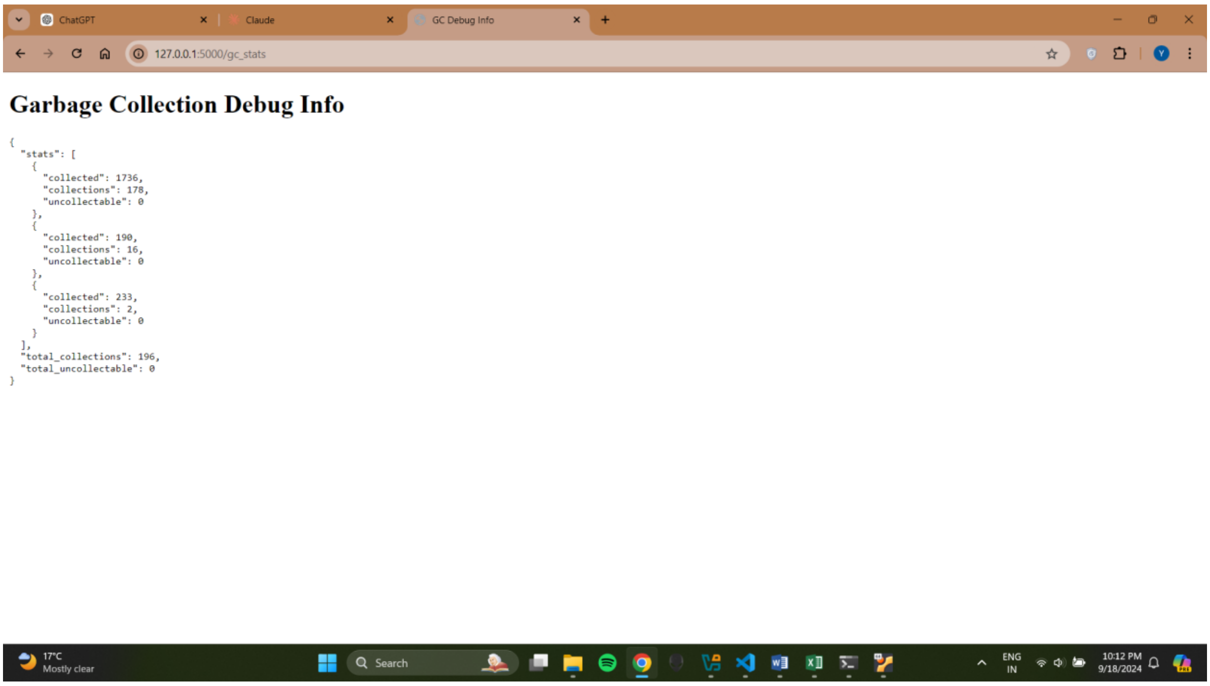
To track Garbage Collection, I implemented a unique route ('/gc_stats') from my main webpage to access the Garbage Collection stats. I implemented this using Flask (with GC import) the same way I implemented the rest of the application. To keep it as similar as possible, I performed the same set of tasks in a row (add contact, edit, delete) before checking garbage collection page. The difference observed isn't major but the VM did perform higher collections than the Container.

The results for the same are displayed below in the two images for the Container and VM respectively.



Garbage Collection Debug Info

```
{
  "stats": [
    {
      "collected": 461,
      "collections": 152,
      "uncollectable": 0
    },
    {
      "collected": 9,
      "collections": 13,
      "uncollectable": 0
    },
    {
      "collected": 74,
      "collections": 2,
      "uncollectable": 0
    }
  ],
  "total_collections": 167,
  "total_uncollectable": 0
}
```



Garbage Collection Debug Info

```
{
  "stats": [
    {
      "collected": 1736,
      "collections": 178,
      "uncollectable": 0
    },
    {
      "collected": 190,
      "collections": 16,
      "uncollectable": 0
    },
    {
      "collected": 233,
      "collections": 2,
      "uncollectable": 0
    }
  ],
  "total_collections": 196,
  "total_uncollectable": 0
}
```

Summary

To summarize my findings, while there are certainly scenarios where the metrics favoured VMs, they are overwhelmingly favourable to Containers, especially in terms of Startup Time and Memory Usage, both of which are quite important. While there might still have use cases that prefer VMs, Containers are largely better thanks to the extra OS abstraction and performance.