

# Exercise 6

*Simple Benchmarking with wrk*

## Prior Knowledge

Previous exercises

## Objectives

Benchmarking runtimes

## Software Requirements

- Java Development Kit 8
- Redis
- wrk - a simple benchmarking tool

## Overview

We will look at using a benchmarking tool to call our APIs very fast and see how they react.

### Steps

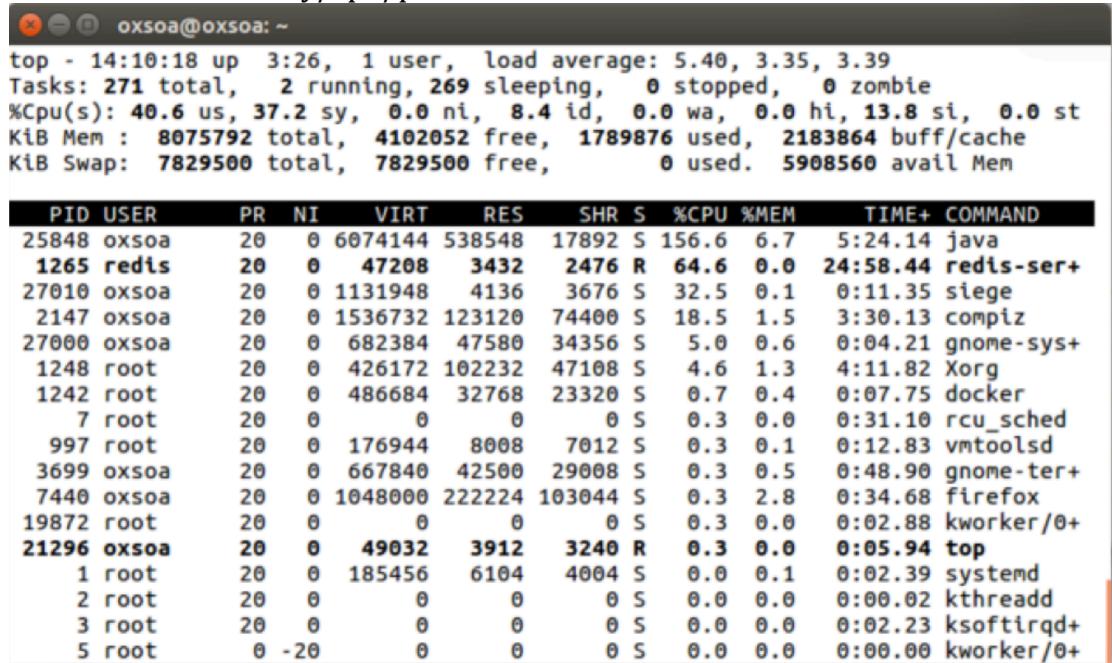
1. Create a directory for this work and clone the git repository:  
`mkdir ~/ex6`  
`cd ~/ex6`  
`git clone https://github.com/pzfereo/PSBComplete.git`
2. Make sure redis is running locally:  
`sudo service redis-server start`
3. Build the code  
`cd PSBComplete`  
`gradle build`
4. Run the “Uber JAR” as before
5. We can performance test our app. First lets install wrk, a simple HTTP performance test app:  
`sudo apt install -y wrk`
6. Now we can run a test:  
`wrk -c 100 -d 2m -t 10 http://localhost:8080/purchase`
7. This will constantly hit our server with 100 concurrent clients calling over 2 minute (using 10 threads).



8. While it is running you can monitor the CPU.

Open up a new terminal window and type:  
**top**

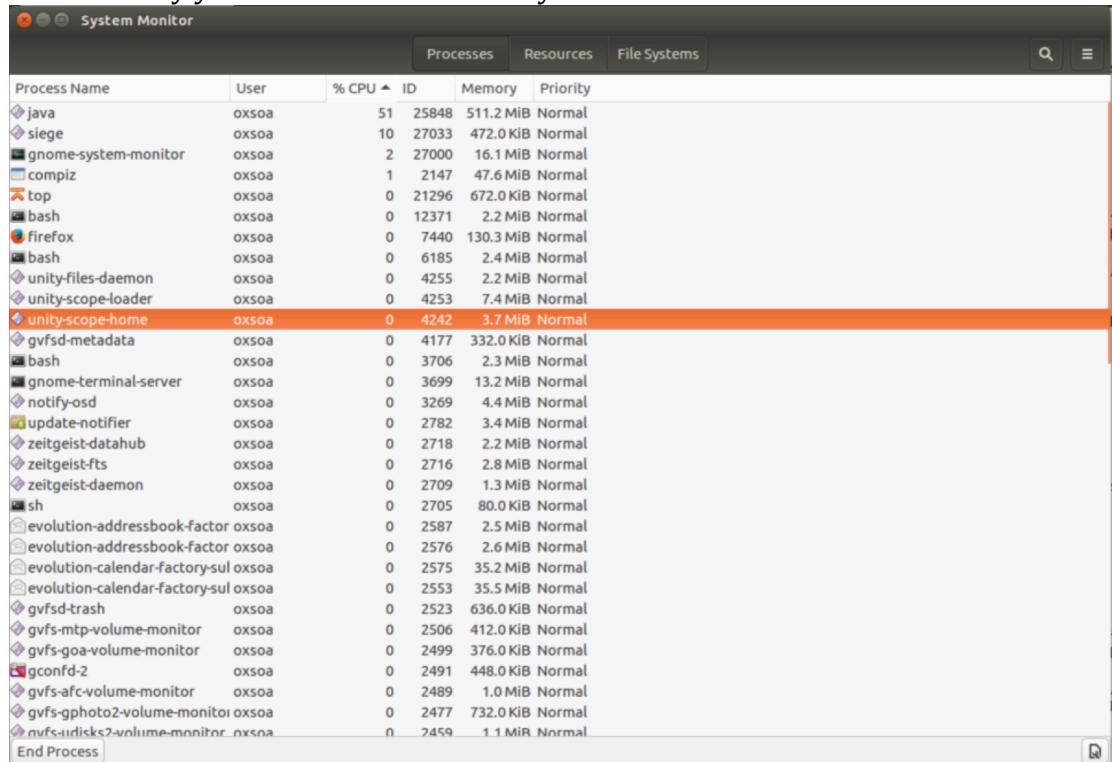
9. You will see a memory/cpu/process monitor.



```
oxsoa@oxsoa: ~
top - 14:10:18 up 3:26, 1 user, load average: 5.40, 3.35, 3.39
Tasks: 271 total, 2 running, 269 sleeping, 0 stopped, 0 zombie
%Cpu(s): 40.6 us, 37.2 sy, 0.0 ni, 8.4 id, 0.0 wa, 0.0 hi, 13.8 si, 0.0 st
KiB Mem : 8075792 total, 4102052 free, 1789876 used, 2183864 buff/cache
KiB Swap: 7829500 total, 7829500 free, 0 used. 5908560 avail Mem

      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
25848 oxsoa     20   0 6074144 538548 17892 S 156.6 6.7  5:24.14 java
1265  redis     20   0  47208  3432  2476 R 64.6 0.0 24:58.44 redis-ser+
27010 oxsoa     20   0 1131948  4136  3676 S 32.5 0.1  0:11.35 siege
2147  oxsoa     20   0 1536732 123120 74400 S 18.5 1.5  3:30.13 compiz
27000 oxsoa     20   0  682384  47580 34356 S 5.0 0.6  0:04.21 gnome-sys+
1248  root      20   0 426172 102232 47108 S 4.6 1.3  4:11.82 Xorg
1242  root      20   0 486684  32768 23320 S 0.7 0.4  0:07.75 docker
7  root       20   0      0      0      0 S 0.3 0.0  0:31.10 rcu_sched
997  root      20   0 176944  8008  7012 S 0.3 0.1  0:12.83 vmtoolsd
3699 oxsoa     20   0 667840  42500 29008 S 0.3 0.5  0:48.90 gnome-ter+
7440 oxsoa     20   0 1048000 222224 103044 S 0.3 2.8  0:34.68 firefox
19872 root     20   0      0      0      0 S 0.3 0.0  0:02.88 kworker/0+
21296 oxsoa     20   0  49032  3912  3240 R 0.3 0.0  0:05.94 top
1  root      20   0 185456  6104  4004 S 0.0 0.1  0:02.39 systemd
2  root      20   0      0      0      0 S 0.0 0.0  0:00.02 kthreadd
3  root      20   0      0      0      0 S 0.0 0.0  0:02.23 ksoftirqd+
5  root      0 -20      0      0      0 S 0.0 0.0  0:00.00 kworker/0+
```

10. Alternatively you can use the Ubuntu System Monitor:



11. Back in your wrk terminal window you should see it complete:

```
oxsoa@oxsoa:~/PSBComplete$ wrk -c 1000 -t 10 -d 1m http://localhost:8080/purchase
Running 1m test @ http://localhost:8080/purchase
  10 threads and 1000 connections
  Thread Stats      Avg      Stdev     Max   +/- Stdev
    Latency    540.59ms  389.74ms   2.00s   60.82%
    Req/Sec   187.00     74.79    797.00   70.07%
  111104 requests in 1.00m, 53.52MB read
  Socket errors: connect 0, read 0, write 0, timeout 708
Requests/sec: 1849.88
Transfer/sec:   0.89MB
```

12. You might want to re-run this again as the JVM will warm up (JIT compilation).

13. By default Spring Boot uses the Tomcat servlet engine as the server. However, that is pretty slow. There is a better engine called Undertow.

We can enable that just by changing the gradle build file.

We need to exclude the Tomcat and include undertow. The lines you need to change are commented out in the build gradle or you can see them below:

```
configurations {
    // exclude Tomcat
    compile.exclude module: 'spring-boot-starter-tomcat'
}

dependencies {
    implementation('org.springframework.boot:spring-boot-starter-undertow')
    implementation('org.springframework.boot:spring-boot-starter-jersey')
    implementation('com.fasterxml.jackson.core:jackson-databind')
```

14. Rebuild and then rerun the performance test. Run it twice to allow the JVM to warm up. How do they compare?

15. Note that this is not a real performance analysis. Ideally the servers would be on a separate machine from the client load drivers (siege engines!). Also, microservices are designed to be run in parallel in multiple containers with load balancing across them, so this model is not the recommended way of running either deployment.

16. That's all

