Exercise 7c

Trying out WSO2 MSF4J instead of Jetty/Jersey
Benchmarking with Siege

Prior Knowledge
Previous exercises

Objectives
Testing another JAX-RS implementation for Microservices
Benchmarking runtimes

Software Requirements

- Java Development Kit 8
- MSF4J
- Redis
- siege

Overview
MSF4J is a new framework for running JAX-RS code as a microservice.
It has a very simple model.
We will also 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 ~/ex7c
   cd ~/ex7c
   git clone https://github.com/pzfreo/POResourceComplete.git

2. Make sure redis is running locally:
   sudo service redis-server start

3. Build both the Jersey and MSF4J versions of the same Resource:
   cd POResourceComplete
   gradle shadowJar
   mvn clean install -Dmaven.test.skip=true

   Note: We are skipping tests because I didn't add logic into the maven build to start the microservice and stop it before and after tests. The MSF4J recommendation is to start the service in the test itself (since its so quick to start), but I didn't want to change the tests as they are shared with the gradle/jersey build.

   Hint: the mvn may take a while as it downloads half the internet to your VM

   Hint 2: While you are waiting for the downloads you can take a look at the MSF4J project site to understand it a bit better:
   https://github.com/wso2/msf4j/

   Hint 3: My pom.xml seems horribly more complex than the one on github. There are three simple reasons. Firstly, I'm using the latest build of MSF4J which isn't yet in the main repos. Secondly, I've added my dependencies (json, etc). Thirdly, I've had to exclude the build from looking at the Jersey Main.java class because that has its own set of dependencies.

4. Run the MSF4J version:
   java -jar target/POResource-1.1.0-SNAPSHOT.jar

5. See if it works (its on port 8080).

6. We can performance test our app. First lets install siege:
   sudo apt install -y siege

7. Now we can run a test:
   siege -b -c 15 -r 10000 http://localhost:8080/purchase

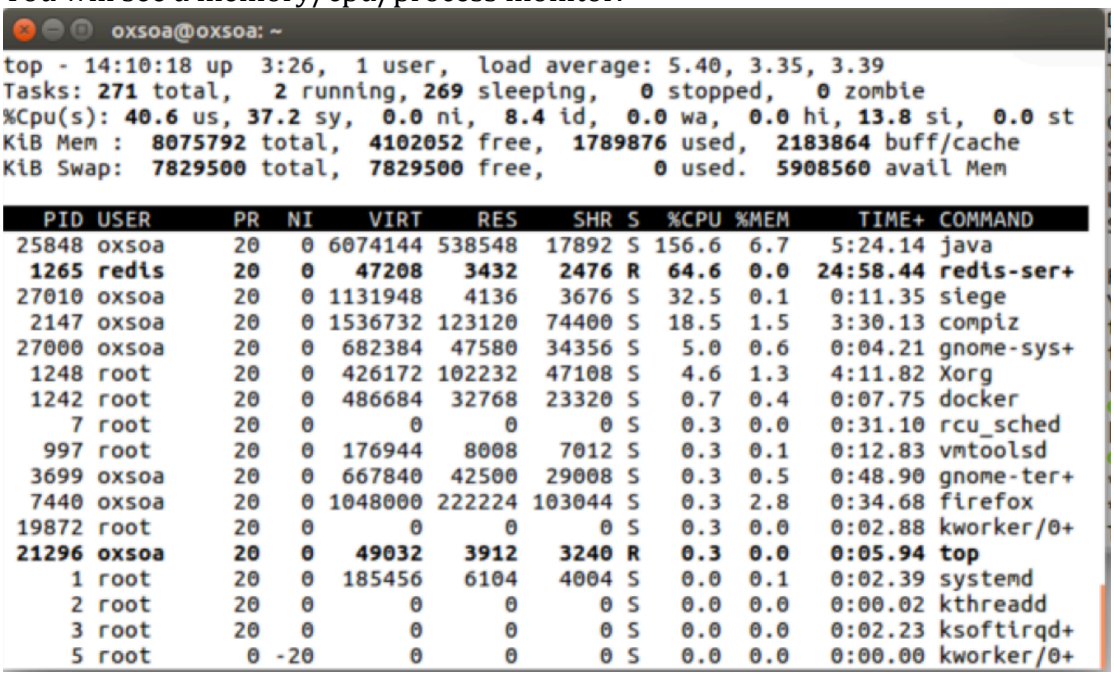
8. This will constantly hit our server with 15 concurrent clients each calling 10k times, in benchmark mode (i.e. each request hits immediately after the one before rather than being random).

9. You should see:

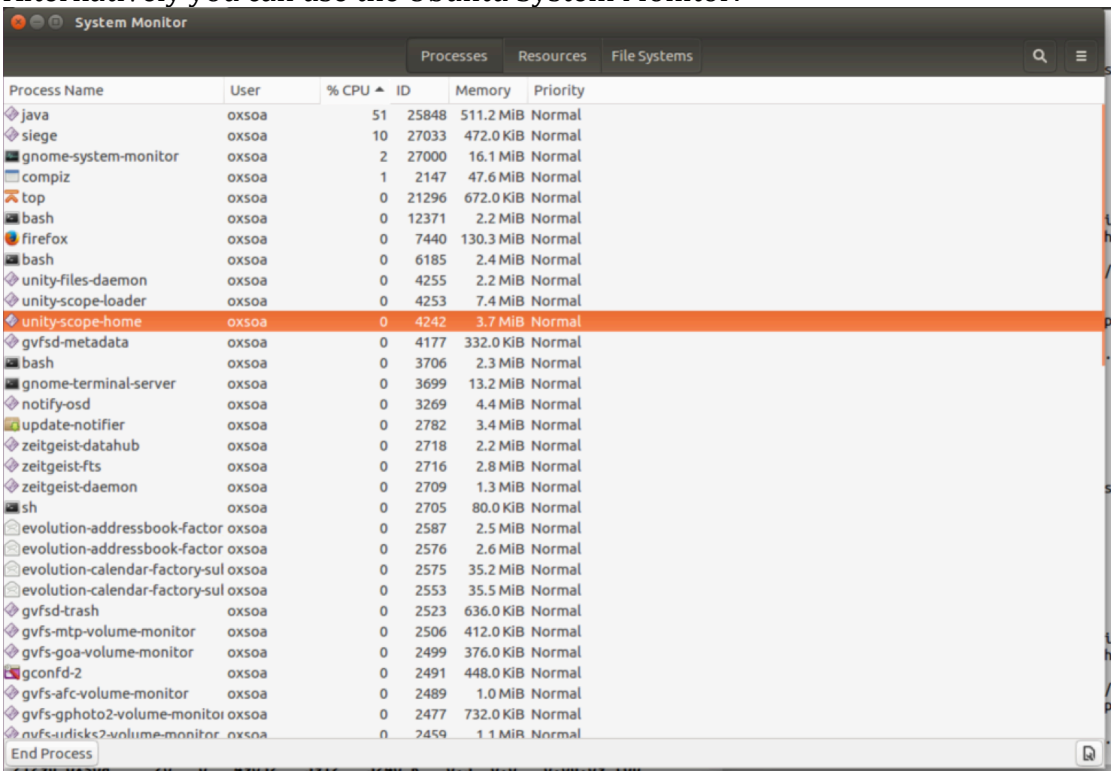
```
oxsoa@oxsoa:~$ siege -b -c 15 -r 10000  http://localhost:8080/purchase
** SIEGE 3.0.8
** Preparing 15 concurrent users for battle.
The server is now under siege..      done.

Transactions:                 150000 hits
Availability:                 100.00 %
Elapsed time:                  70.15 secs
Data transferred:              33.47 MB
Response time:                  0.01 secs
Transaction rate:            2138.28 trans/sec
Throughput:                     0.48 MB/sec
Concurrency:                   14.80
Successful transactions:      150000
Failed transactions:               0
Longest transaction:            0.10
Shortest transaction:           0.00

FILE: /var/log/siege.log
You can disable this annoying message by editing
the .siegerc file in your home directory; change
the directive 'show-logfile' to false.
[error] unable to create log file: /var/log/siege.log: Permission denied
```

10. Open up a new terminal window and type:
    top

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



12. Alternatively you can use the Ubuntu System Monitor:



13. Re-run the siege and check on the system performance as the system is under load.

14. Now run the Jersey / Jetty server instead
    java -jar build/libs/POResourceComplete-all.jar

15. Re-run the siege and compare the memory and tps performance.

16. 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.

17. That's all