**Exercise 8**

*Simple Benchmarking with autocannon*

**Prior Knowledge**

Previous exercises

**Objectives**

Benchmarking runtimes

**Software Requirements**

* docker-compose
* autocannon - 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

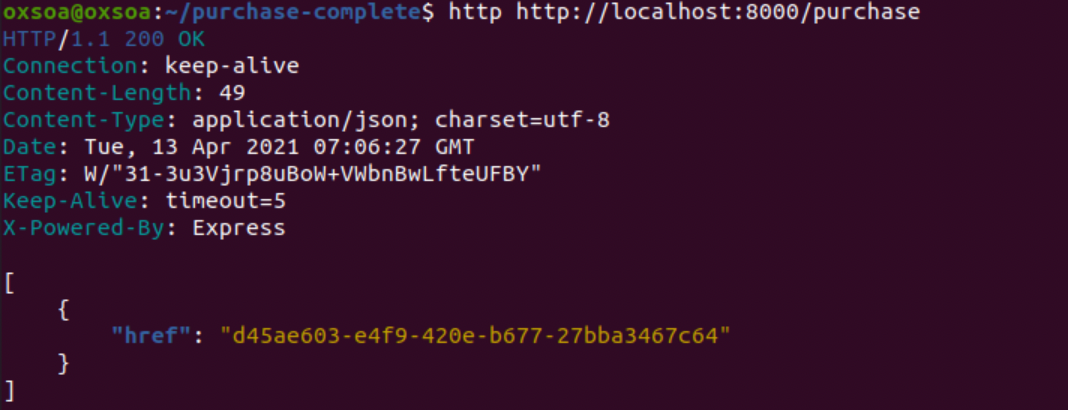
1. Start your service using “docker-compose up”

(If you don’t have a working service but want to try this anyway then do this:  
  
cd ~   
git clone <https://github.com/pzfreo/purchase-complete.git>

cd purchase-complete

yarn install  
docker-compose up --build

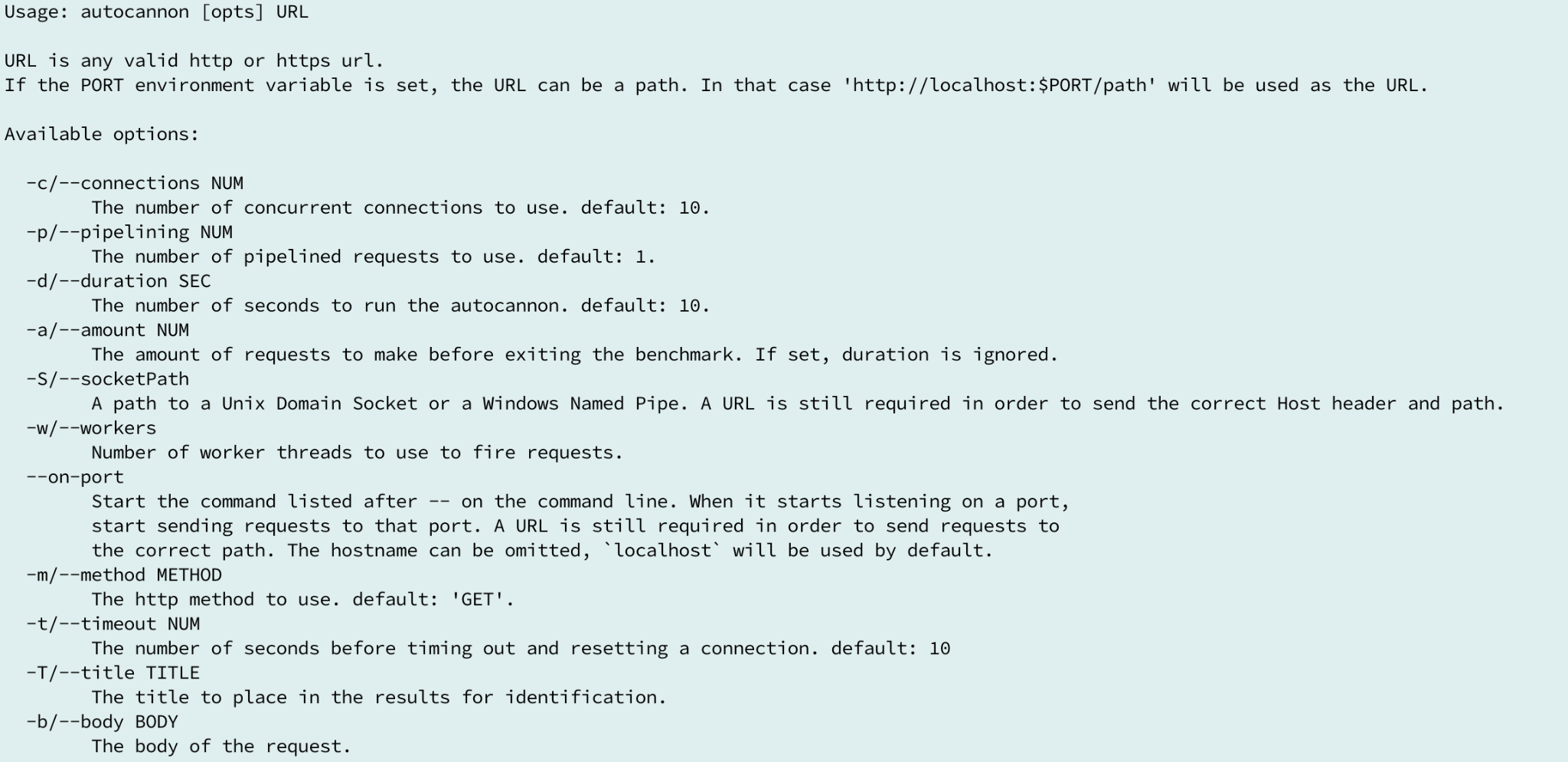
)

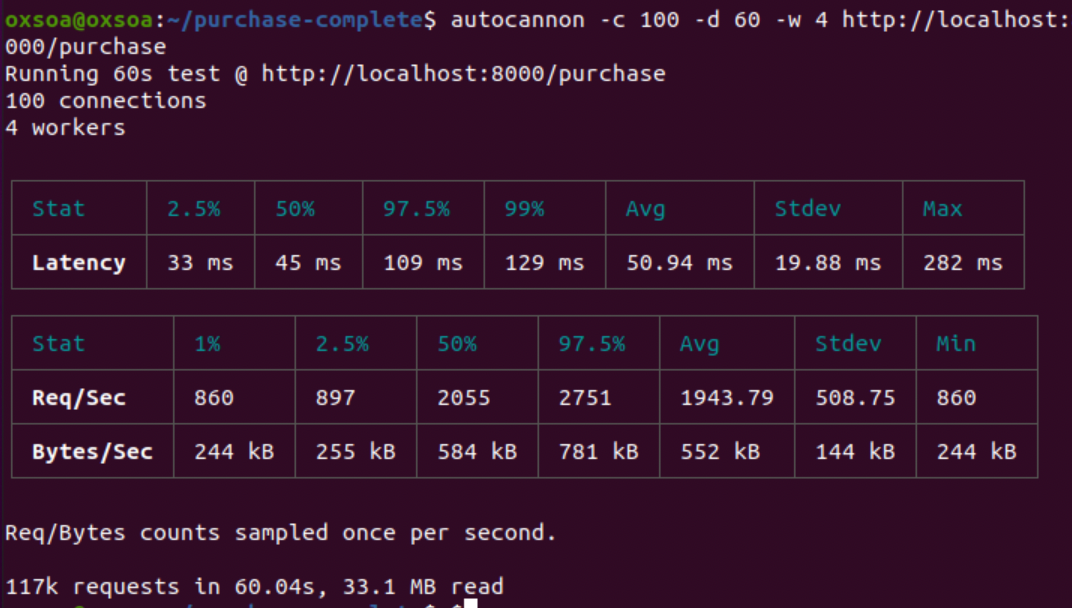
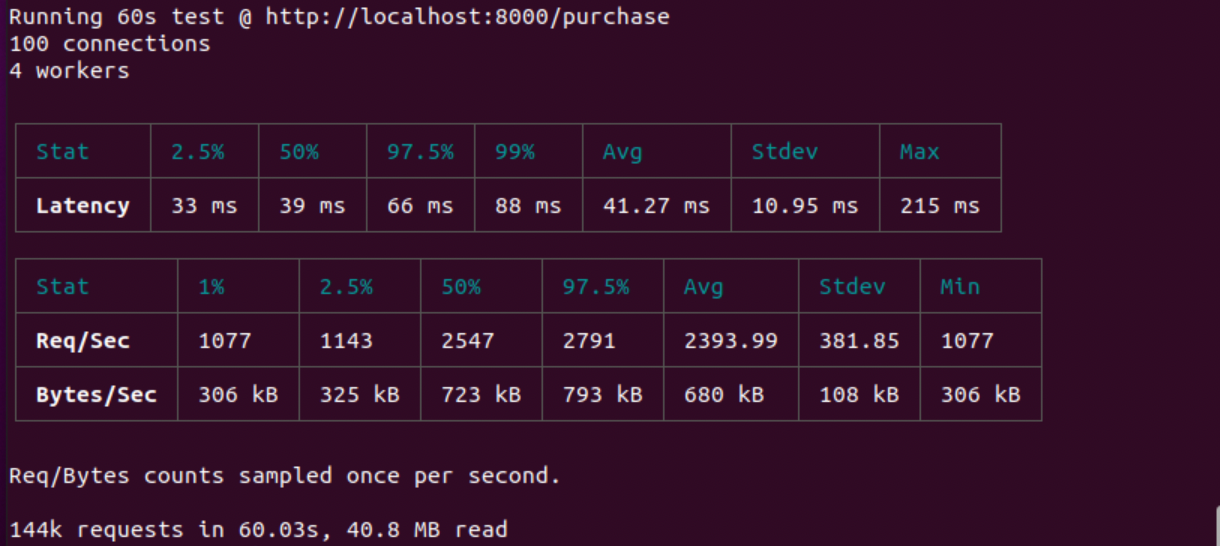
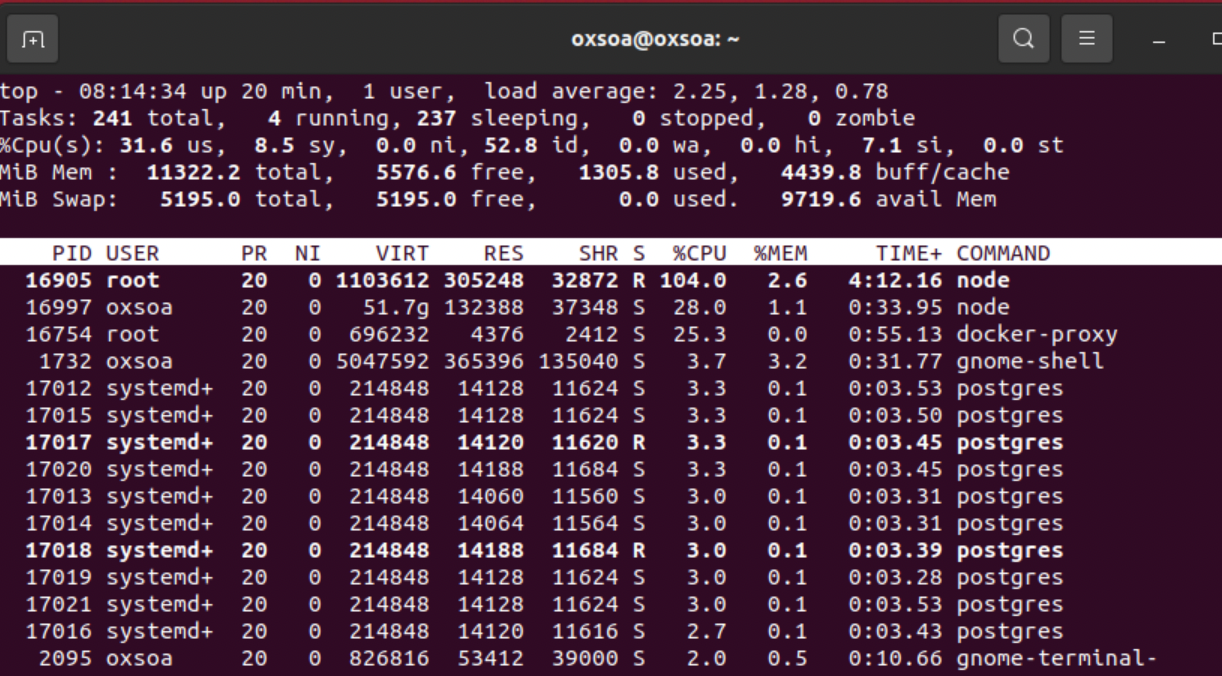
Test your service is up and running:  
http <http://localhost:8000/purchase>  


1. Let’s install autocannon.  
     
   *Autocannon needs a newer version of node than is installed, so let’s upgrade:*  
   curl -sL https://deb.nodesource.com/setup\_14.x | sudo -E bash -

sudo apt install nodejs

Now install autocannon:

yarn global add autocannon  
  
autocannon --help  
  


1. Now we can run a test:  
   autocannon -c 100 -d 60 -w 4 http://localhost:8000/purchase
2. This will constantly hit our server with 100 concurrent clients calling over 60 seconds (using 4 worker threads).
3. You should see something like:  
   
4. In my run I had zero errors.   
   If there are errors there will be a line like:  
     
   120274 2xx responses, 97 non 2xx responses
5. Rerun it now everything is warmed up.  
     
     
     
   I think ~2400 requests per second accessing a database is reasonable. You may get different results on your setup of course. Overall this shows we have built a highly performant (and hopefully scalable) application.
6. While it is running you can monitor the CPU.   
   Extend the time to run longer (e.g. 300s) and rerun  
     
   Open up a new terminal window and type:  
   top
7. You will see a memory/cpu/process monitor.   
     
   If you want to read more about load averages, this is a good read: <http://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html>

1. You will see that there is are two node processes here.  
   One for the autocannon and one for the server  
     
   Node is single-threaded, so on a multi-core system you might want to run more processes for the server. In a Kubernetes environment you would run multiple replicas behind a load-balancer. In other systems you can use tools like pm2 to automatically scale up node instances:  
     
    <https://github.com/Unitech/pm2>
2. 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.
3. That’s all for this lab!

**Extension**If you want to try a similar benchmarking tool written in go, try:

<https://github.com/codesenberg/bombardier>  
  
It should be pretty similar to get running.