DevOps Exercise

The files checked into my GitHub repository: https://github.com/yellowzoneallan/buildit

Dockerfile	Buildit code as docker image
README.md	Initial commit
Vagrantfile	Vagrant config
docker-compose.yaml	Docker Compose Config
nginx.lb.conf	Nginx conf to be a load balancer 80 -> dockers:3000
setup.sh	Build the buildit system

The docker images in my solution were:

- git - a private GitHub for testing
- Jenkins - a private Jenkins for testing
- builditapp - the example code wrapped in an image
- nginx - nginx importing a config file to covert it to a load balancer
- russmckendrick/ab - random prepackaged apache benchmark image

This alias was setup so “localgit” commands were identical to “git” commands

```
alias localgit='sudo docker run -ti --rm -v ${HOME}:/root -v $(pwd):/git alpine/git “$@"'
```

The Vagrant and docker-compose file combination allows browser access to 192.168.3.3

Docker-compose allows the number of instances to vary, in this example 10 were started and apache benchmark shows 10,000 successful hits in 5.6s.

```
vagrant@ubuntu-eoan:~$ sudo docker-compose up --scale builditapp=10 -d
Creating network "vagrant_default" with the default driver
Creating vagrant_builditapp_1  ... done
Creating vagrant_builditapp_2  ... done
Creating vagrant_builditapp_3  ... done
Creating vagrant_builditapp_4  ... done
Creating vagrant_builditapp_5  ... done
Creating vagrant_builditapp_6  ... done
Creating vagrant_builditapp_7  ... done
Creating vagrant_builditapp_8  ... done
Creating vagrant_builditapp_9  ... done
Creating vagrant_builditapp_10 ... done
Creating vagrant_jenkins_1     ... done
Creating vagrant_localgit_1    ... done
Creating vagrant_nginx_1       ... done
vagrant@ubuntu-eoan:~$ sudo docker run russmckendrick/ab ab -k -n 10000 -c 16 http://
192.168.3.3/
This is ApacheBench, Version 2.3 <$Revision: 1826891 $>
Finished 10000 requests
Server Software:        nginx/1.17.9
Server Hostname:        192.168.3.3
Server Port:            80
Document Path:          /
Document Length:        44 bytes
Concurrency Level:      16
Time taken for tests:   5.672 seconds
```

```
Complete requests:      10000
Failed requests:        0
Keep-Alive requests:    0
Total transferred:      1410000 bytes
HTML transferred:       440000 bytes
Requests per second:    1763.04 [#/sec] (mean)
Time per request:       9.075 [ms] (mean)
Time per request:       0.567 [ms] (mean, across all concurrent requests)
Transfer rate:          242.76 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.9      0       7
Processing:     1    8   4.0      8      66
Waiting:        0    8   4.1      8      66
Total:          1    9   3.8      8      66
WARNING: The median and mean for the initial connection 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%      8
  90%     11
  99%     24
 100%     66 (longest request)
vagrant@ubuntu-eoan:~$
```

Platform Engineer task

There already exists an excellent open source product which achieves the task objectives and much more. The application is packaged as a java jar, which can be scaled effectively by selecting larger VMs, JVMs and the java threadpooling, garbage collection and heap tuning parameters.

Via a browser on your laptop https://darcyripper.com/features/downloads/
Then extract the zip and click the darcyripper.jar, it is obvious what to do.

If the exercise is about automation here's the ubuntu linux commands needed:

```
sudo curl --insecure -L "https://darcyripper.com/download/1230/darcyripper.zip" -o
darcyripper.zip # download ripper
sudo apt install -y openjdk-14-jre-headless # install java
sudo apt install -y unzip # install unzip
unzip darcyripper.zip; cd darcyripper/ # unzip ripper
```