CLD Lab 02: App scaling on IaaS

Authors: Edwin Häffner and Arthur Junod

Group: L2GrE

Date: 12 March 2024

TASK 1: CREATE A DATABASE USING THE RELATIONAL DATABASE SERVICE (RDS)

- Edwin's credential for the database

  username : edwintropcool password : 23vzug189we!p&

- Arthur's credential for database

  master username: junadmin / master password: super$1386!laf0li3

> Pricing for our instance

RDS PRICING

db.t3.micro: $0.017/h

gp2 storage: $0.115/GB-month

db-storage: 20GiB

> Commands to connect to the DB from the E2C instance

mysql --host=gre-haffner-wordpress-db.crsk2uw660uh.us-east-1.rds.amazonaws.com --user=edwintropcool --password='23vzug189we!p&'

mysql --host=gre-junod-wordpress-db.crsk2uw660uh.us-east-1.rds.amazonaws.com --user=junadmin --password='super$1386!laf0li3'

DELIVERABLE 1:

> Copy the estimated monthly cost for the database and add it to your report.

Since we don't have the estimated cost we can calculate it ourselve :

$$20 * \$0.115 + 730 * \$0.017 = \$14.71/month$$

> Compare the costs of your RDS instance to a continuously running EC2 instance of the same instance type to see how much AWS charges for the extra functionality.

EC2 pricing

t3.micro: $0.0104/h

gp3: $0.08/GB-month

$$20 * \$0.08 + 730 * \$0.0104 = \$9.192/month$$

We can see that the database is 5.518$ more expensive than our EC2 instance. It's around 50% more expensive to get the database specific functionalities.

> In a two-tier architecture the web application and the database are kept separate and run on different hosts. Imagine that for the second tier instead of using RDS to store the data you would create a virtual machine in EC2 and install and run yourself a database on it. If you were the Head of IT of a medium-size business, how would you argue in favor of using a database as a service instead of running your own database on an EC2 instance? How would you argue against it?

Using a database as a service would mean less maintenantance and operational overhead for the team. We just leave it to the cloud provider, Amazon here, to take care of updating the database software and so on.

The only issue with using RDS would be that we have less control over what we can do with the database, so if our use case is very specific, we may need to still use a EC2 instance to manage the DB. It's also more expensive to use than the EC2 instance if we don't factor in the additional time someone will need to put in to configure the DB manually.

> Copy the endpoint address of the database into the report.

- Edwin :

  gre-haffner-wordpress-db.crsk2uw660uh.us-east-1.rds.amazonaws.com

- Arthur:

  gre-junod-wordpress-db.crsk2uw660uh.us-east-1.rds.amazonaws.com

TASK 2: CONFIGURE THE WORDPRESS MASTER INSTANCE TO USE THE RDS DATABASE

> Arthur's credential for Wordpress:

username: junadmin / password: nul$m4uX!d3P4553

> Edwin's credential for Wordpress:

username: edwintropcoolwp / password: v26tKc*FGQIsTG9KL^

> Copy the part of /var/www/html/wp-config.php that configures the database into the report.

Edwin's configuration :

```
// ** Database settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'GrE_Haffner_Wordpress_DB' );

/** Database username */
define( 'DB_USER', 'edwintropcool' );

/** Database password */
define( 'DB_PASSWORD', '23vzug189we!p&' );

/** Database hostname */
define( 'DB_HOST', 'gre-haffner-wordpress-db.crsk2uw660uh.us-east-1.rds.amazonaws.com' );
```

Arthur's configuration:

```
// ** Database settings - You can get this info from your web host ** //
/** The name of the database for WordPress */
define( 'DB_NAME', 'GrE_Junod_Wordpress_DB' );

/** Database username */
define( 'DB_USER', 'junadmin' );

/** Database password */
define( 'DB_PASSWORD', 'super$1386!laf0li3' );

/** Database hostname */
define( 'DB_HOST', 'gre-junod-wordpress-db.crsk2uw660uh.us-east-1.rds.amazonaws.com' );
```

TASK 3: CREATE A CUSTOM VIRTUAL MACHINE IMAGE

> Copy a screenshot of the AWS console showing the AMI parameters into the report.

Edwin:

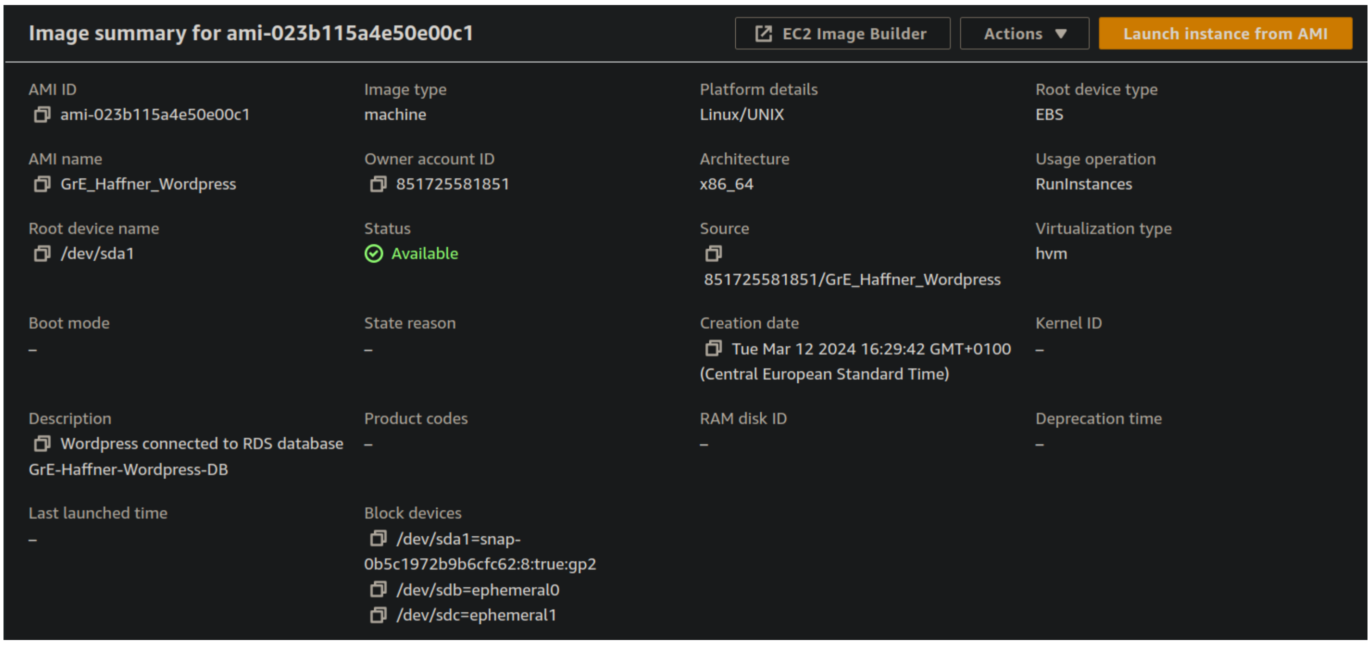


Image summary for ami-023b115a4e50e00c1

AMI ID: ami-023b115a4e50e00c1
Image type: machine
Platform details: Linux/UNIX
Root device type: EBS
AMI name: GrE_Haffner_Wordpress
Owner account ID: 851725581851
Architecture: x86_64
Usage operation: RunInstances
Root device name: /dev/sda1
Status: Available
Source: 851725581851/GrE_Haffner_Wordpress
Virtualization type: hvm
Boot mode: –
State reason: –
Creation date: Tue Mar 12 2024 16:29:42 GMT+0100 (Central European Standard Time)
Kernel ID: –
Description: Wordpress connected to RDS database GrE-Haffner-Wordpress-DB
Product codes: –
RAM disk ID: –
Deprecation time: –
Last launched time: –
Block devices: /dev/sda1=snap-0b5c1972b9b6cfc62:8:true:gp2 /dev/sdb=ephemeral0 /dev/sdc=ephemeral1

Arthur:

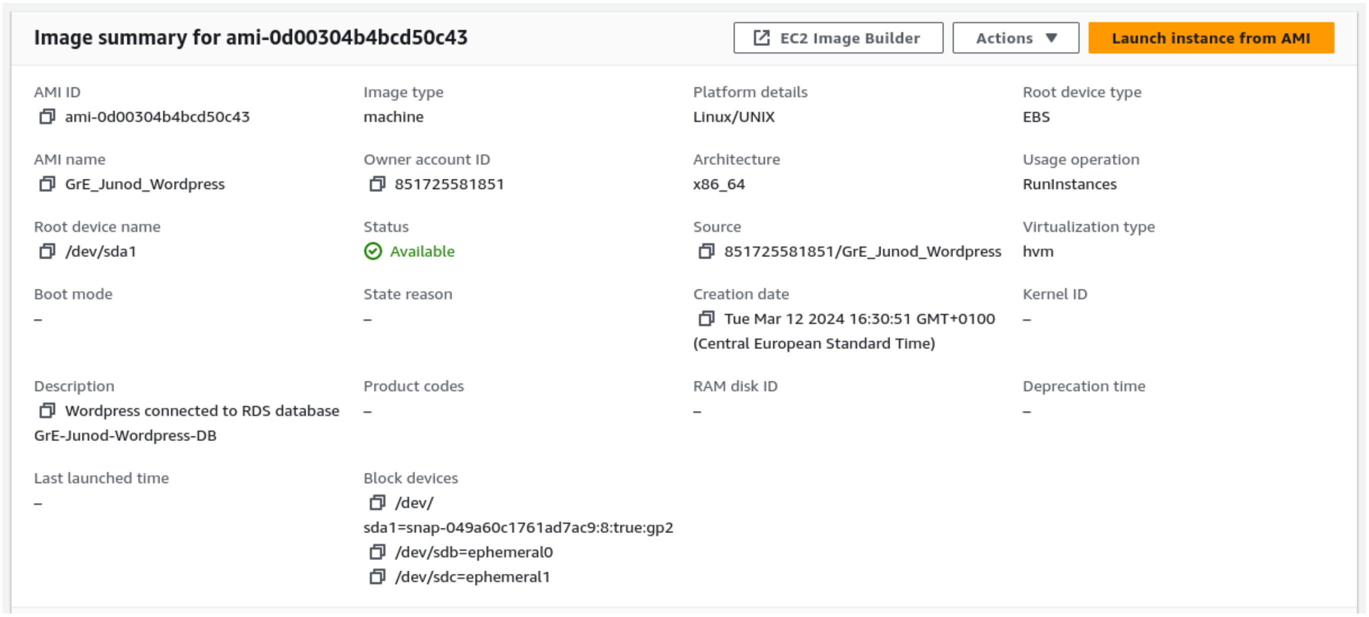


Image summary for ami-0d00304b4bcd50c43

AMI ID: ami-0d00304b4bcd50c43
Image type: machine
Platform details: Linux/UNIX
Root device type: EBS
AMI name: GrE_Junod_Wordpress
Owner account ID: 851725581851
Architecture: x86_64
Usage operation: RunInstances
Root device name: /dev/sda1
Status: Available
Source: 851725581851/GrE_Junod_Wordpress
Virtualization type: hvm
Boot mode: –
State reason: –
Creation date: Tue Mar 12 2024 16:30:51 GMT+0100 (Central European Standard Time)
Kernel ID: –
Description: Wordpress connected to RDS database GrE-Junod-Wordpress-DB
Product codes: –
RAM disk ID: –
Deprecation time: –
Last launched time: –
Block devices: /dev/sda1=snap-049a60c1761ad7ac9:8:true:gp2 /dev/sdb=ephemeral0 /dev/sdc=ephemeral1

TASK 4: CREATE A LOAD BALANCER

> Verifiy and note in which availability zone your instance is running.

Arthur and Edwin's availability zone: us-east-1d

> DNS names

- Edwin : GrE-Haffner-LoadBalancer-1400508041.us-east-1.elb.amazonaws.com
- Arthur : GrE-Junod-LoadBalancer-284381652.us-east-1.elb.amazonaws.com

> On your local machine resolve the DNS name of the load balancer into an IP address using the nslookup command (works on Linux, macOS and Windows). Write the DNS name and the resolved IP Address(es) into the report.

- Edwin :

  ```
  nslookup GrE-Haffner-LoadBalancer-1400508041.us-east-1.elb.amazonaws.com
  Server:         10.193.64.16
  Address:        10.193.64.16#53
  ```

Non-authoritative answer: Name: GrE-Haffner-LoadBalancer-1400508041.us-east-1.elb.amazonaws.com Address: 34.201.231.65 Name: GrE-Haffner-LoadBalancer-1400508041.us-east-1.elb.amazonaws.com Address: 3.221.243.171

~~~
- Arthur :
```bash
Server:         127.0.0.53
Address:    127.0.0.53#53

Non-authoritative answer:
Name:    GrE-Junod-LoadBalancer-284381652.us-east-1.elb.amazonaws.com
Address: 18.233.227.68
Name:    GrE-Junod-LoadBalancer-284381652.us-east-1.elb.amazonaws.com
Address: 3.229.147.95
~~~

> In the Apache access log identify the health check accesses from the load balancer and copy some samples into the report.

- Arthur :

  ```
  172.31.50.225 - - [12/Mar/2024:16:04:47 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  172.31.26.116 - - [12/Mar/2024:16:04:55 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  172.31.50.225 - - [12/Mar/2024:16:04:57 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  172.31.26.116 - - [12/Mar/2024:16:05:05 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  172.31.50.225 - - [12/Mar/2024:16:05:07 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  172.31.26.116 - - [12/Mar/2024:16:05:15 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  ```

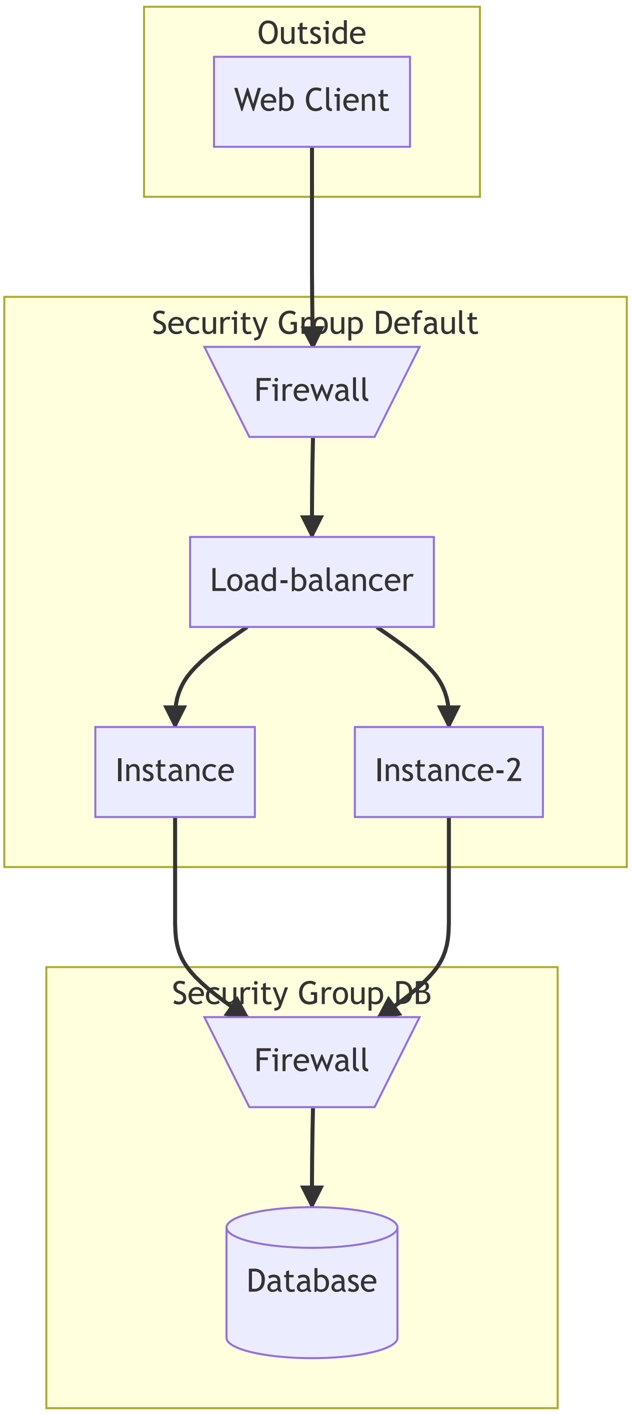
- Edwin :

  ```
  172.31.56.123 - - [19/Mar/2024:13:51:15 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  172.31.21.98  - - [19/Mar/2024:13:51:18 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  172.31.56.123 - - [19/Mar/2024:13:51:25 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  172.31.21.98  - - [19/Mar/2024:13:51:28 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  172.31.56.123 - - [19/Mar/2024:13:51:35 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  172.31.21.98  - - [19/Mar/2024:13:51:38 +0000] "GET / HTTP/1.1" 200 3423 "-" "ELB-HealthChecker/2.0"
  ```

TASK 5: LAUNCH A SECOND INSTANCE FROM THE CUSTOM IMAGE

We had to add the availability zone us-east-1c to the load balancer, so that our newly created instance (the second one from the image) could be used by it.

> Draw a diagram of the setup you have created showing the components (instances, database, load balancer, client) and how they are connected. Include the security groups as well. Make sure to show every time a packet is filtered.



> Calculate the monthly cost of this setup. You can ignore traffic costs.

We can take the previous result we got for our database :

$$\$14.71/month$$

For the EC2 instances we have :

- Instance Type: t2.micro
- Instance Count: 2
- Cost per Hour: $0.0116 (Linux pricing in us-east-1)
- Cost per Month:

$$2 * \$0.0116 * 730 \approx \$16.94$$

We also have EBS volumes :

- Volume Size: 8GB x 2 instances = 16GB
- Cost per GB-month for gp3 volumes: $0.08
- Cost per Month:

$$16 * \$0.08 = \$1.28$$

And finally the Application Load Balancer:

- Cost per Hour: $0.0225
- Cost per Month:

$$\$0.0225 * 730 \approx \$16.43$$

So in total we have a monthly cost of:

$$\$14.71 + \$16.94 + \$1.28 + \$16.43 = \underline{\underline{\$49.36}}$$

Which is a lot !!!!

TASK 5B: DELETE AND RE-CREATE THE LOAD BALANCER USING THE COMMAND LINE INTERFACE

> Listing the important info with the `describe-load-balancers`

- Edwin's AWS CLI CREDENTIAL

    - AWS Access Key ID : AKIA4MTWM7YNYL5ETD4Q
    - AWS Secret Access Key : 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

- Arthur's AWS CLI CREDENTIAL

    - AWS Access Key ID: AKIA4MTWM7YNZ4LTU32N
    - AWS Secret Access Key: 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

    > Put the commands to delete the load balancer, re-create the load balancer and re-create the listener into the report.

To delete the load balancer use:

```
aws elbv2 delete-load-balancer --load-balancer-arn <load-balancer-arn>
```

Then to recreate the load balancer:

```
aws elbv2 create-load-balancer \
    --name <load-balancer-name> \
    --subnets <subnet-id-1> <subnet-id-2> \
    --security-groups <security-group-id> \
    --scheme internet-facing \
    --type application \
    --ip-address-type ipv4
```

To recreate the listener use :

```
aws elbv2 create-listener \
    --load-balancer-arn <new-load-balancer-arn> \
    --protocol HTTP \
    --port 80 \
    --default-actions Type=forward,TargetGroupArn=<target-group-arn>
```

- Edwin:

To delete:

```
aws elbv2 delete-load-balancer --load-balancer-arn arn:aws:elasticloadbalancing:us-east-
1:851725581851:loadbalancer/app/GrE-Haffner-LoadBalancer/3cc0e3bc5c8575c8
```

To re-create the load balancer:

```
aws elbv2 create-load-balancer \
    --name GrE-Haffner-LoadBalancer \
    --subnets subnet-07f74df2f9ca79cef subnet-0a2ab628966261f50 \
    --security-groups sg-01c1a68b7baca6500                                                  \
    --scheme internet-facing \
    --type application \
    --ip-address-type ipv4
```

To re-create the listener:

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:us-east-1:851725581851:loadbalancer/app/GrE-Haffner-
LoadBalancer/583fd0fde6e67f63 \
    --protocol HTTP \
    --port 80 \
    --default-actions Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:us-east-
1:851725581851:targetgroup/GrE-Haffner-TargetGroup/e2ec614fcfbc81a4
```

- Arthur:

To delete:

```
aws elbv2 delete-load-balancer --load-balancer-arn arn:aws:elasticloadbalancing:us-east-
1:851725581851:loadbalancer/app/GrE-Junod-LoadBalancer/1352f144604e66d0
```

To re-create the load balancer:

```
aws elbv2 create-load-balancer \
    --name GrE-Junod-LoadBalancer \
    --subnets subnet-07f74df2f9ca79cef subnet-0a2ab628966261f50 \
    --security-groups sg-04328f385055d83c5 sg-0d6118cda1b7804f8 \
    --scheme internet-facing \
    --type application \
    --ip-address-type ipv4
```

To re-create the listener:

```
aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:us-east-1:851725581851:loadbalancer/app/GrE-Junod-
LoadBalancer/1352f144604e66d0 \
    --protocol HTTP \
    --port 80 \
    --default-actions Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:us-east-
1:851725581851:targetgroup/GrE-Junod-TargetGroup/77705950e6ab6808
```

TASK 6: TEST THE DISTRIBUTED APPLICATION

> Document your observations. Include reports and graphs of the load testing tool and the AWS console monitoring output.

- Edwin :

```
❯ echo "GET http://gre-haffner-loadbalancer-473360745.us-east-1.elb.amazonaws.com/wp-admin/" | vegeta attack -
duration=60s | tee results.bin | vegeta report
Requests      [total, rate, throughput]         3000, 50.02, 12.55
Duration      [total, attack, wait]             1m1s, 59.981s, 1.2s
Latencies     [min, mean, 50, 90, 95, 99, max]  96.444ms, 6.907s, 308.117ms, 30.001s, 30.001s, 30.001s, 30.002s
Bytes In      [total, mean]                     8995401, 2998.47
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           25.60%
Status Codes  [code:count]                      0:579  200:768  500:1399  504:254
Error Set:
500 Internal Server Error
Get "http://GrE-Haffner-LoadBalancer-473360745.us-east-1.elb.amazonaws.com/wp-login.php?
redirect_to=http%3A%2F%2Fgre-haffner-loadbalancer-473360745.us-east-1.elb.amazonaws.com%2Fwp-admin%2F&reauth=1":
context deadline exceeded (Client.Timeout exceeded while awaiting headers)
504 Gateway Time-out
Get "http://gre-haffner-loadbalancer-473360745.us-east-1.elb.amazonaws.com/wp-admin/": context deadline exceeded
(Client.Timeout exceeded while awaiting headers)
❯ echo "GET http://gre-haffner-loadbalancer-473360745.us-east-1.elb.amazonaws.com/wp-admin/" | vegeta attack -
duration=60s | tee results2.bin | vegeta report
Requests      [total, rate, throughput]         3000, 50.02, 15.13
Duration      [total, attack, wait]             1m0s, 59.98s, 375.905ms
Latencies     [min, mean, 50, 90, 95, 99, max]  96.105ms, 7.587s, 317.307ms, 30.001s, 30.001s, 30.001s, 30.005s
Bytes In      [total, mean]                     9019066, 3006.36
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           30.43%
Status Codes  [code:count]                      0:589  200:913  500:1105  504:393
Error Set:
500 Internal Server Error
Get "http://GrE-Haffner-LoadBalancer-473360745.us-east-1.elb.amazonaws.com/wp-login.php?
redirect_to=http%3A%2F%2Fgre-haffner-loadbalancer-473360745.us-east-1.elb.amazonaws.com%2Fwp-admin%2F&reauth=1":
context deadline exceeded (Client.Timeout exceeded while awaiting headers)
Get "http://gre-haffner-loadbalancer-473360745.us-east-1.elb.amazonaws.com/wp-admin/": context deadline exceeded
(Client.Timeout exceeded while awaiting headers)
504 Gateway Time-out
```

The server 2 crashes after 5 ish seconds...

As you can see we also have lots of errors 500 and lots of time-out which makes sense since one of the two instance crashed...

Testing with one request per second -> no issue Testing with 10 requests per second -> no issue nslookup result :

```
❯ nslookup GrE-Haffner-LoadBalancer-473360745.us-east-1.elb.amazonaws.com
Server:         10.193.64.16
Address:        10.193.64.16#53

Non-authoritative answer:
Name:   GrE-Haffner-LoadBalancer-473360745.us-east-1.elb.amazonaws.com
Address: 18.208.17.93
Name:   GrE-Haffner-LoadBalancer-473360745.us-east-1.elb.amazonaws.com
Address: 54.88.211.232
```

Testing with 25 requests per second -> no issue

```
Server:         10.193.64.16
Address:        10.193.64.16#53

Non-authoritative answer:
Name:   GrE-Haffner-LoadBalancer-473360745.us-east-1.elb.amazonaws.com
Address: 54.88.211.232
Name:   GrE-Haffner-LoadBalancer-473360745.us-east-1.elb.amazonaws.com
Address: 18.208.17.93
```

Testing with 40 requests per second -> no issue

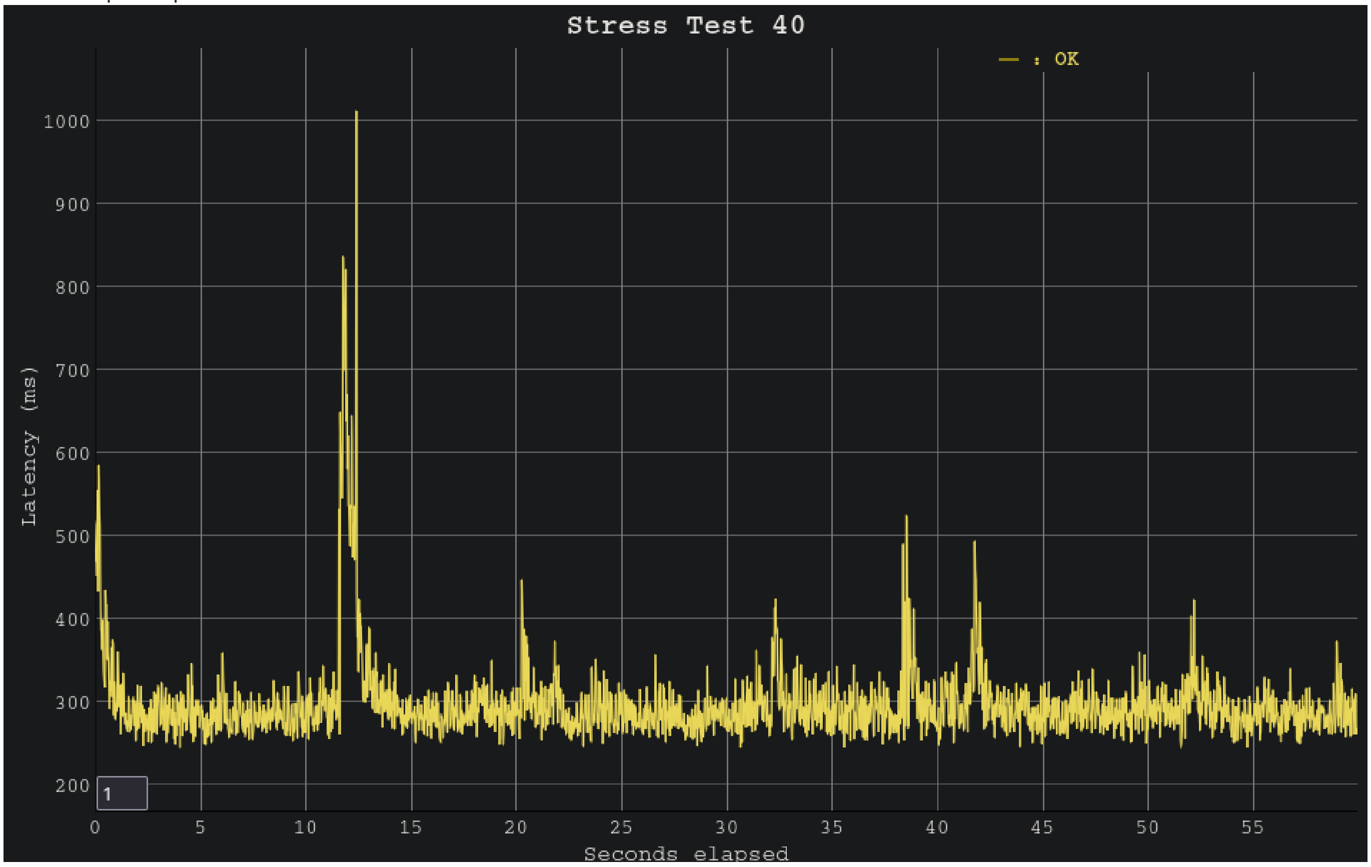
```
❯ nslookup GrE-Haffner-LoadBalancer-473360745.us-east-1.elb.amazonaws.com
Server:         10.193.64.16
Address:        10.193.64.16#53

Non-authoritative answer:
Name:   GrE-Haffner-LoadBalancer-473360745.us-east-1.elb.amazonaws.com
Address: 18.208.17.93
Name:   GrE-Haffner-LoadBalancer-473360745.us-east-1.elb.amazonaws.com
Address: 54.88.211.232
```

Testing with 48 requests per second -> 65.03% ratio, one session crashed :-(

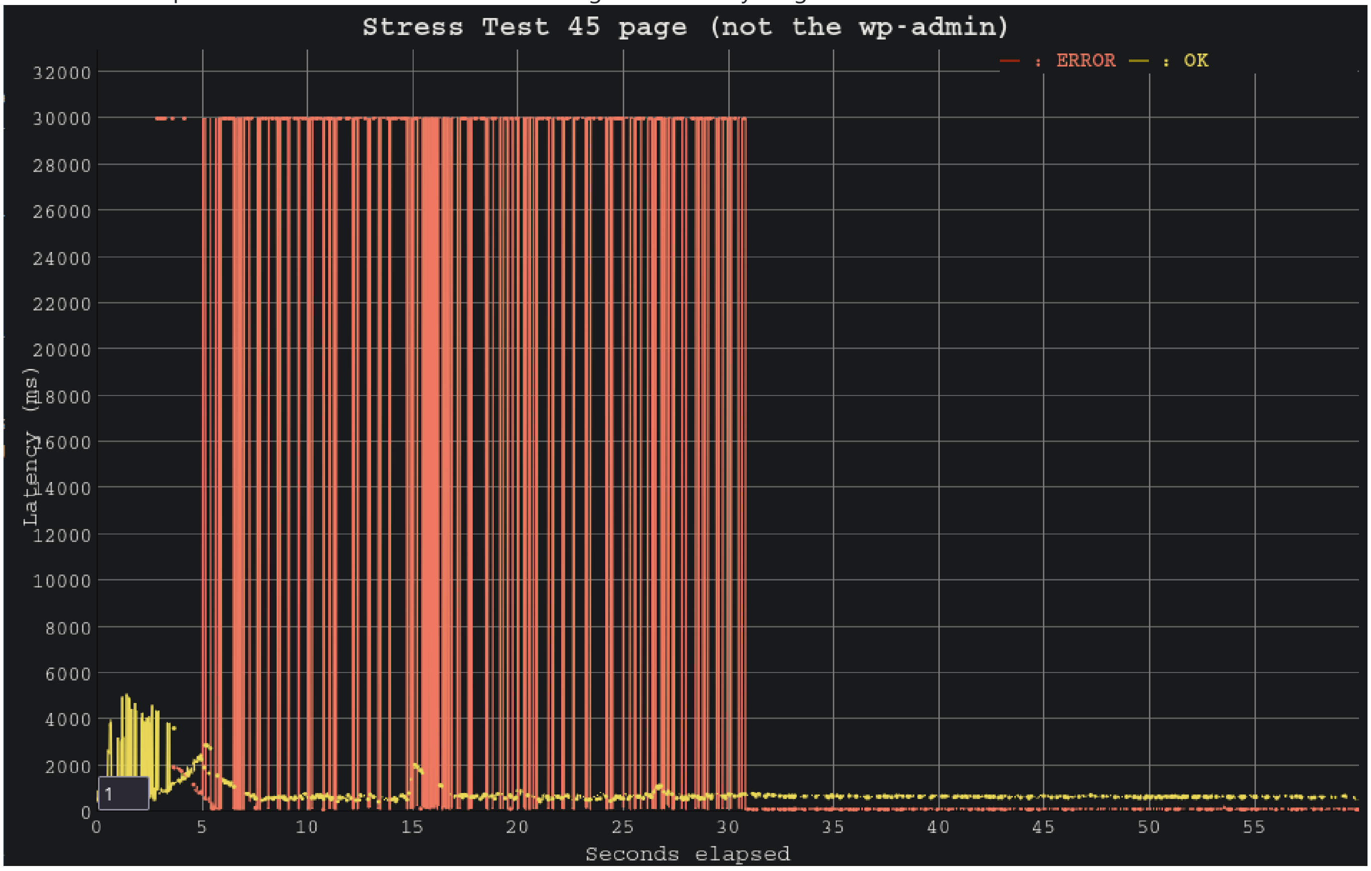
I also tried with 45 and it crashed.... So it seems that 40 is stable, I could thinker around it to find the exact values, but everytime it crashes, it takes a while to restart the instances and I don't think it's worth it to find the exact value.

Graphs

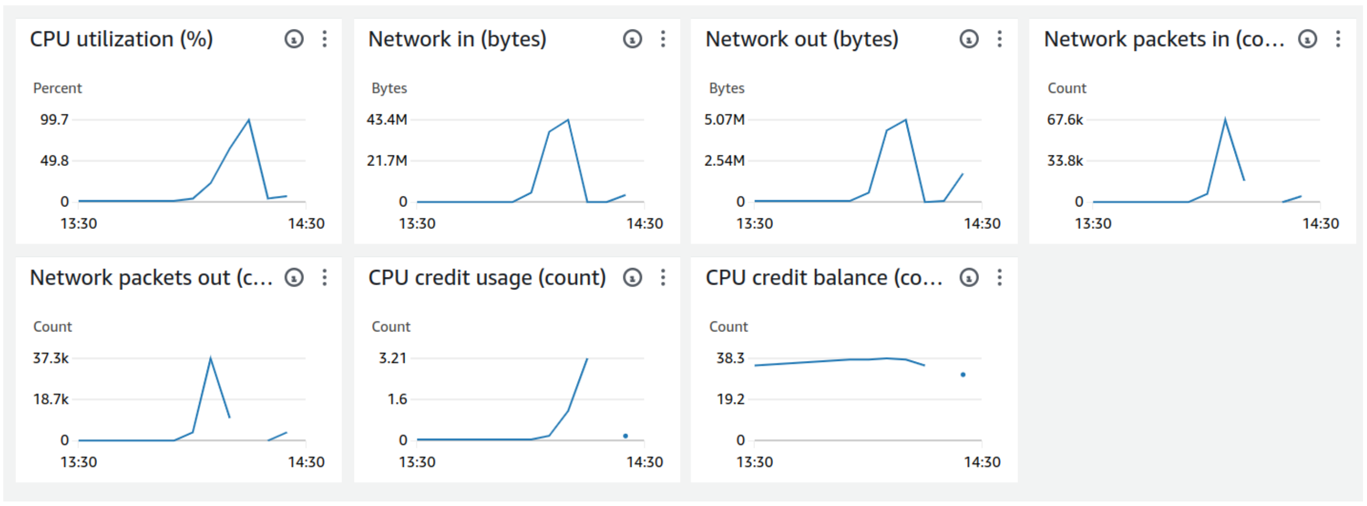
For 40 requests per second



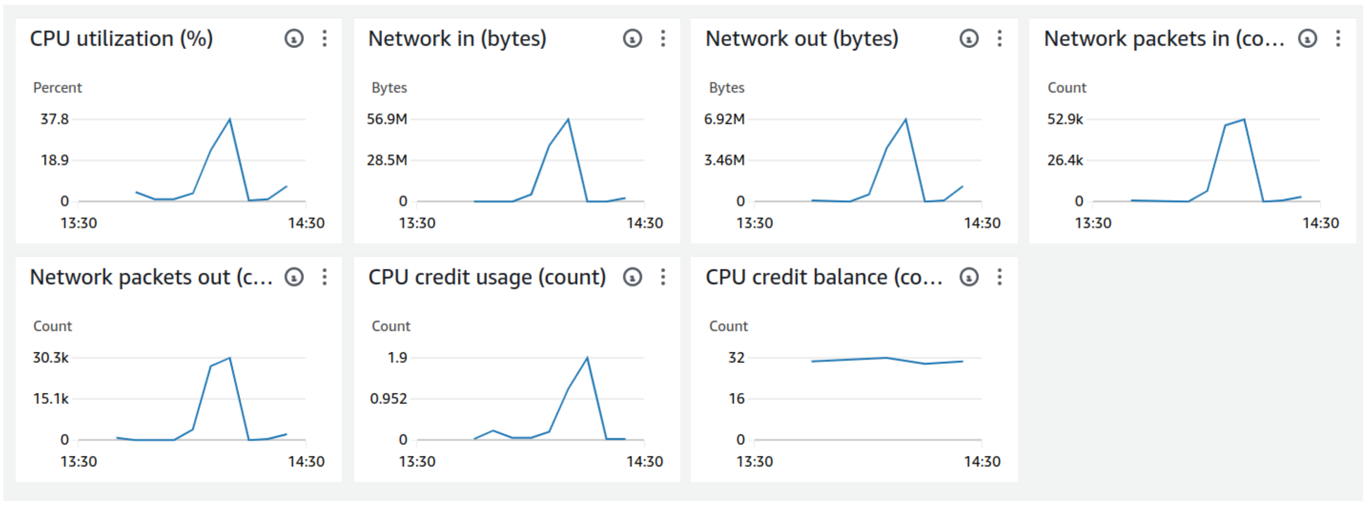
For 45 requests per second, we see that it's absolutely not doing well... We also tried doing the "attack" on a normal page instead of the wp-admin one but it didn't seem to change much of anything



CPU load instance 1 ->



CPU load instance 2 ->



- Arthur :

By launching an attack with default settings vegeta we get:

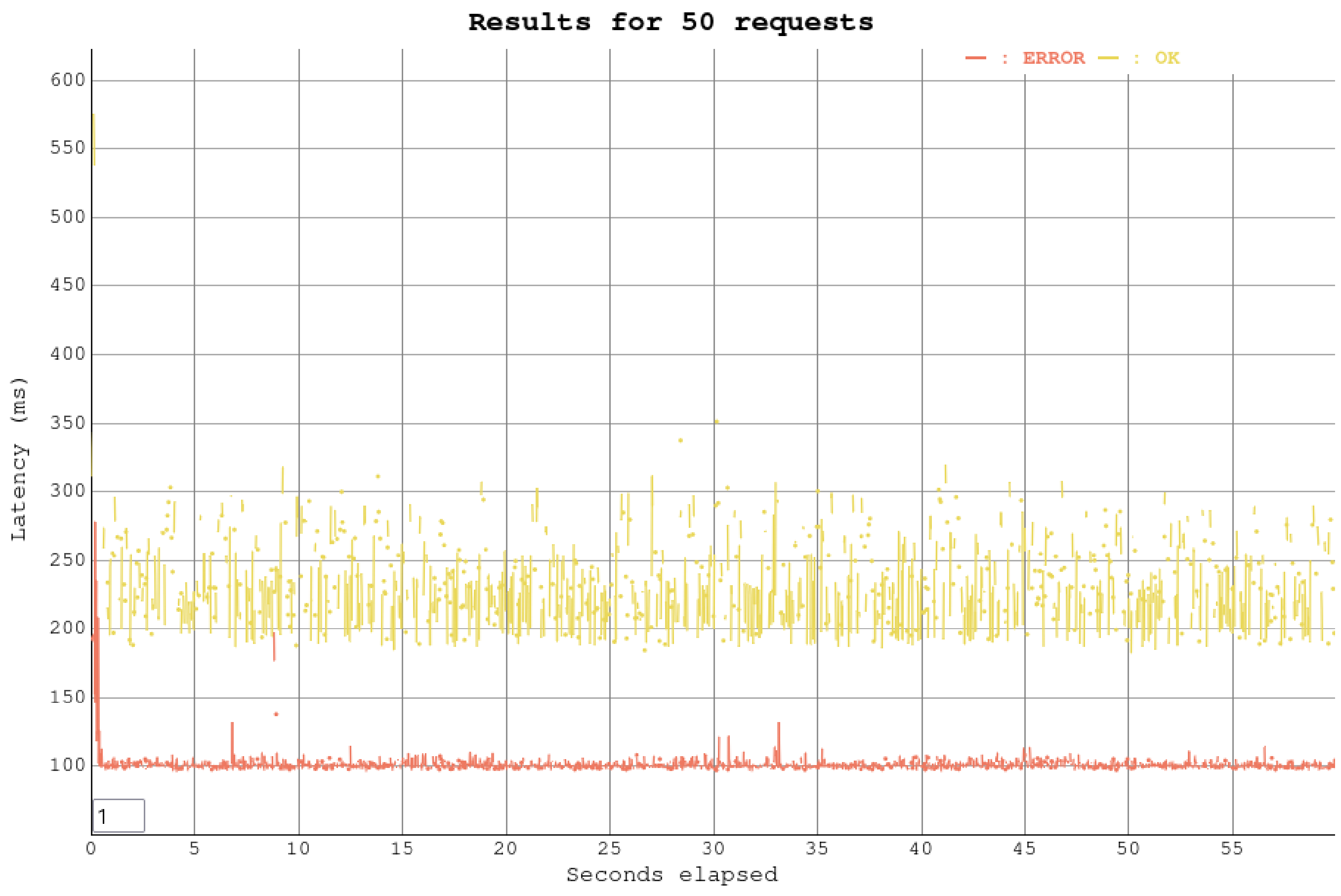
```
echo "GET http://gre-junod-loadbalancer-284381652.us-east-1.elb.amazonaws.com/index.php/les-vaches-cest-superbes/"
| vegeta attack -duration=60s | tee results50r.bin | vegeta report
Requests      [total, rate, throughput]         3000, 50.02, 20.90
Duration      [total, attack, wait]             1m0s, 59.981s, 156.752ms
Latencies     [min, mean, 50, 90, 95, 99, max]  97.844ms, 157.602ms, 103.912ms, 255.684ms, 274.545ms, 297.665ms,
575.482ms
Bytes In      [total, mean]                     92207268, 30735.76
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           41.90%
Status Codes  [code:count]                      200:1257  500:1743
Error Set:
500 Internal Server Error
```

We have HTTP 500 error codes >:(

We can see those in the console of the surviving instance :

```
172.31.89.92 - - [26/Mar/2024:14:38:30 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 500 2808 "-" "Go-
http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:38:28 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:38:28 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.26.116 - - [26/Mar/2024:14:38:28 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:38:28 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.26.116 - - [26/Mar/2024:14:38:28 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:38:29 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.26.116 - - [26/Mar/2024:14:38:28 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:38:30 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 500 2808 "-" "Go-
http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:38:30 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 500 2808 "-" "Go-
http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:38:30 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 500 2808 "-" "Go-
http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:38:30 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 500 2808 "-" "Go-
http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:38:30 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 500 2808 "-" "Go-
http-client/1.1"
```

Graph for 50 requests :



I tried again and again lowering the number of request till i got a success ratio of 100%.

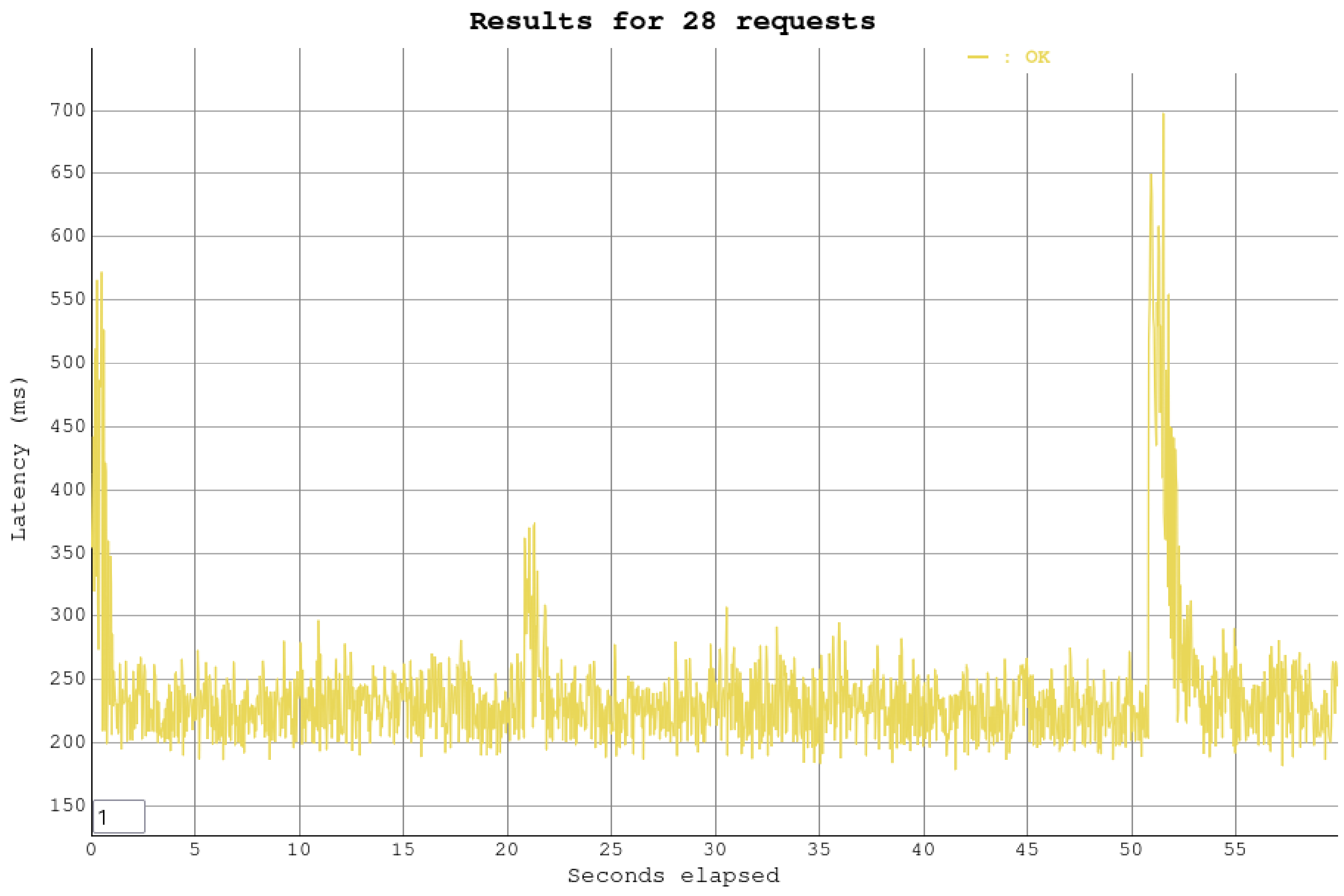
With 28 requests I reached that RATIO :

```
[ajun@fedora CLD]$ echo "GET http://gre-junod-loadbalancer-284381652.us-east-1.elb.amazonaws.com/index.php/les-
vaches-cest-superbes/" | vegeta attack -duration=60s -rate 28 | tee results28r.bin | vegeta report
Requests      [total, rate, throughput]         1680, 28.02, 27.90
Duration      [total, attack, wait]             1m0s, 59.964s, 244.666ms
Latencies     [min, mean, 50, 90, 95, 99, max]  178.617ms, 239.268ms, 231.35ms, 265.259ms, 289.942ms, 522.271ms,
697.71ms
Bytes In      [total, mean]                     117349680, 69851.00
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           100.00%
Status Codes  [code:count]                      200:1680
Error Set:
```

And only HTTP 200 code on the console :

```
172.31.89.92 - - [26/Mar/2024:14:37:34 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.26.116 - - [26/Mar/2024:14:37:34 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.26.116 - - [26/Mar/2024:14:37:34 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:37:34 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.26.116 - - [26/Mar/2024:14:37:34 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:37:34 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:37:34 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.89.92 - - [26/Mar/2024:14:37:34 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.26.116 - - [26/Mar/2024:14:37:34 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
172.31.26.116 - - [26/Mar/2024:14:37:34 +0000] "GET /index.php/les-vaches-cest-superbes/ HTTP/1.1" 200 13674 "-"
"Go-http-client/1.1"
```

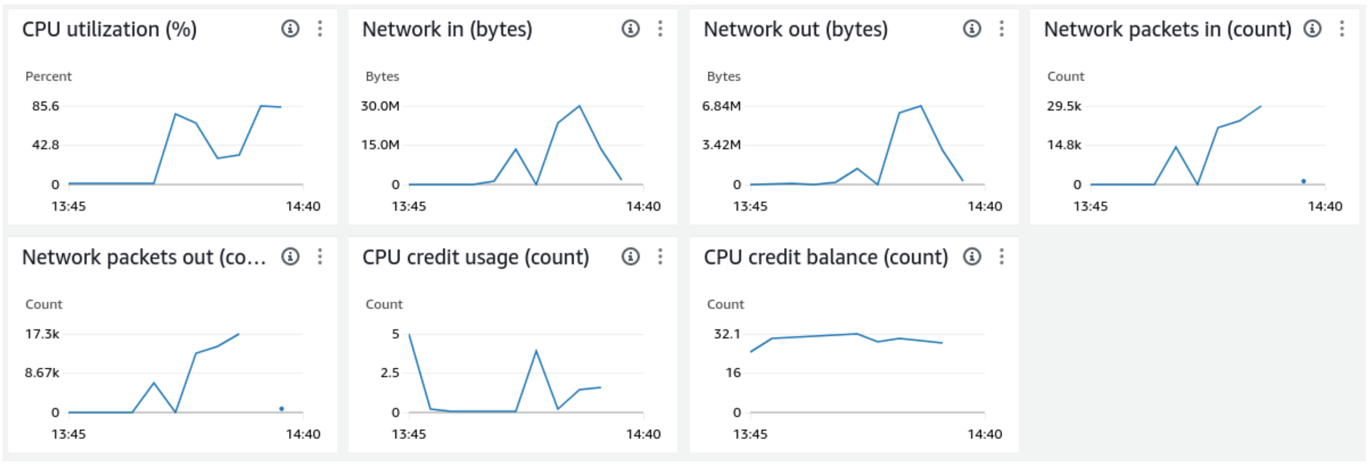
Graph for 28 requests :

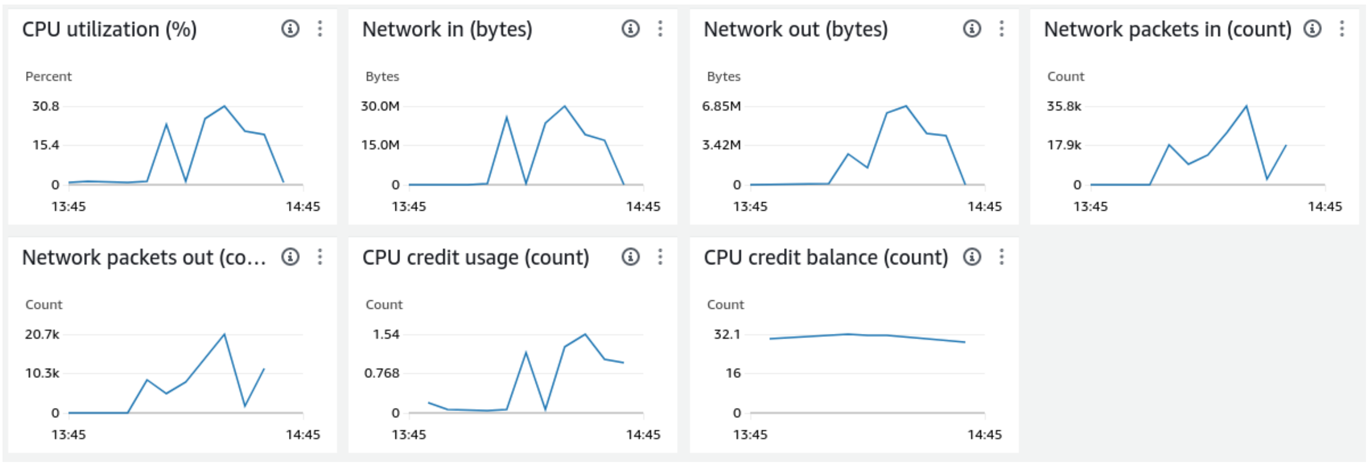


So now we know that the limit at which our instances crash or freeze is 29 requests:

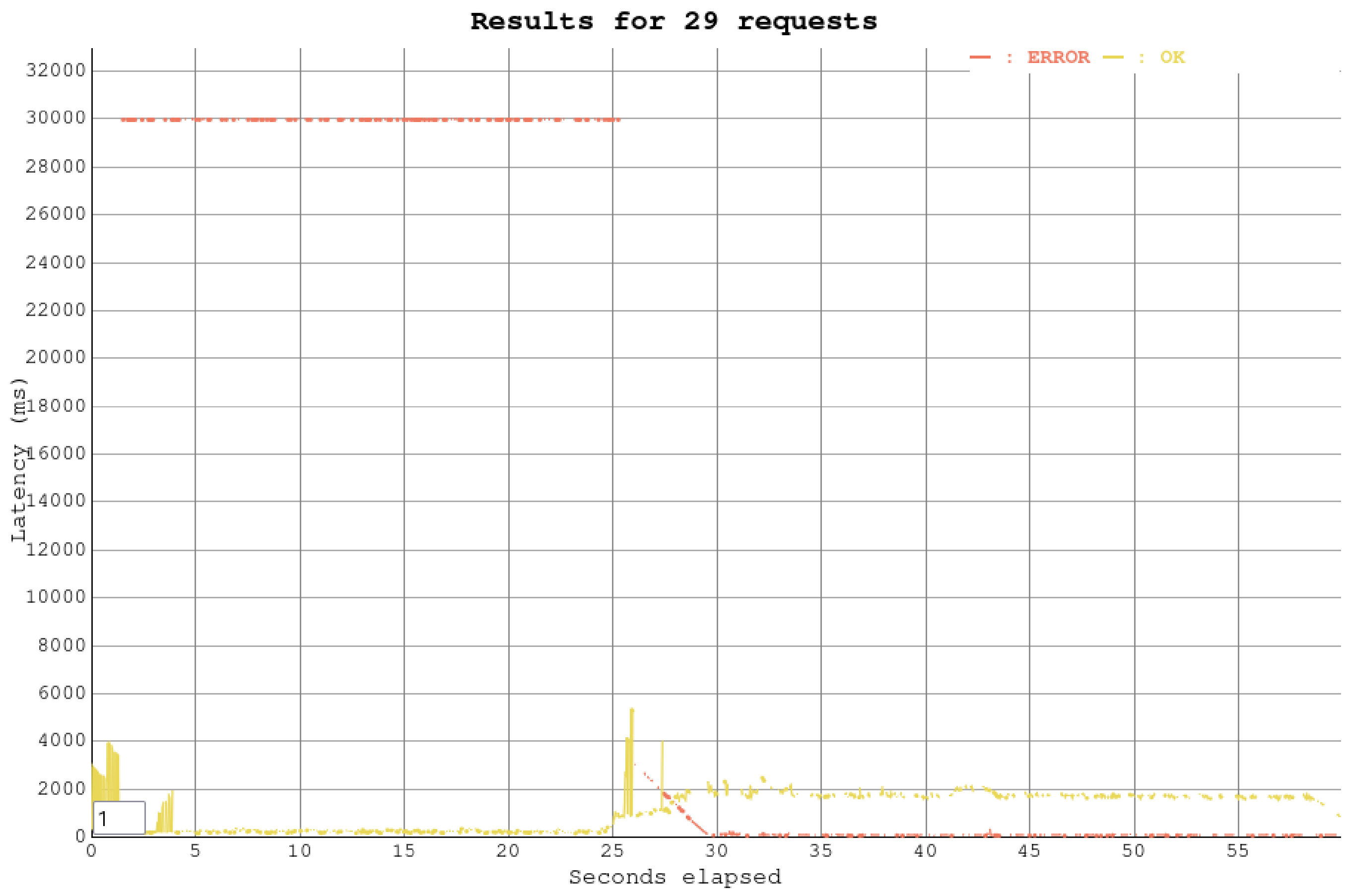
```
[ajun@fedora CLD]$ echo "GET http://gre-junod-loadbalancer-284381652.us-east-1.elb.amazonaws.com/index.php/les-
vaches-cest-superbes/" | vegeta attack -duration=60s -rate 29 | tee results29r.bin | vegeta report
Requests      [total, rate, throughput]         1740, 29.02, 15.08
Duration      [total, attack, wait]             1m1s, 59.965s, 840.052ms
Latencies     [min, mean, 50, 90, 95, 99, max]  97.915ms, 6.393s, 1.113s, 30.001s, 30.001s, 30.001s, 30.002s
Bytes In      [total, mean]                     65296651, 37526.81
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           52.70%
Status Codes  [code:count]                      0:331  200:917  500:492
Error Set:
500 Internal Server Error
Get "http://gre-junod-loadbalancer-284381652.us-east-1.elb.amazonaws.com/index.php/les-vaches-cest-superbes/":
context deadline exceeded (Client.Timeout exceeded while awaiting headers)
```

Monitoring images for the 2 instances after 29 requests :





Graph for 29 requests :



> When you resolve the DNS name of the load balancer into IP addresses what do you see? Explain.

`nslookup` after the 29 requests per seconds :

```
[ajun@fedora CLD]$ nslookup GrE-Junod-LoadBalancer-284381652.us-east-1.elb.amazonaws.com
Server:         127.0.0.53
Address:    127.0.0.53#53

Non-authoritative answer:
Name:    GrE-Junod-LoadBalancer-284381652.us-east-1.elb.amazonaws.com
Address: 54.91.131.146
Name:    GrE-Junod-LoadBalancer-284381652.us-east-1.elb.amazonaws.com
Address: 18.233.227.68
```

We don't see any meaningful differences from the `nslookup` we did earlier in the lab. This may be because we didn't set up any scaling options for our load balancer, so if we have a lot of requests, it cannot do more than try to distribute the load between the two given instances to the best of its ability... (no vertical or horizontal scaling).

> Did this test really test the load balancing mechanism? What are the limitations of this simple test? What would be necessary to do realistic testing?

It was quite hard to really observe the load balancing more than "Yeah, we see a spike in the CPU utilisation in the two instances". So yes, the load balancing is indeed working but we can't see more than that. Which machine is more utilized? Exactly how many requests were sent to each machine and which machine was more prone to errors was harder to quantify. So we don't really think this test is necessary enough to do realistic testing. We should probably use a software that is meant to test the load balancing instead of doing what we just did.