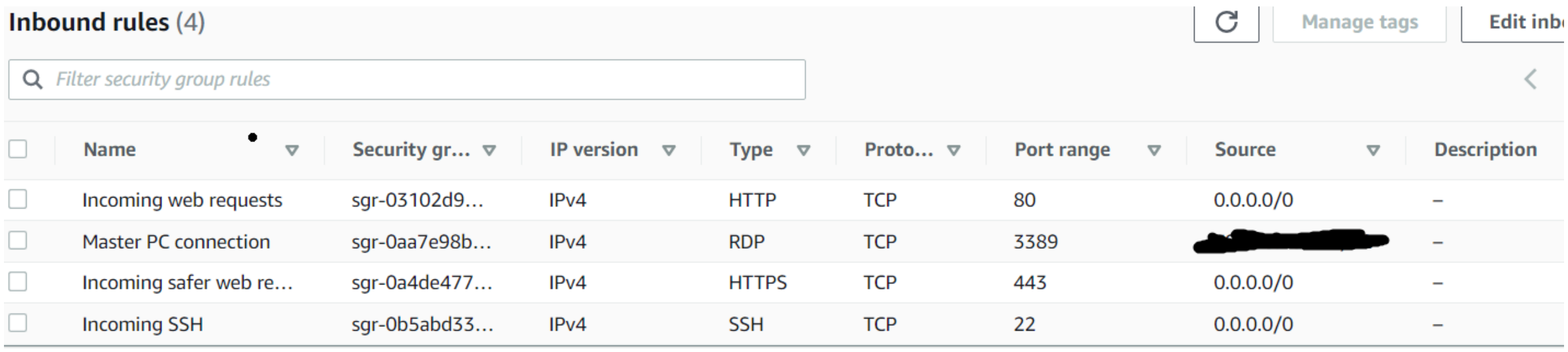
DV1566 - Labb 2

Group	17
Student 1	Isac Svensson
Student 2	Robin Stenius

- **The configuration of your security group and VPC**



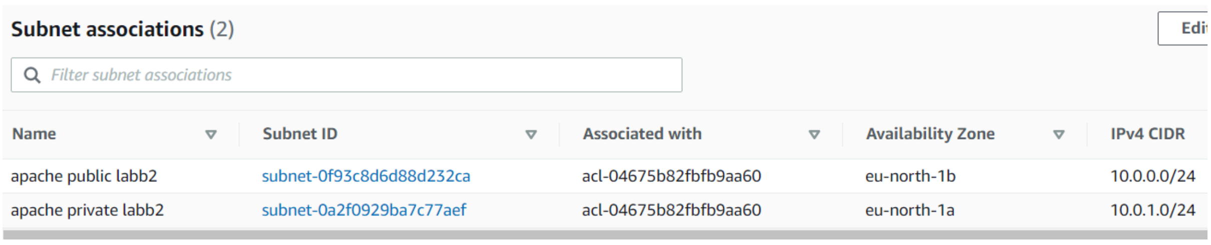
Name	Security gr...	IP version	Type	Proto...	Port range	Source	Description
Incoming web requests	sgr-03102d9...	IPv4	HTTP	TCP	80	0.0.0.0/0	–
Master PC connection	sgr-0aa7e98b...	IPv4	RDP	TCP	3389	[redacted]	–
Incoming safer web re...	sgr-0a4de477...	IPv4	HTTPS	TCP	443	0.0.0.0/0	–
Incoming SSH	sgr-0b5abd33...	IPv4	SSH	TCP	22	0.0.0.0/0	–

(pic 1. Security group)

These are the configurations for our security group, we followed the Amazon documentation for security group rules for different use cases together with the various set-up guides by Amazon in order to complete this laboratory[1].

The four rules used are simply for letting our computer reach the VM through remote access and enabling HTTP/HTTPS-traffic. When looking back we realized that HTTPS-access was not necessary.

[1] https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/security-group-rules-reference.html



Subnet associations (2)

Filter subnet associations

Edi

Name	Subnet ID	Associated with	Availability Zone	IPv4 CIDR
apache public labb2	subnet-0f93c8d6d88d232ca	acl-04675b82fbfb9aa60	eu-north-1b	10.0.0.0/24
apache private labb2	subnet-0a2f0929ba7c77aef	acl-04675b82fbfb9aa60	eu-north-1a	10.0.1.0/24

(pic 2. VPC Subnet associations)

For our non-default VPC, we created subnets in order to successfully have a private subnet that our apache server and client instances could use, and a public subnet for our home computers to connect with.

- **The results of performance test (Task 2)**
    - **The performance metrics**

We chose the following metric for CPU benchmarking:

- CPU-Utilization

We chose this because it shows the percentage utilization of the CPU, and with this we can see if it reaches potential bottlenecks under certain running conditions. For example, if the CPU reaches above 70% it would be considered a bottleneck.

We also chose to monitor the VolumeQueueLength, VolumeIdleTime, VolumeWriteBytes and VolumeReadBytes in order to see if the average queue length exceeds 2.0, which would indicate a possible bottleneck in the disk. The read and write bytes can be used in combination with the idle time to monitor the disk utilization contrary to the reading/writing demands.

- **How you generated the load**

We generated the load by writing a script looping sysbench with 1 - 128 threads creating 1,000,000 prime numbers.

./cpuscript1

```
for i in  1 2 3 4 5 6 7 8 9 10;
do
        for threads in 1 2 4 8 16 32 64 128;
        do
                sysbench --test=cpu --cpu-max-prime=1000000 --num-threads=$threads run
        done
done
```

We used this script because it was simple to use and gave some interesting metrics. The sysbench test is run with an increasing number of threads, and the entire test is run 10 times.

./cpuscript2

```
for threads in  1 2 4 8 16 32 64 128;
do
        for i in 1 2 3 4 5 6 7 8 9 10;
        do
                sysbench --test=cpu --cpu-max-prime=1000000 --num-threads=$threads run
        done
done
```

This is basically the same script, but reversed the loops, meaning instead of increasing the number of threads per testrun we instead run each number of threads 10 times in a row before increasing the number of threads. This gives a different visual result from cloudwatch.

./diskscript

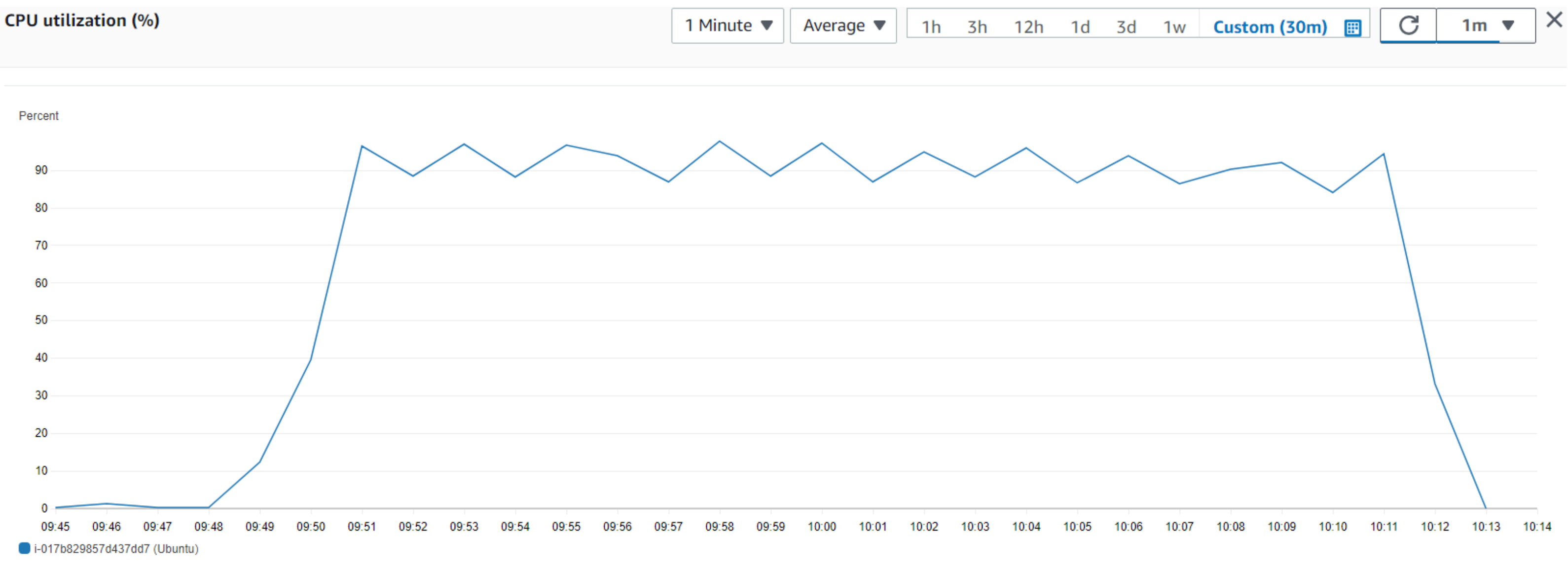
```
sysbench --test=fileio --file-total-size=5G --file-num=3 prepare

for threads in 1 4 8 16 32;
do
      sysbench --test=fileio --file-total-size=5GB --file-test-mode=rndwr --max-time=240
--max-requests=0 --file-block-size=4K --file-num=3 --num-threads=$threads run;
done

rm test_file.*
```

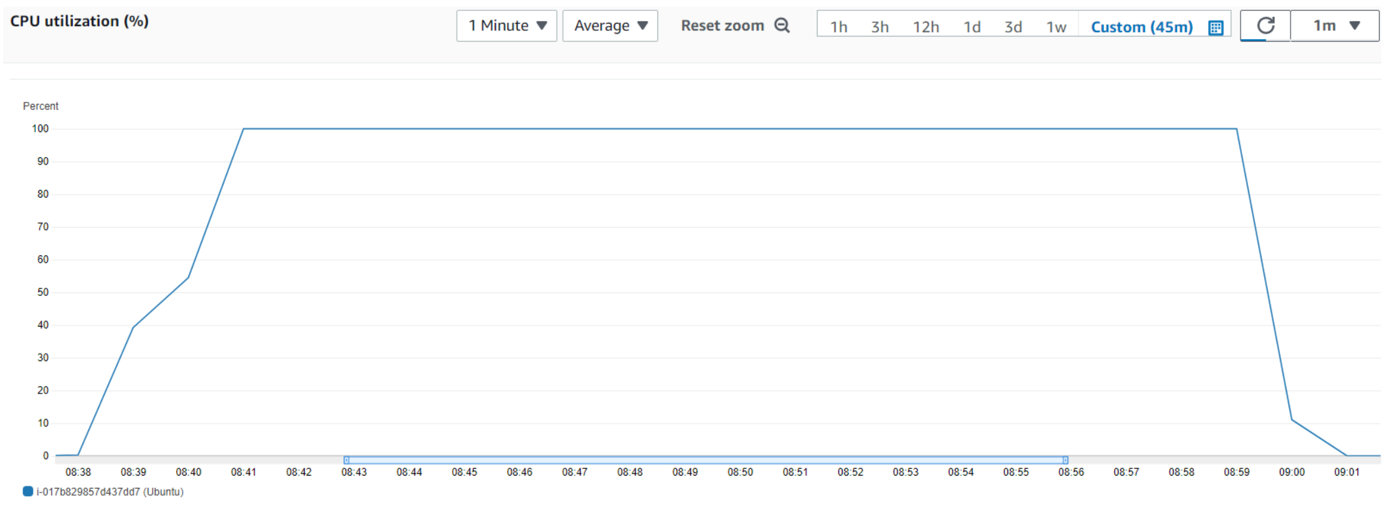
We chose this script to stress test the disk, it first creates the 3 files needed and then uses a loop to test with various numbers of threads ranging from 1 to 32 and afterwards removes the test files. This test ensures that both the reading and writing capabilities of the disk are tested.

- **The meaning of your results**



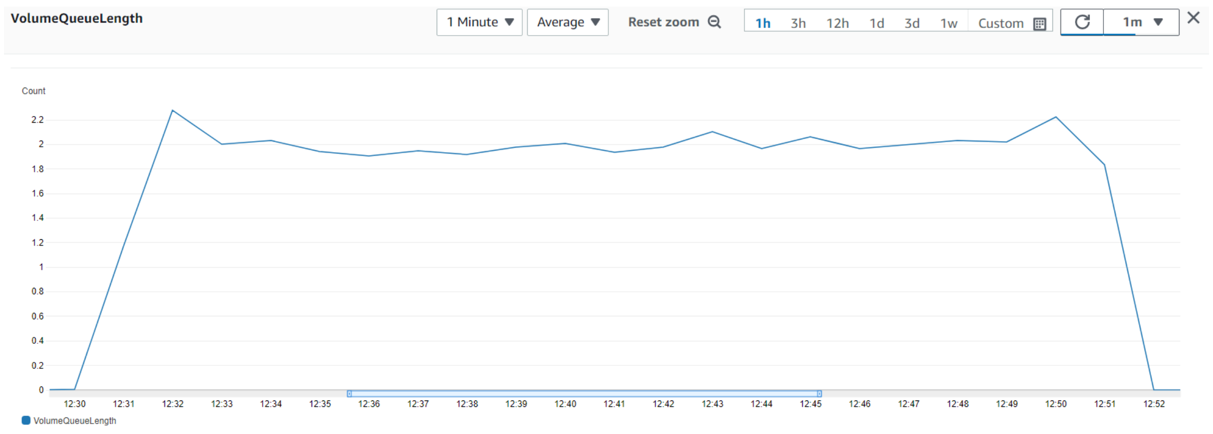
(pic 3. cpuscript1 metrics from cloudwatch)

The meaning of these results shown in picture 3 is to show the CPU utilization from our stress testing with *cpuscript1*, we can see that there are 10 peaks showing the maximum load of the script we ran. When running sysbench on fewer threads the CPU-utilization is lower and when using many threads the CPU-utilization is higher.



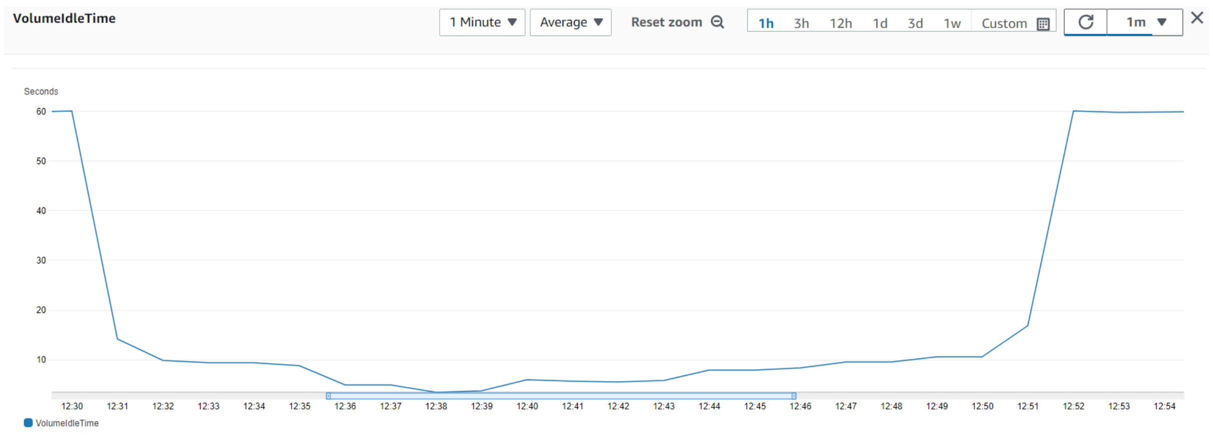
(pic 4. cpuscript2 metrics from cloudwatch)

If we look at picture 4 that shows the performance of the CPU while running *cpuscript2*. This shows that the CPU-utilization quickly reaches the maximum capacity when running on multiple threads.



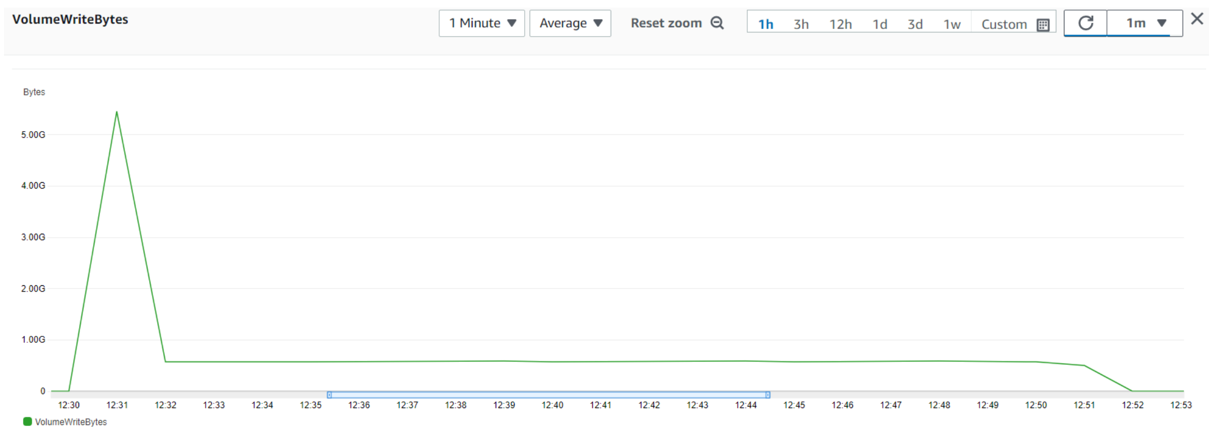
(pic 5. Average queue length from disk stress test)

Picture 5 shows the average queue length from our **diskscript**, this metric shows us that the disk is hovering around the theoretical value of 2.0 to be considered a potential bottleneck. However, since it never goes very far beyond this, it’s possible the instance has a set cap on the number of requests to have in the queue at any given moment.



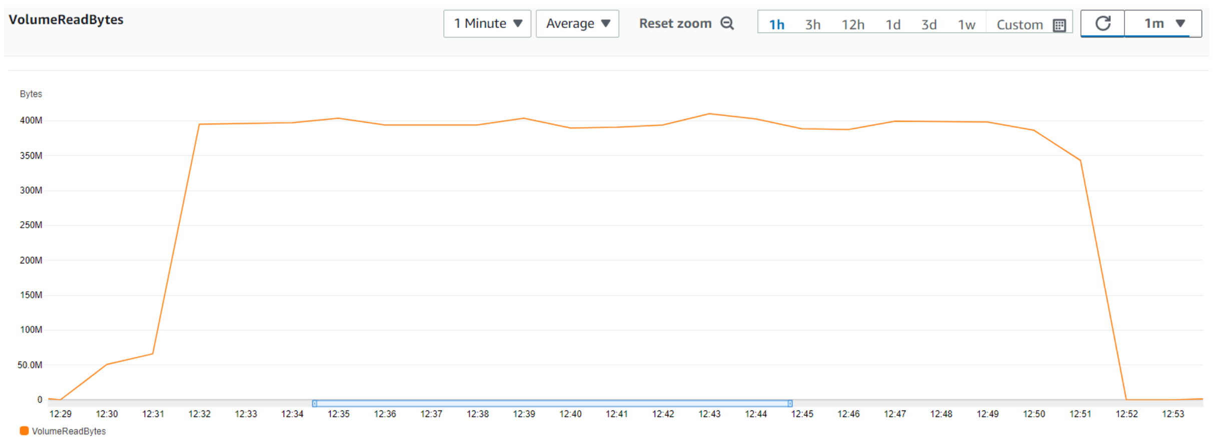
(pic 6. Average idle time from disk stress test)

The average idle time metric tells us that the disk was busy a significant amount during the stress test, only having a couple of seconds for idling which means it would have been very hard for any other process to get access to it.



(pic 7. Average bytes written from disk stress test)

This metric shows that the 3 files of total 5GB are created in the beginning and then they are steadily overwritten with about 600 MB/min.



(pic 8. Average bytes read from disk stress test)

The average bytes read from the stress test shows hovering around 400M, which shows us that it was working in a relatively consistent manner.

- **The results of performance test (Task 3)**
    - **The performance metrics**

The metrics we chose to observe from these tests were the NetworkOut and the ActiveConnectionCount metrics in order to see the number of bytes per second sent from the server and the number of active connections respectively.
However, even though according to the documentation the ActiveConnectionCount is the

"*The total number of concurrent TCP connections active from clients to the load balancer and from the load balancer to targets.*"[2].

We did not get any data from cloudwatch for this metric.

- **How you generated the load**

When we first made a script for this, we had an increasing number of users up to 10,000. But when the number of users exceeded well beyond 1000 we got an error "**socket: Too many open files (24)**". From further trial and error we found the cap to be at 1020 users, so we adjusted our script accordingly.

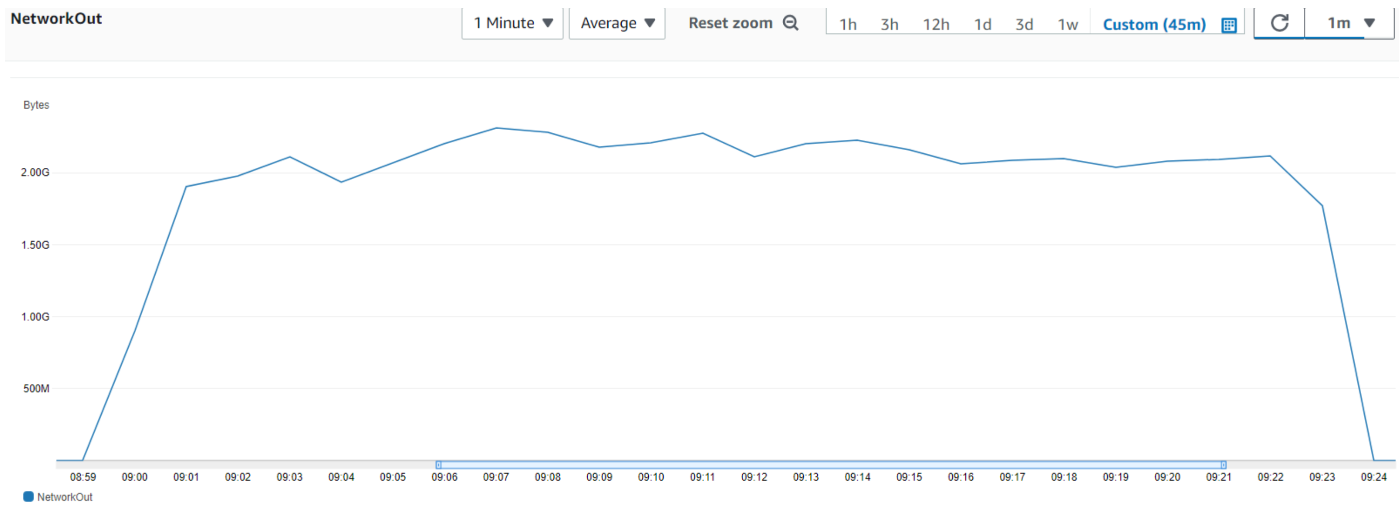
./webscript (on a Ubuntu instance acting as client)

```
for users in 10 50 100 250 500 1020;
do
    for reqs in 10000 25000 50000 100000 250000 500000 1000000;
    do
        ab -n $reqs -c $users http://<private-ip>
    done
done
```

In the final script we wanted to try and test both ends of the spectrum, with low numbers of users/requests but also high number users/requests. Perhaps it would have been possible to go even higher with the number of requests but since the metrics from cloudwatch were hovering around 2.0-2.3G we believe it would not have given any significant differences.

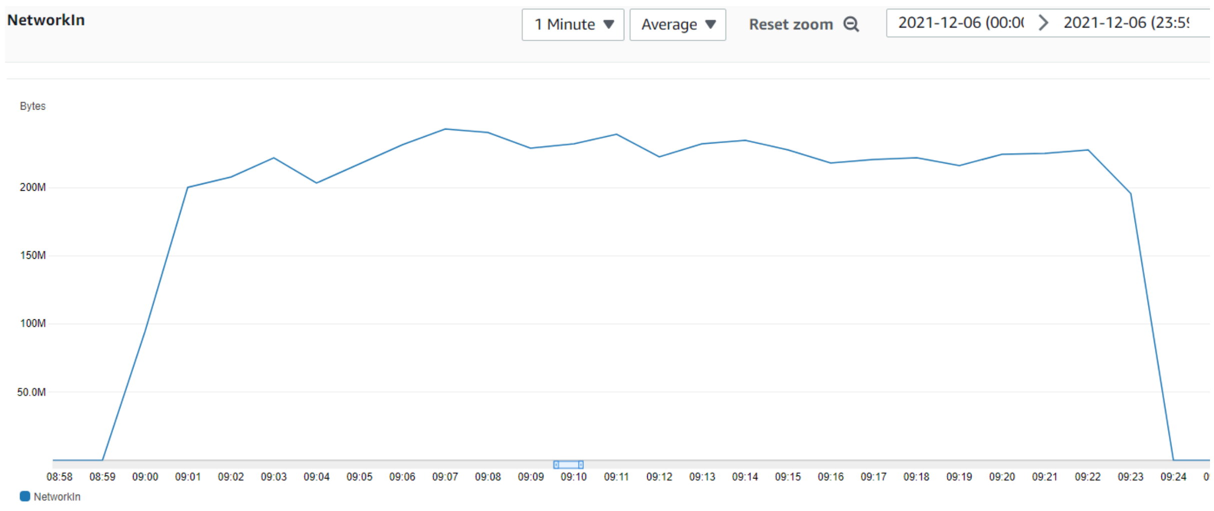
[2] https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-cloudwatch-metrics.html

- ## The meaning of your results



(pic 9. NetworkOut metric on the apache server)

The NetworkOut metric shows us how much data the apache server is sending outwards.



(pic 10. NetworkIn metric on the apache server)

Together with the NetworkIn metric, which shows how much average data is flowing into the server, you can compare the in/out data flow of your web-server.