

GROUP 11

EMMANUEL VAN BERLO – 199507107978 – emvn21@student.bth.se

PARTHASARATHY SINGH SAMANTA - 20000207-T215 - pasb21@student.bth.se

PROJECT ASSIGNMENT: IMPLEMENTATION AND DEMONSTRATION OF LOAD BALANCER FOR HTTP REQUESTS IN MS AZURE USING VIRTUALIZATION TECHNIQUES.

TASK/ AIM

We are to implement, demonstrate and test an adaptive web service that applies a load balance using Docker, NGINX and Python in the MS Azure cloud with our focus on Docker MS Azure, Overlay and Adaptation.

PROJECT ENVIRONMENT

We used Ubuntu and MS Azure cloud services

PROCESS

Virtual machines 31,32 and 33 were assigned to our group for the group project and with that we considered the virtual machine 31 as our load balancer while the remaining two servers were the web servers. The flask module was used to configure the web servers and load balancer server.

In creating and running of the Docker container we used this command “\$ docker run --name xxx -p 8080:4000 xxx”.

CONFIGURING THE WEB SERVERS

```
# Use an official Python runtime as a parent image
FROM python:3-slim
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org Flask
RUN pip install psutil
ENV NAME Server1
CMD ["python", "server1.py"]
~
~
~
~
~
~
~
~
~
~
```

Dockerfile

```

from flask import Flask
import os, psutil
import socket

app = Flask(__name__)
@app.route("/")
def hello():
    html = "<h3>You are logged into {name}!</h3> <b>Hostname:</b> {hostname} <br/> <b>Load on {name}:</b> {loadpercent}"
    return html.format(name=os.getenv("NAME", "Server1"), hostname=socket.gethostname(), loadpercent = load() )

@app.route("/load")
def load():
    load = psutil.cpu_percent()
    return str(load)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=4000)

```

Server Code written in Python.

Checking of Web servers:

Using this command “sudo docker build -t name” to build an image and “sudo docker run -p 8080:4000 name” to run the docker.



You are logged into Server1!

Hostname: 7feea7bb1923

Load on Server1: 0.0

Load on Server 1



⚠ Not secure | 20.166.88.40:8080



Gmail



YouTube



Maps



BTH - Canvas log-in



Movies

You are logged into Server2!

Hostname: 95c878e3e3b4

Load on Server2: 0.5

Load on Server 2

ROUND ROBIN

Round Robin load balancing is a technique where incoming network traffic is distributed evenly across multiple servers. Each server is given a turn to handle a request, in a cyclic order, hence the name "Round Robin". This method helps to distribute the workload evenly among servers and prevent any one server from becoming a bottleneck. It's a simple and easy to implement algorithm that distributes the traffic in a fair way.

This is how we executed our Round Robin;

```
# Use an official Python runtime as a parent image
FROM python:3-slim
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org Flask
RUN pip install requests
ENV NAME RoundRobin
CMD ["python", "rr.py"]
~
~
~
~
~
~
~
~
```

Dockerfile

```
from flask import Flask, request, send_file, session, url_for, redirect
import requests
import os
import socket
import sys

app = Flask(__name__)

rr=0

ipaddr1 = "20.238.124.104"
ipaddr2 = "20.166.88.40"

@app.route('/')
def func():
    global rr
    if rr==0:
        response = requests.get("http://{}:8080".format(ipaddr1))
        rr=1
        return str(response.content)

    elif rr==1:
        response = requests.get("http://{}:8080".format(ipaddr2))
        rr=0
        return str(response.content)

    else:
        return "Error: Server Unavailable", 404

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=4000)
~
~
~
~
```

Rounded Robin python code

Rounded Robin Algorithm results using “/rr”:



⚠ Not secure | 40.69.73.130:8080



Gmail



YouTube



Maps



BTH - Canvas log-in



Movies



Books

You are being redirected by RoundRobin, Stay Tuned

Hostname: 6aa5d7aa9a93



⚠ Not secure | 40.69.73.130:8080/rr



Gmail



YouTube



Maps



BTH - Canvas log-in



Movies



Books



b'

You are logged into Server2!

Hostname: 95c878e3e3b4

Load on Server2: 0.1

,

b'

You are logged into Server1!

Hostname: 7feea7bb1923

Load on Server1: 0.1

,

WORKLOAD SENSITIVITY

A workload sensitivity load balancer is a load balancing technique that adjusts the distribution of incoming network traffic based on the current workload of the servers. It monitors the resources of each server, such as CPU usage and memory usage, and routes traffic to the server that has the most available resources. This helps to ensure that the workload is distributed evenly among servers and that no one server becomes overwhelmed. In simpler terms, it's a load balancer that can adjust traffic based on the servers' current workload to avoid overloading them.

This is how we executed our workload sensitivity;

```
# Use an official Python runtime as a parent image
FROM python:3-slim
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org Flask
RUN pip install requests
ENV NAME LoadBalancer
CMD ["python", "loadbalancer.py"]
~
~
~
~
~
~
~
```

Dockerfile

```
from flask import Flask, request
import requests
import os
import socket
import sys

app = Flask(__name__)

ipaddr_srv1 = "20.238.124.104"
ipaddr_srv2 = "20.166.88.40"

@app.route("/")
def hello():
    html = "<h3>You are being redirected by {name}, Stay Tuned</h3> <b>Hostname:</b> {hostname} <br/>"
    return html.format(name=os.getenv("NAME", "LoadBalancer"), hostname=socket.gethostname())

@app.route("/1sa")
def load():
    global load_srv1, load_srv2
    result = loadsensitivity()
    return result

def loadsensitivity():
    global load_srv1, load_srv2
    load_srv1 = getload(ipaddr_srv1)
    load_srv2 = getload(ipaddr_srv2)

    if load_srv1 <= load_srv2:
        response = getresponse(ipaddr_srv1)
        print("/n Request Sent to Server 1 /n")
        print("Load of the Server1 is : ", load_srv1)
        return response
    else:
        response = getresponse(ipaddr_srv2)
        print("/n Request Sent to Server 2 /n")
        print("Load of the Server1 is : ", load_srv2)
        return response
    return 1
```






```
def getload(ipaddr):  
    global load_srv  
    load_srv = requests.get("http://{ }:8080/load".format(ipaddr)).content  
    return float(load_srv)  
  
def getresponse(ipaddr):  
    global srv_response  
    srv_response = requests.get("http://{ }:8080/".format(ipaddr)).content  
    return srv_response  
  
if __name__ == "__main__":  
    app.run(host='0.0.0.0', port=4000)
```

Python algorithm for Workload Sensitivity.

Workload Sensitivity Algorithm results using “/load”:



← → ↻ ⚠ Not secure | 40.69.73.130:8080/lsa





 Gmail  YouTube  Maps  BTH - Canvas log-in 

You are logged into Server1!

Hostname: 7feea7bb1923

Load on Server1: 0.0

← → ↻ ⚠ Not secure | 40.69.73.130:8080/lsa

 Gmail  YouTube  Maps  BTH - Canvas log-in 

You are logged into Server2!

Hostname: 95c878e3e3b4

Load on Server2: 0.0

WEIGHTED ROUND ROBIN

Weighted Round Robin is a variation of the Round Robin load balancing algorithm that allows different servers to have different levels of priority. Instead of treating all servers as equal and giving them the same amount of traffic, a weight is assigned to each server, and traffic is distributed to them in proportion to their weight. This allows for more control over how traffic is distributed among servers and can be used to prioritize certain servers or workloads. In simpler terms, it's a Round Robin algorithm where each server is assigned a weight, and traffic is distributed according to the weight they have, allowing more control on how traffic is distributed.

This is how we executed our Weighted Round Robin;

```
# Use an official Python runtime as a parent image
FROM python:3-slim
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org Flask
RUN pip install requests
ENV NAME WeightedRoundRobin
CMD ["python", "w_rr.py"]
```

Dockerfile

```

from flask import Flask
import requests
import os
import socket
import sys
import random

app = Flask(__name__)

w=[0]*2 # These are the change in weight instances
s=[1,2] # These are the servers
count=0 # The iterative count
c =[4,6] # The constraints of weights defined.

ipaddr_srv1 = "20.238.124.104"
ipaddr_srv2 = "20.166.88.40"

@app.route("/")
def hello():
    html = "<h3> You are being redirected by {name}, Stay Tuned </h3> <b> Hostname:</b> {hostname} <br/>"
    return html.format( name=os.getenv("NAME", "WeightedRoundRobin"), hostname=socket.gethostname() )

@app.route('/wrr')
def wrr():
    global random_srv, count, s

    random_srv = random.choice(s)
    result = wrr_alg(random_srv)
    count = count + 1

    if count==10:
        count = 0
    return result

```

With this algorithm we have taken weights values 4 and 6, where we expect that from each batch of 10 requests with server 1 showing 4 times and server 2 showing 6 times.

```

def wrr_alg(random_srv):
    global w

    if random_srv == 1:
        # getserver(random_srv)

        if sum(w) < sum(c):
            if w[0] < c[0]:
                w[0] = w[0] + 1
                response = getresponse(ipaddr_Srv1)
            else:
                # srv_w(random_srv)
                w[0] = 0
                w[1] = w[1] + 1
                response = getresponse(ipaddr_Srv2)
        else:
            # getlimit(random_srv)
            w[0] = 1
            response = getresponse(ipaddr_Srv1)

    elif random_srv == 2:
        # getserver(random_srv)

        if sum(w) < sum(c):
            if w[1] < c[1]:
                w[1] = w[1] + 1
                response = getresponse(ipaddr_Srv2)
            else:
                # srv_w(random_srv)
                w[1] = 0
                w[0] = w[0] + 1
                response = getresponse(ipaddr_Srv1)
        else:
            # getlimit(random_srv)
            w[1] = 1

```

```

        else:
            # getlimit(random_srv)
            w[0] = 1
            response = getresponse(ipaddr_Srv1)
    elif random_srv == 2:
        # getserver(random_srv)

        if sum(w) < sum(c):
            if w[1] < c[1]:
                w[1] = w[1] + 1
                response = getresponse(ipaddr_Srv2)
            else:
                # srv_w(random_srv)
                w[1] = 0
                w[0] = w[0] + 1
                response = getresponse(ipaddr_Srv1)
        else:
            # getlimit(random_srv)
            w[1] = 1
            response = getresponse(ipaddr_Srv2)

    else:
        response = geterror()

    return response

def getresponse(ipaddr):
    global srv_response
    srv_response = requests.get("http://{}:8080/".format(ipaddr)).content
    return srv_response

def geterror():
    html = "<h3> Invalid Choice, Please, Reload </h3>"
    return html

```

```

def geterror():
    html = "<h3> Invalid Choice, Please, Reload </h3>"
    return html

"""
def getserver(val):
    if val == 1:
        html = "<h3> You are being redirected to {name}, Stay Tuned </h3>"
        return html.format( name="Server 1")
    if val == 2:
        html = "<h3> You are being redirected to {name}, Stay Tuned </h3>"
        return html.format( name="Server 2")

def srv_w(val):
    if val == 1:
        html = "<h3> {name1} full, Redirecting request through {name2} </h3>"
        return html.format( name1="Server 1", name2="Server 2" )
    if val == 2:
        html = "<h3> {name1} full, Redirecting request through {name2} </h3>"
        return html.format( name1="Server 2", name2="Server 1" )

def getlimit(val):
    if val == 1:
        html = "<h3> Both servers full, Redirecting request through {name1} </h3>"
        return html.format( name1="Server 1" )
"""

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=4000)

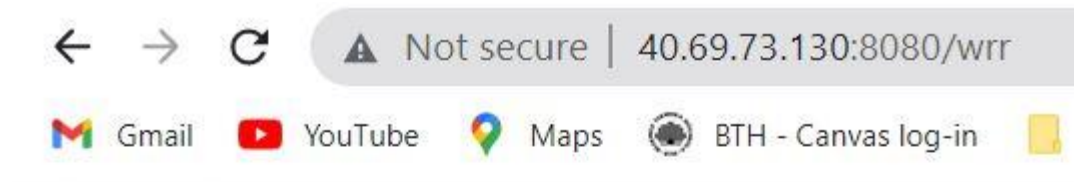
```

Python algorithm for Weighted Round Robin.

Workload Sensitivity Algorithm results using “/wrr”:



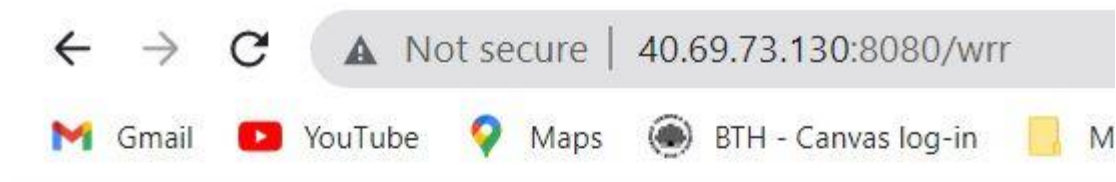
Hostname: e8a50b1c30f8



You are logged into Server1!

Hostname: 7feea7bb1923

Load on Server1: 0.1



You are logged into Server2!

Hostname: 95c878e3e3b4

Load on Server2: 0.4

CONCLUSION

We were able to load balance http in docker using Workload sensitivity, Round Robin and Weighted Round Robin without the use embedded NGINX to implement and demonstrate.