Group5
Rajnandini Bhutada: 20000612-T149
Charishma Ganta: 20000405-T405

ET2599 – PROJECT

Implementation and Demonstration of Load Balancer for HTTP Requests in MS Azure using Virtualization Techniques

Task:
To implement, demonstrate and test an adaptive web service that applies a load balancer.

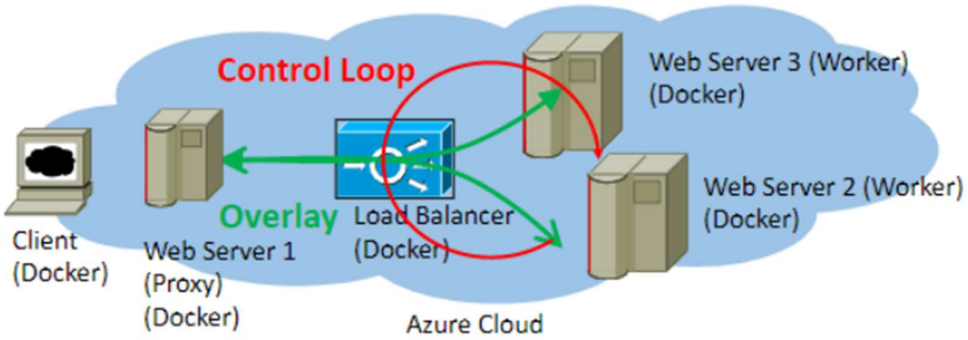

Figure 1

Introduction:
The main goal of this project is to understand the concept of Docker containers and virtualization techniques by implementing web servers and demonstrating load balancers.

Docker is an open-source application that allows administrators to create, manage, deploy, and replicate applications using containers. Containers can be thought of as a package that houses dependencies that an application requires to run at an operating system level. This means that each application deployed using Docker lives in an environment of its own and its requirements are handled separately.

Flask is a web micro-framework that is built on Python. The Flask framework is lightweight and flexible, yet highly structured, making it especially popular for small web apps written in Python. Deploying a Flask application with Docker will allow you to replicate the application across different servers with minimal reconfiguration.

According to Figure 1, vm13 acts as a proxy server which has the configuration of round-robin, weighted-round robin, and workload sensitivity, vm14 acts as webserver-1 and vm15 acts as webserver-2.

- VM13: ~/RRcode – path for round-robin code
  ~/WRRcode – path for workload sensitivity code
  ~/new – path for weighted-round robin code
- VM14: ~/app – path for web service code for server-1
- VM15: ~/app – path for web service code for server-2

Algorithm:

1. **Round-Robin:**
   The round-robin load balancing technique is the simplest way to distribute traffic across a group of servers. The load balancer forwards the incoming requests to dedicated servers sequentially (a one-by-one mechanism).
   Explanation:
   - **i**: This is the request number that is processed. It is initially set to zero.
   - **T**: This is the total number of servers that are available.

   Step 1: Begin
   Step 2: Assign the current request to the i mod Tth server.
   Step 3: Increment variable i value by 1.
   Step 4: Repeat steps 2 and 3 until there are no more requests.
   Step 5: End.

2. **Weighted-Round Robin:**
   In the weighted round-robin load balancer, the network administrator assigns a numeric weight to all of the servers behind the load balancer.
   Explanation:
   - **i**: This is the request number that is processed. It is initially set to zero.
   - **Weights**: This is an array that holds the weights for all the servers.
   - **T**: This is the total number of servers that are available

   Step 1: Begin.
   Step 2: Assign a specific number of requests to i mod Tth server, according to its numeric weight Weights[i].
   Step 3: Increment the value of i by a value of 1.
   Step 4: Repeat steps 2 and 3 until there are no more requests.
   Step 5: End.

3. **Workload sensitivity:**
   The balancer will receive workload updates from the servers. An update of the balancing is triggered by the received information.

Implementation of web service:

```
vm14@et2599vm14:~/app$ cat app.py
from flask import Flask
import os
import socket
import psutil

app = Flask(__name__)
@app.route("/")
def hello():
    html = "<h3>Hello {name}! I am from webserver1 </h3> <b>Hostname:</b> {hostname}<br/> <p> Load = {myvalue} </p>"
    return html.format(name="all", hostname=socket.gethostname(), myvalue=str(psutil.cpu_percent()))

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


if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8080)
```

Figure 2

```
vm14@et2599vm14:~/app$ cat Dockerfile
FROM python:2.7-slim
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org Flask
RUN pip install psutil
ENV NAME World
CMD ["python","app.py"]
```

Figure 3

Figure 2 is the flask application that runs on both VM14 and VM15 which initiates the starting of the service on both servers. Later, through VM13 as a proxy, different load-balancing algorithms are implemented among the servers.

Figure 4

Figure 5

Figure 4 and Figure 5 depict that the flask application is running.

Implementation of Round-Robin:

```
vm13@et2599vm13:~/RRcode$ cat app.py

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

servers = ['1', '2']

n=0
def get_servers():
        global n
        x = servers[n %len(servers)]
        n = n + 1
        return x

app = Flask(__name__)

ip1 = "http://104.41.211.213:8080"
ip2 = "http://40.85.126.76:8080"

@app.route('/')
def func():
        round_robin=get_servers()
        if round_robin == '1':
                response = requests.get(url = ip1)
                return str(response.content)
        elif round_robin == '2':
                response = requests.get(url = ip2)
                return str(response.content)
        else:
                return "Not Found", 404

if __name__ == "__main__":
        app.run(host='0.0.0.0', port=8080)
```

Figure 6

```
vm13@et2599vm13:~/RRcode$ cat Dockerfile
FROM python:2.7-slim
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org Flask
RUN pip install requests
CMD ["python", "app.py"]
```

Figure 7

Figure 6 is the code for round-robin load balancing implementation, while Figure 7 is the Dockerfile.

Figure 8

Figure 9

Figure 8 and Figure 9 show that round-robin load balancing is successfully implemented.

Implementation of weighted load sensitivity:

```
vm13@et2599vm13:~/WRRcode$ cat app.py
from flask import Flask
import requests
import sys
import os

app = Flask(__name__)

url1="http://104.41.211.213:8080"
url2="http://40.85.126.76:8080"

load_server1 = ""
load_server2 = ""

@app.route('/')
def func():
        global load_server1, load_server2
        load_server1 = requests.get("http://104.41.211.213:8080/load").content
        load_server2 = requests.get("http://40.85.126.76:8080/load").content
        if float(load_server1) > float(load_server2):
                response = requests.get(url=url1)
                return str(response.content)
        elif float(load_server1) <= float(load_server2):
                response = requests.get(url=url2)
                return str(response.content)
        else:
                return "Not Found", 404

if __name__ == "__main__":
        app.run(host='0.0.0.0',port=8080)
```

Figure 10

```
vm13@et2599vm13:~/WRRcode$ cat Dockerfile
FROM python:2.7-slim
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org Flask
RUN pip install requests
CMD ["python", "app.py"]
```

Figure 11

Figure 10 is the code for workload sensitivity while Figure 11 is its Dockerfile.

Figure 12

Figure 13

Figure 14

Figure 15

Figure 12, Figure 13, Figure 14, and Figure 15 depict the working of load sensitivity based on the server's load (CPU).

Implementation of weighted-round robin:

```
vm13@et2599vm13:~/new$ cat app.py
from flask import Flask , request, send_file, redirect, url_for, session
import requests
import sys
import os

servers = ['1', '2']
w1 = 1
w2 = 2

n=0
i=0
def get_servers():
        global n,i
        tw = w1 + w2
        if i %tw > w1:
                x = servers[0]
        else :
                x = servers[1]
        i = i + 1
        return x

app = Flask(__name__)

ip1 = "http://104.41.211.213:8080"
ip2 = "http://40.85.126.76:8080"

@app.route('/')
def func():
        round_robin=get_servers()
        if round_robin == "1":
                response = requests.get(url = ip1)
                return str(response.content)
        elif round_robin == "2":
                response = requests.get(url = ip2)
                return str(response.content)
        else:
                return "Not Found", 404

if __name__ == "__main__":
        app.run(host='0.0.0.0', port=8080)
```

Figure 16

```
vm13@et2599vm13:~/new$ cat Dockerfile
FROM python:2.7-slim
WORKDIR /app
ADD . /app
RUN pip install --trusted-host pypi.python.org Flask
RUN pip install requests
CMD ["python", "app.py"]
```

Figure 17

Figure 16 shows the code for the weighted round robin whereas Figure 17 shows its DockerFile.

Figure 18

Figure 19

Figure 20

Figure 18, Figure 19, and Figure 20 depict the working of a weighted round-robin algorithm where the load on server-2 is 2 and the load of server-1 is 1.

Conclusion:

Web service using load-balancing techniques is implemented and demonstrated on virtual machines.