

Otto-Friedrich-University Bamberg
Professorship for Computer Science,
Communication Services, Telecommunication,
Systems and Computer Networks



Foundation of Internet Communication

Assignment-03: Domain Name System (DNS) and Load Balancing

Submitted by:
Group J

Reem Eslam Mohamed Mekky Khalil
Shivasharan Reddy
Azar Ghadami
Reema Miranda
Manjunath B Marigoudar

Supervisor: Prof. Dr. Udo Krieger

Bamberg, June 07, 2020
Summer Term 2020

Contents

1	The Domain Information Groper (dig)	2
1.1	Determine the authoritative DNS servers for the top level domain ru.	2
1.2	Determine the addresses of the Internet DNS root servers. . .	3
1.3	Display the nameservers for the domain uni-bamberg.de	3
2	DNS Configuration with CoreDNS	4
2.1	Lab.conf	4
2.2	Startup files	4
2.3	Setup a Kàthara node to reach DNS server	5
2.3.1	Corefile file	5
2.3.2	db.root	6
2.3.3	db.gik.de	6
2.4	Add name server to pc1 and pc2	7
2.5	Capturing the name resolution on CD A with wireshark	8
2.6	Dig root domain in our case dns_root	9
2.7	DNS server setup at node DNS_lb	9
2.7.1	Corefile	9
2.7.2	db.gik.org	10
2.8	Modifying dns_root to forward name resolution of org to dns_lb	11
2.9	configure CoreDNS on dns_lb to load balance the entry gik.org	11
2.9.1	Corefile with org server block which handles load balancing	11
2.9.2	Curl gik.org	12
2.9.3	Disadvantages of DNS Load Balancing	13
2.10	Dig tool	14
3	Load Balancing with Traefik	15
3.1	Replacing web1 with three web servers	15
3.1.1	Startup files	16
3.1.2	Connecting devices in the network	17

3.2	Adding load-balancer traefik_lb	18
3.2.1	listen on port 80 and use a file provider	18
3.2.2	which forwards requests on gik.de to the new web servers	19
3.3	Adjusting the record of gik.de from dns root to point to traefik lb.	19
3.4	Adding the static routes to the topology	20
3.4.1	web_Sheldon routes	20
3.4.2	web_Howard routes	20
3.4.3	web_Leonard routes	20
3.4.4	Traefik_lb routes	20
3.5	Testing the load balancing behavior and add a weighted round robin to forward	21
3.6	Differences between DNS and software load balancing	22
3.6.1	pros and cons of DNS	22
3.6.2	Pros and cons of software load balancing	22

List of Figures

1.1	dig command: DNS servers for "ru" TLD	2
1.2	dig command: showing Internet DNS root servers	3
1.3	dig command: nameservers for a specific domain	3
2.1	Adding new nodes and configure there images and interfaces in Kàthara lab conf file	4
2.2	Example for new added nodes setup file with configuring it's route	5
2.3	Corefile for dns_root node with root and de Server blocks . . .	5
2.4	db.root file	6
2.5	db.root file	7
2.6	Adding nameserver to pc1 and testing that with curl gik.de . .	7
2.7	Result of Capturing pc1 curl gik.de on wireshark	8
2.8	Dig root domain	9
2.9	Corefile for dns_root node with server block forwarded to dns_root	9
2.10	Corefile declaration in dns_lb.startup file	10
2.11	db.gik.org File	10
2.12	Corefile File	11
2.13	Load Balance at dns_lb	12
2.14	Curl gik.org execution	13
2.15	Dig gik.org execution	14
3.1	Experiment configuration with traefik	15
3.2	web_Sheldon startup	16
3.3	web_Howard startup	16
3.4	web_Leonard startup	16
3.5	web_1 Replaced	17
3.6	Connecting web_Sheldon	17
3.7	connecting web_Howard	17
3.8	connecting web_Leonard	18
3.9	creating Traefik_lb	18
3.10	Port 80	18

3.11 Request forwarding	19
3.12 Adjusting the record of gik.de from dns root to point to traefik_lb	19
3.13 Web-Sheldon routes	20
3.14 Web-Howard routes	20
3.15 Web_Leonard routes	20
3.16 Traefik_lb routes	20
3.17 Round robin	21

Chapter 1

The Domain Information Groper (dig)

1.1 Determine the authoritative DNS servers for the top level domain ru.

Command: `dig ns ru`

```
C:\Program Files\ISC BIND 9\bin>dig ns ru
;; <<>> DiG 9.11.19 <<>> ns ru
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 30850
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;ru.                                IN      NS
;; ANSWER SECTION:
ru.      21526   IN      NS      a.dns.ripgn.net.
ru.      21526   IN      NS      f.dns.ripgn.net.
ru.      21526   IN      NS      b.dns.ripgn.net.
ru.      21526   IN      NS      d.dns.ripgn.net.
ru.      21526   IN      NS      e.dns.ripgn.net.
;; Query time: 32 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu May 28 17:23:48 W. Europe Daylight Time 2020
;; MSG SIZE  rcvd: 123

C:\Program Files\ISC BIND 9\bin>dig ns ru +short
b.dns.ripgn.net.
e.dns.ripgn.net.
d.dns.ripgn.net.
f.dns.ripgn.net.
a.dns.ripgn.net.
```

Figure 1.1: dig command: DNS servers for "ru" TLD

1.2 Determine the addresses of the Internet DNS root servers.

Command: `dig ns .`

```
C:\Program Files\ISC BIND 9\bin>dig ns .
; <<>> DiG 9.11.19 <<>> ns .
;; global options: +cmd
;; Got answer:
;;->HEADER<- opcode: QUERY, status: NOERROR, id: 38244
;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:;, udp: 512
;; QUESTION SECTION:
; .
      IN      NS
;; ANSWER SECTION:
; 86054 IN      NS      e.root-servers.net.
; 86054 IN      NS      a.root-servers.net.
; 86054 IN      NS      n.root-servers.net.
; 86054 IN      NS      f.root-servers.net.
; 86054 IN      NS      d.root-servers.net.
; 86054 IN      NS      c.root-servers.net.
; 86054 IN      NS      b.root-servers.net.
; 86054 IN      NS      l.root-servers.net.
; 86054 IN      NS      j.root-servers.net.
; 86054 IN      NS      g.root-servers.net.
; 86054 IN      NS      k.root-servers.net.
; 86054 IN      NS      h.root-servers.net.
; 86054 IN      NS      i.root-servers.net.
;; Query time: 19 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu May 28 18:10:58 W. Europe Daylight Time 2020
;; MSG SIZE rcvd: 239
```

Figure 1.2: dig command: showing Internet DNS root servers

1.3 Display the nameservers for the domain uni-bamberg.de

Command: `dig ns uni-bamberg.de`

```
C:\Program Files\ISC BIND 9\bin>dig NS uni-bamberg.de
; <<>> DiG 9.11.19 <<>> NS uni-bamberg.de
;; global options: +cmd
;; Got answer:
;;->HEADER<- opcode: QUERY, status: NOERROR, id: 21113
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:;, udp: 512
;; QUESTION SECTION:
; uni-bamberg.de.
      IN      NS
;; ANSWER SECTION:
uni-bamberg.de. 21499 IN      NS      ns02.rz.uni-bamberg.de.
uni-bamberg.de. 21499 IN      NS      dns-3.dfn.de.
uni-bamberg.de. 21499 IN      NS      ns01.rz.uni-bamberg.de.
;; Query time: 27 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Thu May 28 16:57:42 W. Europe Daylight Time 2020
;; MSG SIZE rcvd: 108
```

Nameservers

Figure 1.3: dig command: nameservers for a specific domain

Chapter 2

DNS Configuration with CoreDNS

In this Section we are going to setup a Kàthara node to reach DNS server

2.1 Lab.conf

First step is, we need to add some configurations to the conf file we created in the previous assignment.

```
25
26 dns_root[0]="F"
27 dns_root[image]="unibaktr/alpine:coredns"
28
29 dns_lb[0]="G"
30 dns_lb[image]="unibaktr/alpine:coredns"
31
32
33 web1[0]="D"
34 web1[image]="unibaktr/alpine:whoami"
35
36 web_penny[0]="E"
37 web_penny[image]="unibaktr/alpine:whoami"
38
39 web_bernadette[0]="E"
40 web_bernadette[image]="unibaktr/alpine:whoami"
41
42
43 web_amy[0]="E"
44 web_amy[image]="unibaktr/alpine:whoami"
45
```

Figure 2.1: Adding new nodes and configure there images and interfaces in Kàthara lab conf file

2.2 Startup files

Second step we need to also configure the startup files for each new node and configure there routes so the whole network would be connected.

A screenshot of a terminal window titled 'web_amy.startup — Consoles'. The terminal shows a shell prompt '#!/bin/sh' followed by three lines of commands: 'ip addr add 50.50.0.102/25 brd + dev eth0', 'ip route add default via 50.50.0.3 dev eth0', and a cursor at the end of the third line.

```
web_amy.startup X
web_amy.startup
1 #!/bin/sh
2 ip addr add 50.50.0.102/25 brd + dev eth0
3 ip route add default via 50.50.0.3 dev eth0
```

Figure 2.2: Example for new added nodes setup file with configuring it's route

2.3 Setup a Kàthara node to reach DNS server

Third, We will setup `dns_root` to be an authoritative server for the root domain and `de`. In order to do that we will need to create a folder with the same name of the node. The folder will contain three main files.

2.3.1 Corefile file

Corefile is used to configure `coreDns`. When `coreDns` runs the first thing it will do, look for a file named **Corefile**. In `coreFile` we define the `Server` blocks we will use. In this section we will use two server blocks. The first one is the root `.` which is responsible for all zones below the root zone. The second server block we will use is `de` zone.

```
. {
    log
    errors
    file /hostlab/dns_root/db.root
}

de {
    log
    errors
    file /hostlab/dns_root/db.gik.de
}
```

Figure 2.3: Corefile for `dns_root` node with root and `de` Server blocks

As we can see in the `coreFile` we are calling two more files **db.root** and **db.gik.de**. Now the question what are those files. These files are used to

support and define one or more zone.

2.3.2 db.root

In this file will define the root server and how the protocol will be as in 2.4 you will see that you define the TTL (Time to live) which is how much time packets will stay in that network and then you find that you define some attributes for the server like expire time , cache time .. etc. Then you will make the root server look to the ip of de as shown.

```
dns_root / = db.root
1  $TTL  60000
2  @IN SOA ROOT-SERVER. root.ROOT-SERVER. (
3      2006031201 ; serial
4      28800 ; refresh
5      14400 ; retry
6      3600000 ; expire
7      0 ; negative cache ttl
8  )
9  IN NS ROOT-SERVER.
10 ROOT-SERVER. IN A 1.1.1.1
11
12 de. IN NS dns.de.
13 dns.de. IN A 1.1.1.1
```

Figure 2.4: db.root file

2.3.3 db.gik.de

In this file will define dns.de and how the protocol will be as in 2.5. Notice it is same as 2.4 but instead you will make dns.de server look to gik and make it point to the ip of web1.

```

1 $TTL 60000
2 @ IN SOA dns.de. root.dns.de. (
3     2006031201 ; serial
4     28 ; refresh
5     14 ; retry
6     3600000 ; expire
7     0 ; negative cache ttl
8 )
9 IN NS dns.de.
0 dns IN A 1.1.1.1
1 gik. IN NS gik.de.
2 gik IN A 40.40.0.100
3
4
5

```

Figure 2.5: db.root file

2.4 Add name server to pc1 and pc2

Now we will add nameserver entry to pc1 and pc2 that points to dns.root and then will test the connectivity using curl from any pc as shown in 2.6 for pc1.

The screenshot shows the VS Code interface with the `resolv.conf` file open for `pc1`. The file content is:

```

1 nameserver 1.1.1.1
2

```

The Explorer sidebar shows the project structure, including `dns_lb`, `dns_root`, `pc1`, and `pc2`. The `resolv.conf` file is highlighted under `pc1`.

The terminal window shows the following output:

```

(base) Reems-MacBook-Pro:conFiles reemeslam$ kathara connect pc1
/ # curl gik.de
I'm web1
/ #

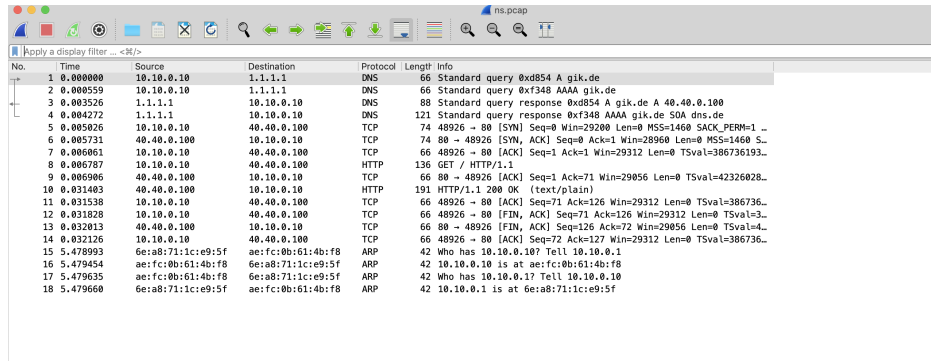
```

The terminal also displays a table of running containers:

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
p02vpaGGAnr94Q69ux7uPQ	reemeslam	r1	running 0.00% 15.53 MB / 1.94 GB 0.78% 2.02 KB / 0 B
p02vpaGGAnr94Q69ux7uPQ	reemeslam	r2	running 0.00% 19.77 MB / 1.94 GB 0.99% 1.01 KB / 0 B
p02vpaGGAnr94Q69ux7uPQ	reemeslam	r3	running 0.00% 876.0 KB / 1.94 GB 0.04% 1.51 KB / 0 B
p02vpaGGAnr94Q69ux7uPQ	reemeslam	web1	running 0.00% 1.47 MB / 1.94 GB 0.07% 516.0 B / 0 B
p02vpaGGAnr94Q69ux7uPQ	reemeslam	web_amy	running 0.00% 1.53 MB / 1.94 GB 0.08% 426.0 B / 0 B
p02vpaGGAnr94Q69ux7uPQ	reemeslam	web_bernadette	running 0.00% 1.62 MB / 1.94 GB 0.08% 426.0 B / 0 B
p02vpaGGAnr94Q69ux7uPQ	reemeslam	web_penny	running 0.00% 1.5 MB / 1.94 GB 0.08% 266.0 B / 0 B

Figure 2.6: Adding nameserver to pc1 and testing that with curl gik.de

2.5 Capturing the name resolution on CD A with wireshark



The image shows a Wireshark packet capture window with the filter 'Apply a display filter ... <?>'. The packet list shows 18 packets. The first three are DNS queries from 10.10.0.10 to 1.1.1.1. The fourth is a DNS response from 1.1.1.1 to 10.10.0.10. The fifth is a TCP SYN packet from 10.10.0.10 to 40.40.0.100. The sixth is a TCP SYN-ACK packet from 40.40.0.100 to 10.10.0.10. The seventh is a TCP ACK packet from 10.10.0.10 to 40.40.0.100. The eighth is an HTTP GET packet from 10.10.0.10 to 40.40.0.100. The ninth is a TCP ACK packet from 40.40.0.100 to 10.10.0.10. The tenth is an HTTP 200 OK packet from 40.40.0.100 to 10.10.0.10. The eleventh is a TCP ACK packet from 10.10.0.10 to 40.40.0.100. The twelfth is a TCP FIN packet from 10.10.0.10 to 40.40.0.100. The thirteenth is a TCP FIN-ACK packet from 40.40.0.100 to 10.10.0.10. The fourteenth is a TCP ACK packet from 40.40.0.100 to 10.10.0.10. The fifteenth is an ARP request from 6e:a8:71:1c:e9:5f to ae:fc:0b:61:4b:f8. The sixteenth is an ARP response from ae:fc:0b:61:4b:f8 to 6e:a8:71:1c:e9:5f. The seventeenth is an ARP request from 6e:a8:71:1c:e9:5f to ae:fc:0b:61:4b:f8. The eighteenth is an ARP response from ae:fc:0b:61:4b:f8 to 6e:a8:71:1c:e9:5f.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.10.0.10	1.1.1.1	DNS	66	Standard query 0xd854 A gik.de
2	0.000559	10.10.0.10	1.1.1.1	DNS	66	Standard query 0xf348 AAAA gik.de
3	0.003526	1.1.1.1	10.10.0.10	DNS	88	Standard query response 0xd854 A gik.de A 40.40.0.100
4	0.004272	1.1.1.1	10.10.0.10	DNS	121	Standard query response 0xf348 AAAA gik.de SOA dns.de
5	0.005026	10.10.0.10	40.40.0.100	TCP	74	48926 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1
6	0.005731	40.40.0.100	10.10.0.10	TCP	74	80 → 48926 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 S...
7	0.006061	10.10.0.10	40.40.0.100	TCP	66	48926 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=386736193...
8	0.006787	10.10.0.10	40.40.0.100	HTTP	136	GET / HTTP/1.1
9	0.006906	40.40.0.100	10.10.0.10	TCP	66	80 → 48926 [ACK] Seq=1 Ack=71 Win=29056 Len=0 TSval=42326028...
10	0.031403	40.40.0.100	10.10.0.10	HTTP	191	HTTP/1.1 200 OK (text/plain)
11	0.031538	10.10.0.10	40.40.0.100	TCP	66	48926 → 80 [ACK] Seq=71 Ack=126 Win=29312 Len=0 TSval=386736...
12	0.031928	10.10.0.10	40.40.0.100	TCP	66	48926 → 80 [FIN, ACK] Seq=71 Ack=126 Win=29312 Len=0 TSval=3...
13	0.032013	40.40.0.100	10.10.0.10	TCP	66	80 → 48926 [FIN, ACK] Seq=126 Ack=72 Win=29056 Len=0 TSval=4...
14	0.032126	10.10.0.10	40.40.0.100	TCP	66	48926 → 80 [ACK] Seq=72 Ack=127 Win=29312 Len=0 TSval=386736...
15	5.478993	6e:a8:71:1c:e9:5f	ae:fc:0b:61:4b:f8	ARP	42	Who has 10.10.0.10? Tell 10.10.0.1
16	5.479454	ae:fc:0b:61:4b:f8	6e:a8:71:1c:e9:5f	ARP	42	10.10.0.10 is at ae:fc:0b:61:4b:f8
17	5.479635	ae:fc:0b:61:4b:f8	6e:a8:71:1c:e9:5f	ARP	42	Who has 10.10.0.1? Tell 10.10.0.10
18	5.479660	6e:a8:71:1c:e9:5f	ae:fc:0b:61:4b:f8	ARP	42	10.10.0.1 is at 6e:a8:71:1c:e9:5f

Figure 2.7: Result of Capturing pc1 curl gik.de on wireshark

Lets Now explain the steps when you curl gik.de from pc1

- it begins with pc1 wants to send gik.de a http request so it needs it's ip address
- pc1 request for gik.de ip address will be sent to the root domain asking for the ip of gik.de
- root domain will reply with where you can find de server and give it web1 ip address
- pc1 will send to web1 ip address request asking for the ip of gik.de
- web1 will ensure that he can connect with gik.de and will give pc1 the ip of gik.de
- so pc1 now can send directly gik.de http request using gik.de ip address

2.6 Dig root domain in our case dns_root

```
-- --
/ # dig dns_root

; <<> DiG 9.14.12 <<> dns_root
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NXDOMAIN, id: 48384
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags: udp: 4096
;; COOKIE: 89c3cd6bcafd6f9a (echoed)
;; QUESTION SECTION:
;dns_root.                IN      A

;; AUTHORITY SECTION:
\@in.      60000    IN      SOA      root-server. root.root-server. 2006031201 28800 14400 3600000 0

;; Query time: 5 msec
```

Figure 2.8: Dig root domain

2.7 DNS server setup at node DNS_lb

We will setup dns_lb to be an authoritative server for the org domain. The Setup includes creating a folder dns_lb which contains 2 files.

2.7.1 Corefile

In Corefile of dns_lb we define 2 server blocks one for each domains we want to route to. The first server block is forwarded to dns_root where we have setup root domain.

```
. {
    log
    errors
    forward 1.1.1.1
}
```

Figure 2.9: Corefile for dns_root node with server block forwarded to dns_root

Define Corefile in dns_lb.startup file

```
#!/bin/sh
ip addr add 2.2.2.2/8 brd + dev eth0
ip route add default via 2.0.0.3 dev eth0
coredns -conf /hostlab/dns_lb/Corefile &
```

Figure 2.10: Corefile declaration in dns_lb.startup file

2.7.2 db.gik.org

This is DNS zone file, and it will have gik.org name entry. The gik.org points to web servers web bernadette, web amy, and web penny which can be seen in last 3 lines of the file (IP Address).

```
$TTL      60000
@          IN      SOA     dns_lb.gik.org.  root.dns_lb.gik.org. (
2006031201 ; serial
28 ; refresh
14 ; retry
3600000 ; expire
0 ; negative cache ttl
)
@          IN      NS      dns_lb.gik.org.
dns_lb     IN      A        2.2.2.2
IN NS gik.org.
gik IN A 50.50.0.100
gik IN A 50.50.0.101
gik IN A 50.50.0.102
```

Figure 2.11: db.gik.org File

2.8 Modifying dns_root to forward name resolution of org to dns_lb

In this section we modify the Corefile of dns_root to forward the gik.org to dns_lb.

```
org {  
    log  
    errors  
    forward org 2.2.2.2  
}
```

Figure 2.12: Corefile File

2.9 configure CoreDNS on dns_lb to load balance the entry gik.org

DNS load balancing is used to send the DNS requests of the domain to different server machines in order to reduce the load and make the responses faster as there will less load on each of the server machines.

2.9.1 Corefile with org server block which handles load balancing

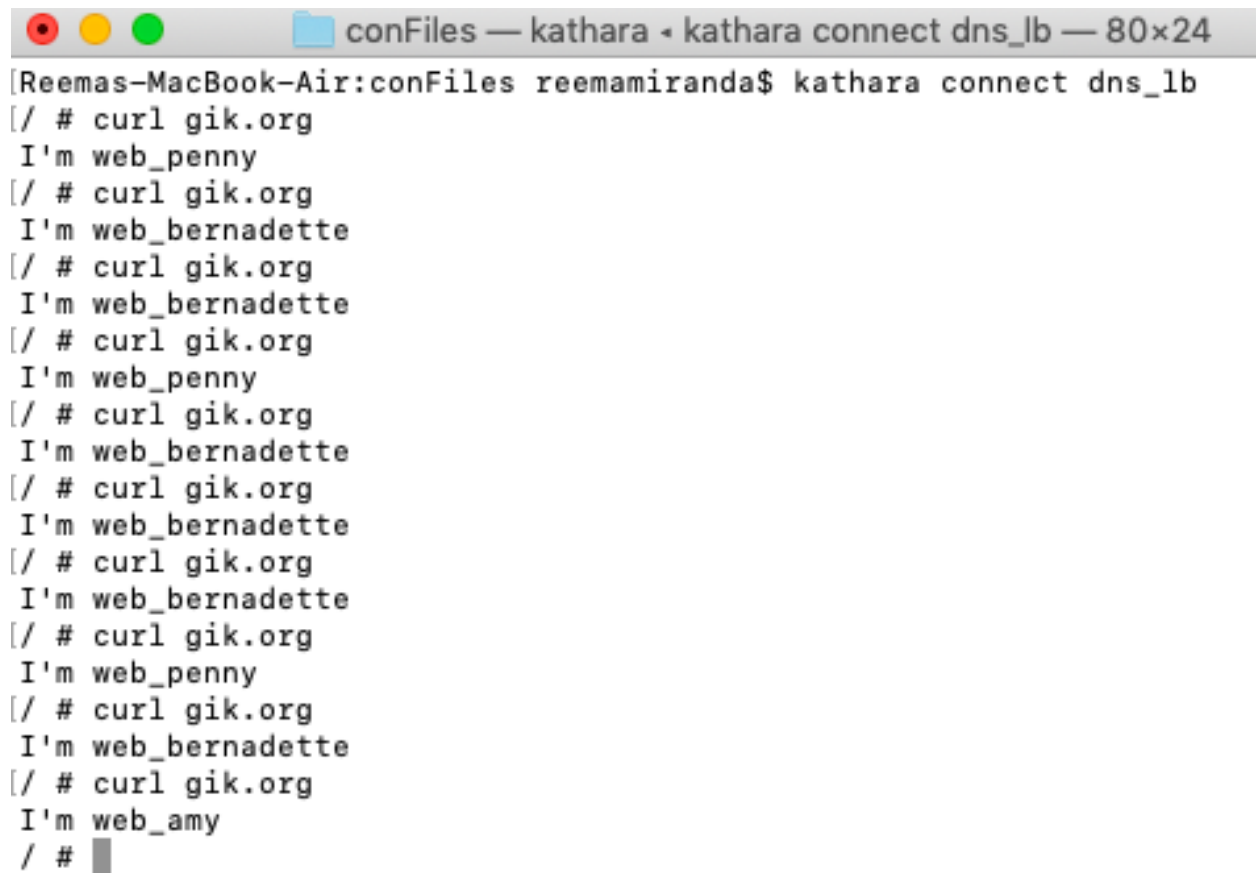
In this section we add Load balance command: "loadbalance round robin" in corfile within org server block

```
. {  
    log  
    errors  
    forward 1.1.1.1  
}  
  
org {  
    log  
    errors  
    loadbalance round_robin  
    file /hostlab/dns_lb/db.gik.org  
}
```

Figure 2.13: Load Balance at dns_lb

2.9.2 Curl gik.org

we use `curl gik.org` command to know which server the domain is pointing to. When we run `curl gik.org` several times we observe that the domain request is forwarded to same server many times and in random order.

A terminal window titled "conFiles — kathara • kathara connect dns_lb — 80x24". The prompt is "Reemas-MacBook-Air:conFiles reemamiranda\$". The user enters "kathara connect dns_lb". The terminal shows a series of "curl gik.org" commands and responses. The responses are: "I'm web_penny", "I'm web_bernadette", "I'm web_bernadette", "I'm web_penny", "I'm web_bernadette", "I'm web_bernadette", "I'm web_bernadette", "I'm web_penny", "I'm web_bernadette", "I'm web_bernadette", "I'm web_penny", "I'm web_bernadette", "I'm web_amy". The last line is a partial command " / # █".

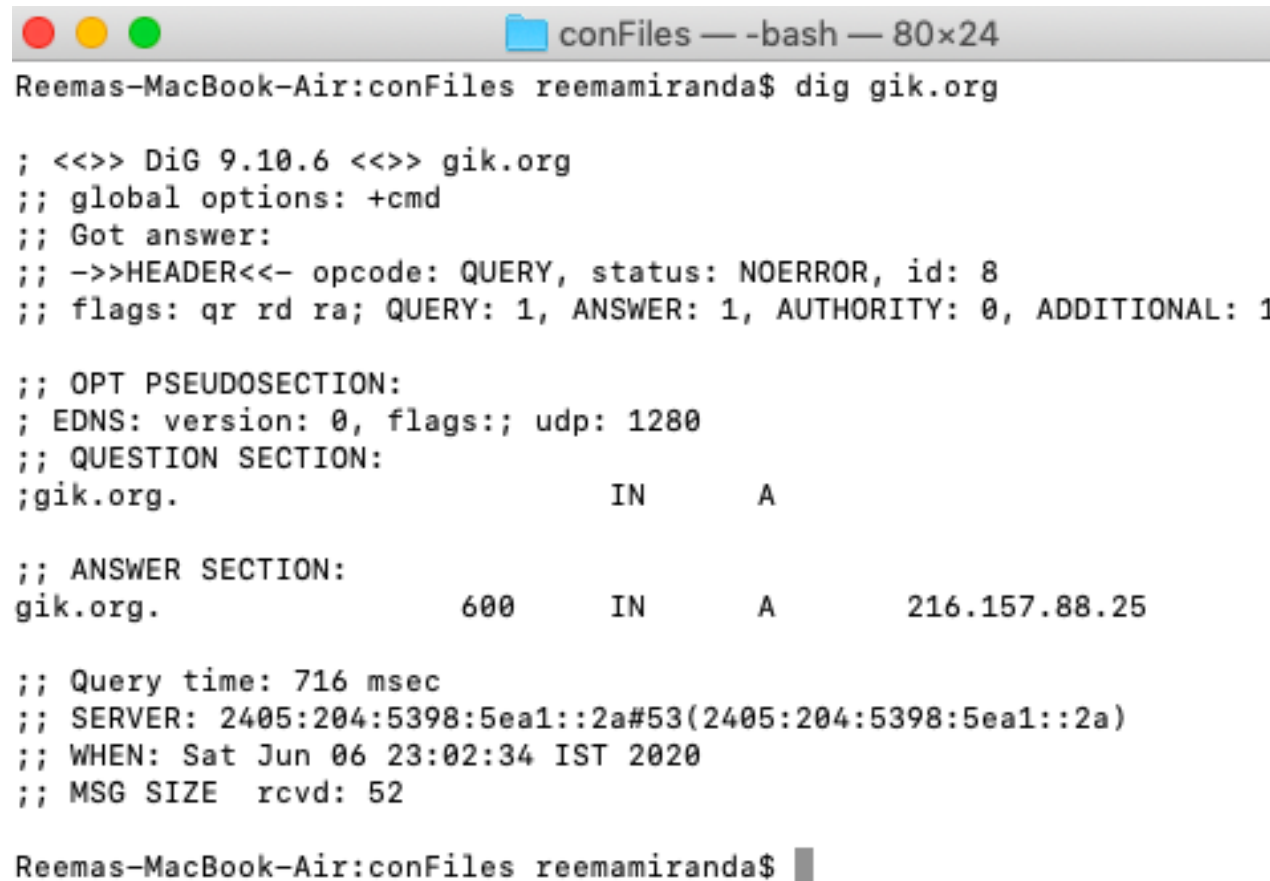
```
Reemas-MacBook-Air:conFiles reemamiranda$ kathara connect dns_lb
[/ # curl gik.org
I'm web_penny
[/ # curl gik.org
I'm web_bernadette
[/ # curl gik.org
I'm web_bernadette
[/ # curl gik.org
I'm web_penny
[/ # curl gik.org
I'm web_bernadette
[/ # curl gik.org
I'm web_bernadette
[/ # curl gik.org
I'm web_bernadette
[/ # curl gik.org
I'm web_penny
[/ # curl gik.org
I'm web_bernadette
[/ # curl gik.org
I'm web_amy
 / # █
```

Figure 2.14: Curl gik.org execution

2.9.3 Disadvantages of DNS Load Balancing

Client request is forwarded to same server destination even if it is overloaded. We can observe random distribution of request to the servers which leads to traffic congestion.

2.10 Dig tool



```
Reemas-MacBook-Air:conFiles reemamiranda$ dig gik.org

; <<>> DiG 9.10.6 <<>> gik.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<- opcode: QUERY, status: NOERROR, id: 8
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;gik.org.                IN      A

;; ANSWER SECTION:
gik.org.                  600     IN      A      216.157.88.25

;; Query time: 716 msec
;; SERVER: 2405:204:5398:5ea1::2a#53(2405:204:5398:5ea1::2a)
;; WHEN: Sat Jun 06 23:02:34 IST 2020
;; MSG SIZE rcvd: 52

Reemas-MacBook-Air:conFiles reemamiranda$
```

Figure 2.15: Dig gik.org execution

Chapter 3

Load Balancing with Traefik

In this section we are going to use traefik for load balancing

3.1 Replacing web1 with three web servers

Creating web servers(web_Sheldon,web_Leonard,web_Howard) In this section we are going to replace web1 with three new web-servers.

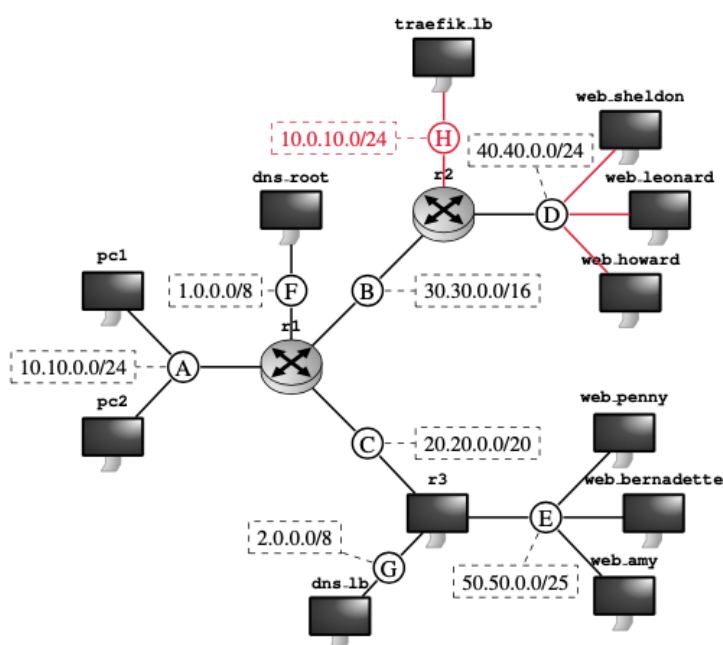
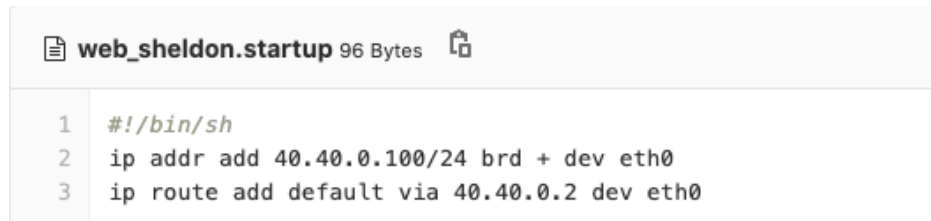


Figure 3.1: Experiment configuration with traefik

3.1.1 Startup files

Below mentioned are the start-up files of the new web servers.



The screenshot shows a file named **web_sheldon.startup** with a size of 96 Bytes. The file content is as follows:

```
1  #!/bin/sh
2  ip addr add 40.40.0.100/24 brd + dev eth0
3  ip route add default via 40.40.0.2 dev eth0
```

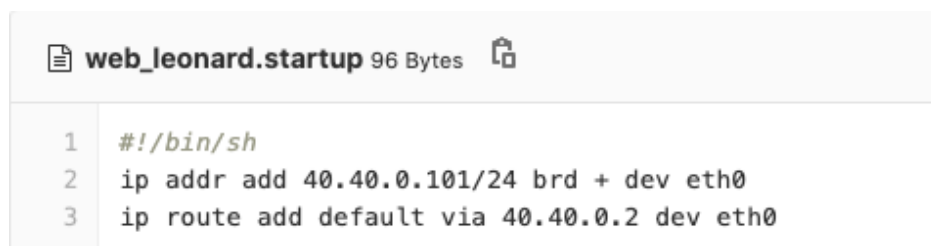
Figure 3.2: web_Sheldon startup



The screenshot shows a file named **web_howard.startup** with a size of 96 Bytes. The file content is as follows:

```
1  #!/bin/sh
2  ip addr add 40.40.0.102/24 brd + dev eth0
3  ip route add default via 40.40.0.2 dev eth0
```

Figure 3.3: web_Howard startup



The screenshot shows a file named **web_leonard.startup** with a size of 96 Bytes. The file content is as follows:

```
1  #!/bin/sh
2  ip addr add 40.40.0.101/24 brd + dev eth0
3  ip route add default via 40.40.0.2 dev eth0
```

Figure 3.4: web_Leonard startup

LAB HASH	USER	MACHINE NAME	STATUS	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	dns_lb	running	0.00%	15.18 MB / 1.94 GB	0.76%	696.0 B / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	dns_root	running	0.00%	23.14 MB / 1.94 GB	1.16%	516.0 B / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	pc1	running	0.00%	2.01 MB / 1.94 GB	0.10%	836.0 B / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	pc2	running	0.00%	692.0 KB / 1.94 GB	0.03%	836.0 B / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	r1	running	0.00%	24.61 MB / 1.94 GB	1.24%	3.04 KB / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	r2	running	0.00%	28.45 MB / 1.94 GB	1.43%	2.31 KB / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	r3	running	0.00%	1.85 MB / 1.94 GB	0.09%	1.78 KB / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	traefik_lb	running	0.00%	880.0 KB / 1.94 GB	0.04%	516.0 B / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	web_amy	running	0.00%	1.64 MB / 1.94 GB	0.08%	266.0 B / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	web_bernadette	running	0.00%	1.41 MB / 1.94 GB	0.07%	426.0 B / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	web_howard	running	0.00%	1.32 MB / 1.94 GB	0.07%	516.0 B / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	web_leonard	running	0.00%	1.43 MB / 1.94 GB	0.07%	516.0 B / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	web_penny	running	0.00%	1.39 MB / 1.94 GB	0.07%	516.0 B / 0 B
1wYSn1k3zWPmwjBxiqkJg	shivasharanreddyreddy	web_sheldon	running	0.00%	6.45 MB / 1.94 GB	0.27%	516.0 B / 0 B

Figure 3.5: web_1 Replaced

3.1.2 Connecting devices in the network

We configure all three web servers in the network and try to reach them.

```
/ # ping 40.40.0.100
PING 40.40.0.100 (40.40.0.100): 56 data bytes
64 bytes from 40.40.0.100: seq=0 ttl=62 time=0.375 ms
64 bytes from 40.40.0.100: seq=1 ttl=62 time=0.238 ms
64 bytes from 40.40.0.100: seq=2 ttl=62 time=0.239 ms
64 bytes from 40.40.0.100: seq=3 ttl=62 time=0.240 ms
64 bytes from 40.40.0.100: seq=4 ttl=62 time=0.242 ms
^C
--- 40.40.0.100 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.238/0.266/0.375 ms
```

Figure 3.6: Connecting web_Sheldon

```
/ # ping 40.40.0.101
PING 40.40.0.101 (40.40.0.101): 56 data bytes
64 bytes from 40.40.0.101: seq=0 ttl=62 time=0.712 ms
64 bytes from 40.40.0.101: seq=1 ttl=62 time=0.240 ms
64 bytes from 40.40.0.101: seq=2 ttl=62 time=0.198 ms
64 bytes from 40.40.0.101: seq=3 ttl=62 time=0.295 ms
64 bytes from 40.40.0.101: seq=4 ttl=62 time=0.238 ms
^C
--- 40.40.0.101 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.198/0.336/0.712 ms
```

Figure 3.7: connecting web_Howard

```

[/ # ping 40.40.0.102
PING 40.40.0.102 (40.40.0.102): 56 data bytes
64 bytes from 40.40.0.102: seq=0 ttl=62 time=0.468 ms
64 bytes from 40.40.0.102: seq=1 ttl=62 time=0.244 ms
64 bytes from 40.40.0.102: seq=2 ttl=62 time=0.239 ms
64 bytes from 40.40.0.102: seq=3 ttl=62 time=0.244 ms
64 bytes from 40.40.0.102: seq=4 ttl=62 time=0.404 ms
^C
--- 40.40.0.102 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.239/0.319/0.468 ms

```

Figure 3.8: connecting web_Leonard

3.2 Adding load-balancer traefik_lb

In this part we add loadbalancer traefik_lb and configure it.

```

Part -03 config files > traefik_lb.startup
1  #!/bin/sh
2  ip addr add 10.0.10.1/24 brd + dev eth0
3  ip route add default via 10.0.10.2 dev eth0
4  traefik --configFile=hostlab/traefik_lb/traefik_lb.toml &

```

Figure 3.9: creating Traefik_lb

3.2.1 listen on port 80 and use a file provider

This can be achieved by 'address' option at entry points.

```

traefik_lb.toml
## Static configuration
[entryPoints]
  [entryPoints.web]
    address = ":80"

[providers]
  [providers.file]
    directory = "/traefik_lb/services/"
    watch = true

```

Figure 3.10: Port 80

3.2.2 which forwards requests on gik.de to the new web servers

```
Part -03 config files > traefik_lb > services > ⚙ gik.toml
1  ## Dynamic configuration
2  [http.routers]
3    [http.routers.gik]
4      rule = "Host(`gik.de`)"
5      service = "app"
6
```

Figure 3.11: Request forwarding

3.3 Adjusting the record of gik.de from dns root to point to traefik lb.

```
$TTL 60000
@ IN SOA dns.de. root.dns.de. (
    2006031201 ; serial
    28 ; refresh
    14 ; retry
    3600000 ; expire
    0 ; negative cache ttl
)
IN NS dns.de.
dns IN A 1.1.1.1
gik. IN NS gik.de.
gik IN A 10.0.10.1
```

Figure 3.12: Adjusting the record of gik.de from dns root to point to traefik_lb

3.4 Adding the static routes to the topology

In this section we created three new web servers and a traefik_lb so we are adding routing typologies for the same.

3.4.1 web_Sheldon routes

```
[web_sheldon [/app]# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 40.40.0.2 0.0.0.0 UG 0 0 0 eth0
40.40.0.0 * 255.255.255.0 U 0 0 0 eth0
```

Figure 3.13: Web-Sheldon routes

3.4.2 web_Howard routes

```
[web_howard [/app]# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 40.40.0.2 0.0.0.0 UG 0 0 0 eth0
40.40.0.0 * 255.255.255.0 U 0 0 0 eth0
```

Figure 3.14: Web-Howard routes

3.4.3 web_Leonard routes

```
[web_leonard [/app]# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 40.40.0.2 0.0.0.0 UG 0 0 0 eth0
40.40.0.0 * 255.255.255.0 U 0 0 0 eth0
```

Figure 3.15: Web_Leonard routes

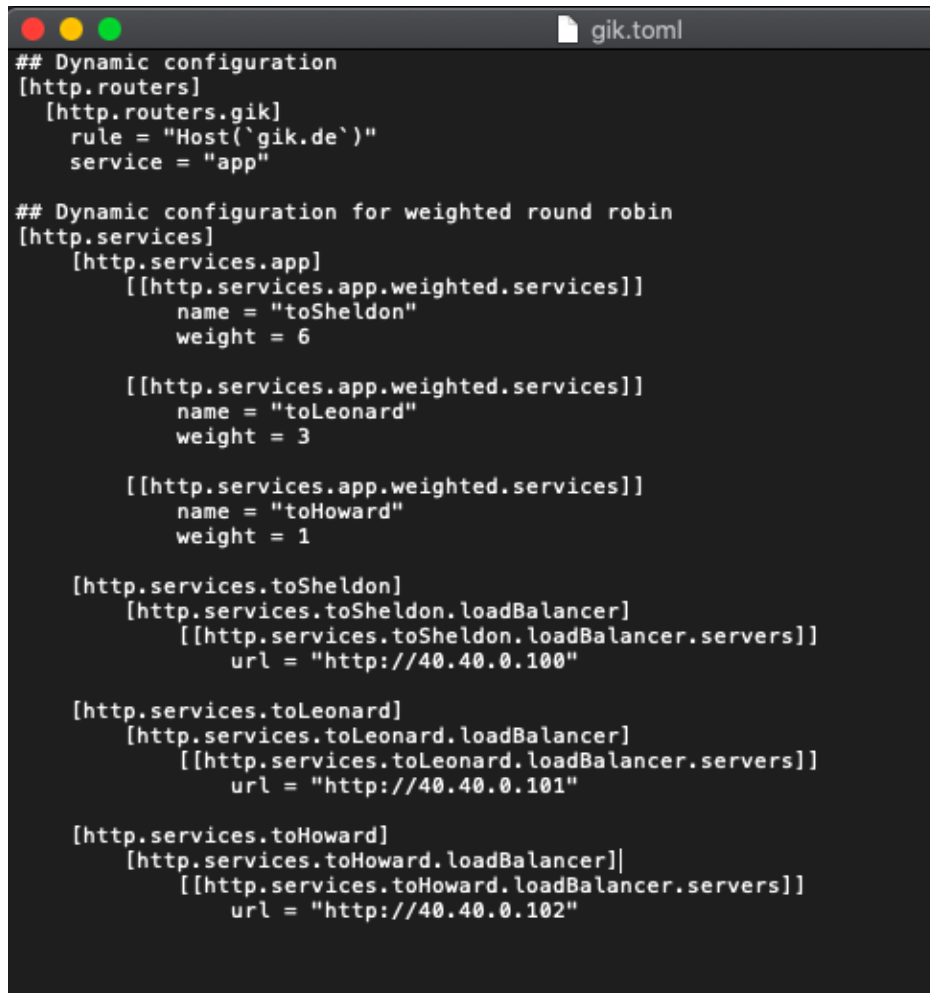
3.4.4 Traefik_lb routes

```
[traefik_lb [/]# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 10.0.10.2 0.0.0.0 UG 0 0 0 eth0
10.0.10.0 * 255.255.255.0 U 0 0 0 eth0
```

Figure 3.16: Traefik_lb routes

3.5 Testing the load balancing behavior and add a weighted round robin to forward

60 % of the requests to web Sheldon,
30 % of the requests to web Leonard,
and 10 % of the requests to web Howard.



```
## Dynamic configuration
[http.routers]
[http.routers.gik]
  rule = "Host(`gik.de`)"
  service = "app"

## Dynamic configuration for weighted round robin
[http.services]
[http.services.app]
  [[http.services.app.weighted.services]]
    name = "toSheldon"
    weight = 6

    [[http.services.app.weighted.services]]
      name = "toLeonard"
      weight = 3

    [[http.services.app.weighted.services]]
      name = "toHoward"
      weight = 1

[http.services.toSheldon]
[http.services.toSheldon.loadBalancer]
  [[http.services.toSheldon.loadBalancer.servers]]
    url = "http://40.40.0.100"

[http.services.toLeonard]
[http.services.toLeonard.loadBalancer]
  [[http.services.toLeonard.loadBalancer.servers]]
    url = "http://40.40.0.101"

[http.services.toHoward]
[http.services.toHoward.loadBalancer]
  [[http.services.toHoward.loadBalancer.servers]]
    url = "http://40.40.0.102"
```

Figure 3.17: Round robin

3.6 Differences between DNS and software load balancing

- Dns load balancing is easy to configure when compared to software load balancing
- In DNS we can assign multiple IP address but in software based load balancing it requires one specified ip address.
- In DNS load balancing client request is forwarded to same server destination even if is overloaded, But in software based load balancing we dont observe this pit fall.

3.6.1 pros and cons of DNS

- Easy to configure and understand.
- Multiple IP addresses can be assigned to the host record.It divides workload equally.
- DNS based cluster nodes don't require multiple network interface cards
- No capability other than round-robin.
- DNS cannot identify if the server is down.
- Each server requires a different IP address

3.6.2 Pros and cons of software load balancing

- Cost effective- we don't require physical hardware systems to perform the task.
- It does not require a separate public IP for each server
- Compared to hardware load balancer, the main drawback to software load balancer is in its performance.