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

List of Figures

Chapter 1

The Domain Information Groper (dig)

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

Command: *dig ns ru*

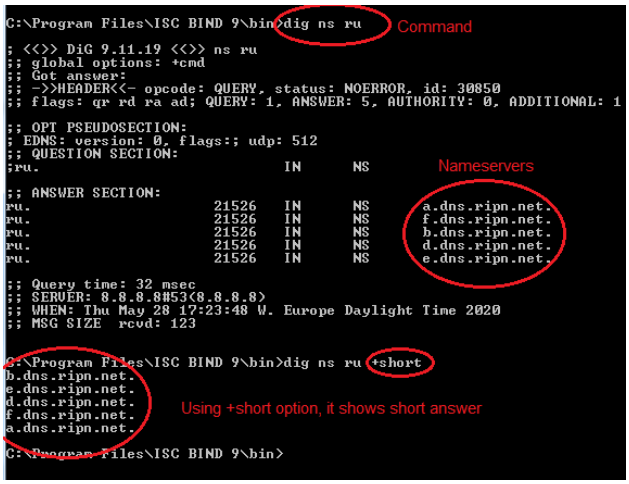

Figure 1.1: dig command: DNS servers for "ru" TLD

1.2 Determine the addresses of the Internet DNS root servers.

Command: *dig ns .*

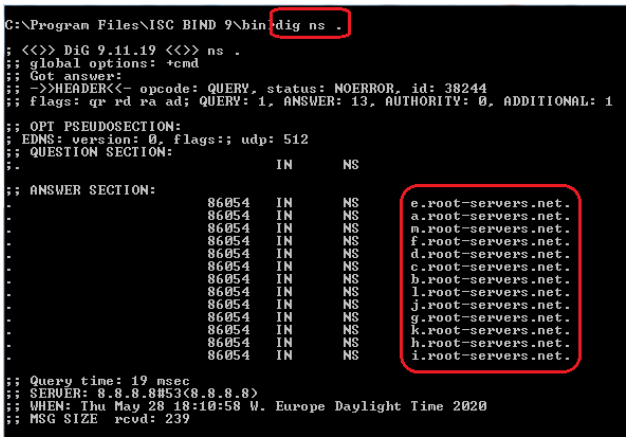

Figure 1.2: dig command: showing Internet DNS root servers

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

Command: *dig ns uni-bamberg.de*

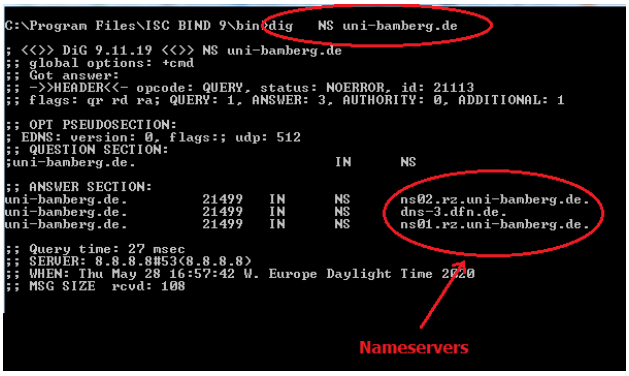

Figure 1.3: dig command: nameservers for a specific domain

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.

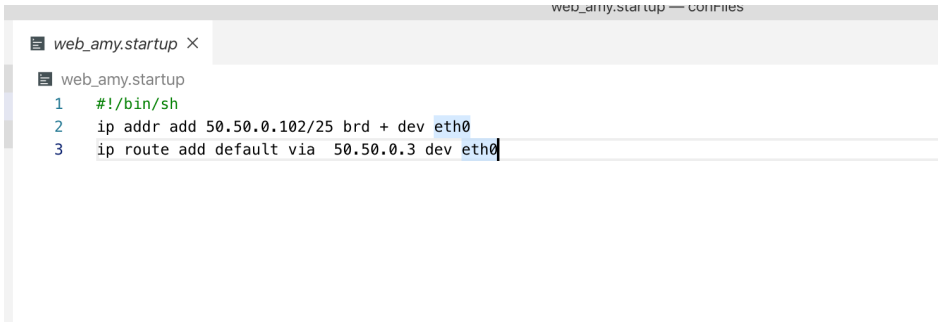

Figure 2.2: Example for new added nodes setup file with configuring it's route

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

Corfile 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 us 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. This 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 fined 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.

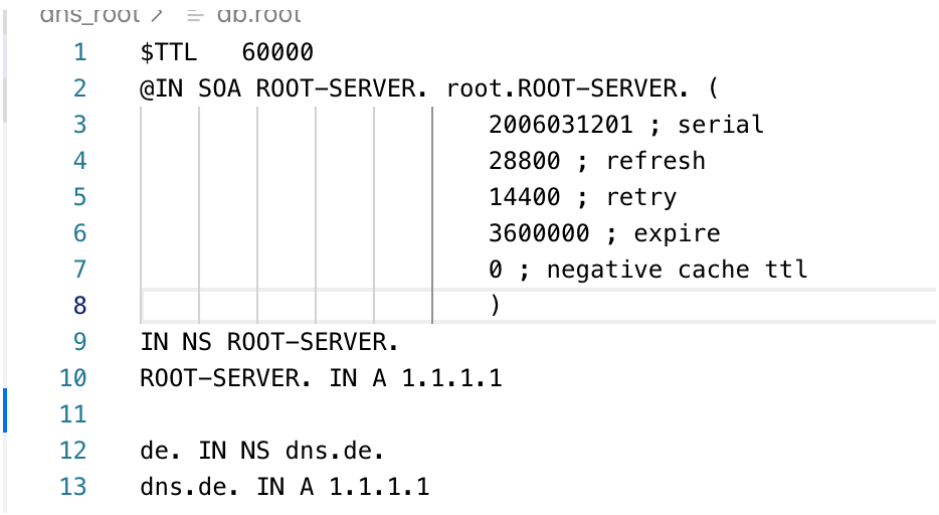

Figure 2.4: db.root file

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.

```
$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 40.40.0.100
```

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.

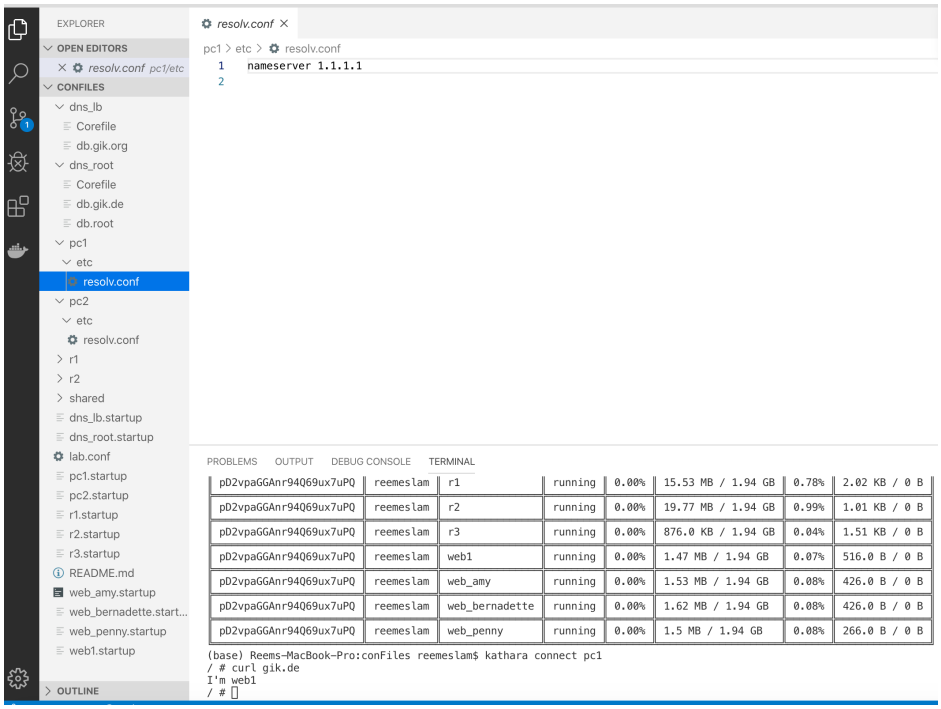

Figure 2.6: Adding nameserver to pc1 and testing that with curl gik.de

2.5 Capturing the name resolution on CD A with wireshark

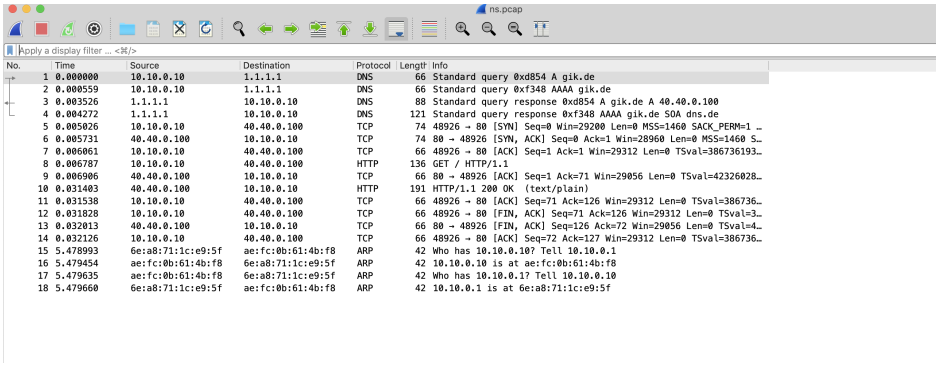

Figure 2.7: Result of Capturing pc1 curl gik.de on wireshark

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

The is DNS zone file, and it will have gik.org name entry.The gik.org points to webservers web bernadette,web amy, and web penny which can we 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.

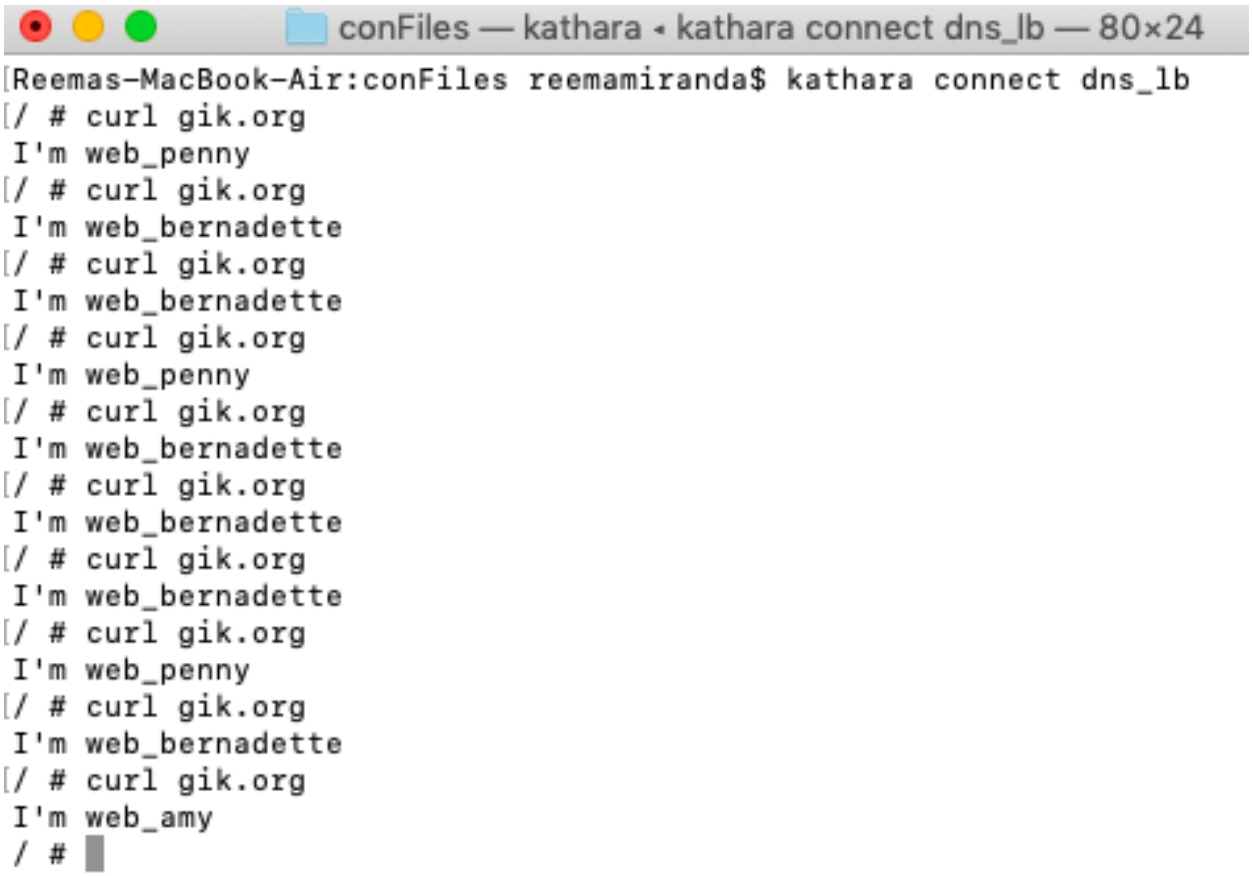

Figure 2.14: Curl gik.org execution

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

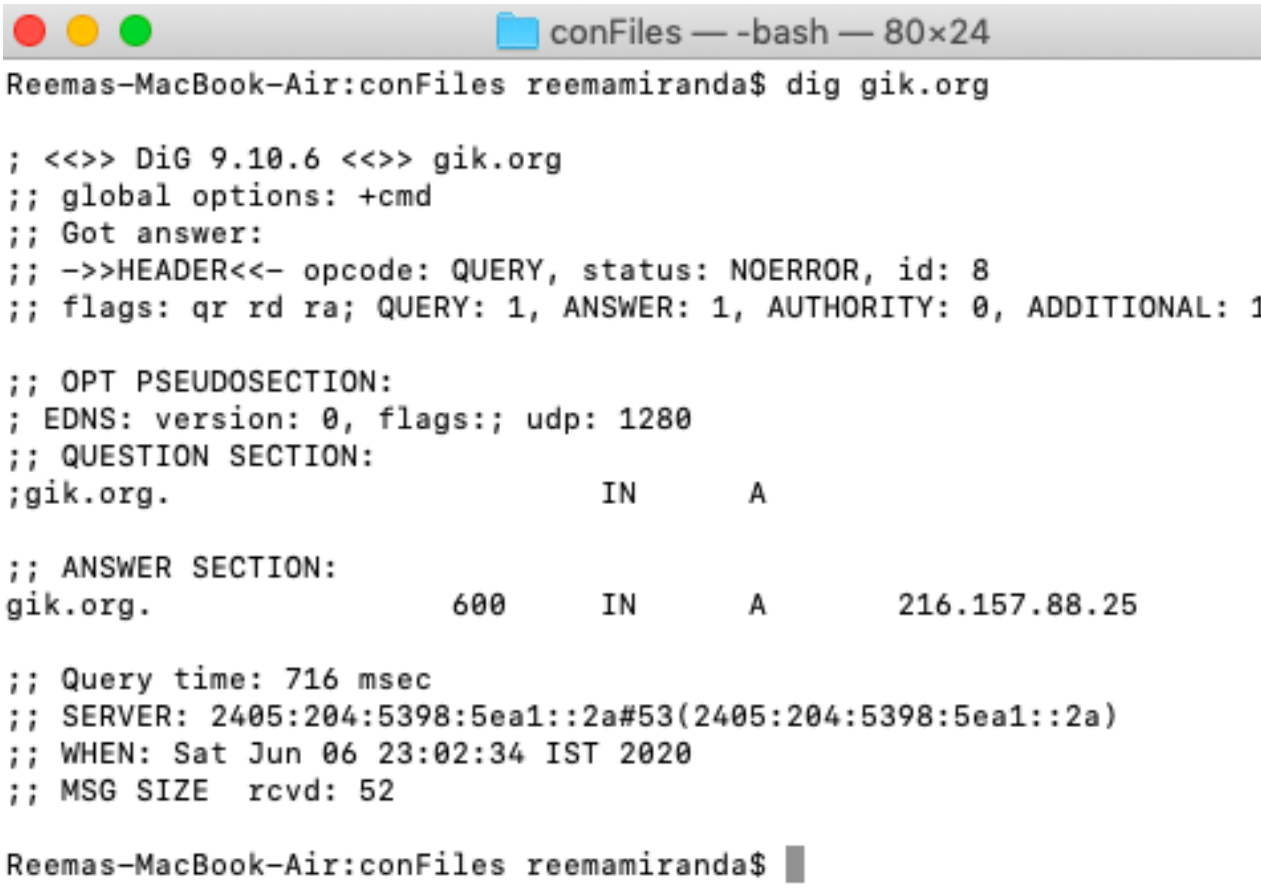

Figure 2.15: Dig gik.org execution

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.

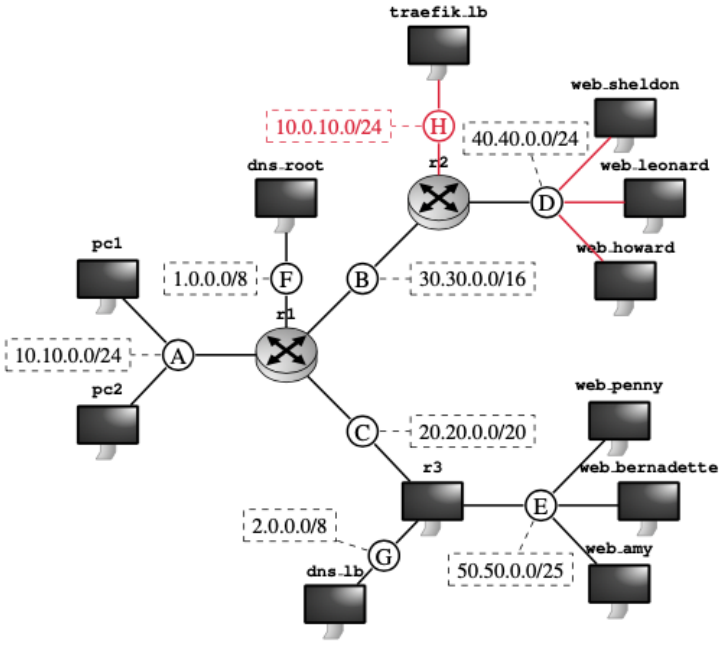

Figure 3.1: Experiment configuration with traefik

3.1.1 Startup files

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

web_sheldon.startup 96 Bytes

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

Figure 3.2: web_Sheldon startup

web_howard.startup 96 Bytes

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

Figure 3.3: web_Howard startup

web_leonard.startup 96 Bytes

```
#!/bin/sh
ip addr add 40.40.0.101/24 brd + dev eth0
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
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	dns_lb	running	0.00%	15.18 MB / 1.94 GB	0.76%	696.0 B / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	dns_root	running	0.00%	23.14 MB / 1.94 GB	1.16%	516.0 B / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	pc1	running	0.00%	2.01 MB / 1.94 GB	0.10%	836.0 B / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	pc2	running	0.00%	692.0 KB / 1.94 GB	0.03%	836.0 B / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	r1	running	0.00%	24.61 MB / 1.94 GB	1.24%	3.04 KB / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	r2	running	0.00%	28.45 MB / 1.94 GB	1.43%	2.31 KB / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	r3	running	0.00%	1.85 MB / 1.94 GB	0.09%	1.78 KB / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	traefik_lb	running	0.00%	880.0 KB / 1.94 GB	0.04%	516.0 B / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	web_amy	running	0.00%	1.64 MB / 1.94 GB	0.08%	266.0 B / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	web_bernadette	running	0.00%	1.41 MB / 1.94 GB	0.07%	426.0 B / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	web_howard	running	0.00%	1.32 MB / 1.94 GB	0.07%	516.0 B / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	web_leonard	running	0.00%	1.43 MB / 1.94 GB	0.07%	516.0 B / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	web_penny	running	0.00%	1.39 MB / 1.94 GB	0.07%	516.0 B / 0 B
lwYSn1k3zWPmwjBxiqkjLg	shivasharanreddyreddy	web_sheldon	running	0.00%	5.45 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.

gik.toml

```
## 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.