

DNS LOAD BALANCING

M2 DELIVERABLES REPORT

TEAM MEMBERS:

Lokeshwar Reddy Nandanapalli 2299460

Gujja Shanthan Rao 2300228

DETAILED CONTRIBUTIONS:

Lokeshwar Reddy Nandanapalli 2299460

- Creation of domain
- Creation of a sample website
- Configuring the created ec2 instance to run the DNS server
- Implementing load balancing DNS Server (Round Robin algorithm)
- Implementing load listener to update load data file and
- Implementing Recovery Mechanism.
- Implementing File Watcher for Config file
- Creating hostnames for custom nameservers
- Testing the DNS server with nslookup

Gujja Shanthan Rao 2300228

- Creation of three Web Server instances
- Installing Web servers on the created EC2 instances
- Creation of and ec2 instance for DNS Server
- Implementing Logging and monitoring logs
- Implementing Geo Location Based Load Balancing.
- Implementing Web Server Load Updation to DNS
- Implementing File watchers for Geo data and Load data.
- Implementing Configuration file Checking to maintain integrity.
- Running the DNS server

GOAL OF THE PROJECT:

To distribute load across available web servers by creating and deploying custom nameserver for a domain.

To efficiently utilize the architecture of DNS protocol to balance load using different algorithms (Round Robin, Geo Location, Load Based)

To ensure that the load balancing works efficiently but implementing many algorithms that can be configured based on the choice of the administrator.

DESIGN:

1. Creation of domain (By Lokeshwar Reddy Nandanapalli)

Successfully created lokeshwarreddyshanthanrao.com domain with spaceship registrar

2. Creation of three Web Server instances (By Shanthan Rao Gujja)

Created an ec2 instance in AWS using Amazon linux for Web Server 1 in N Virginia.

Allowed http and https traffic in the firewall of the security group so that the web server can serve pages to the internet.

Allowed http and https traffic from Anywhere (0.0.0.0/0)

Similarly created ec2 instances named Web Server 2 in California and Web Server 3 in Oregon.

3. Installing Web servers on the created EC2 instances (By Shanthan Rao Gujja)

Installed apache server on Web Server 1 using the command
`sudo yum install httpd`

The installation completed successfully.

```
[ec2-user@ip-172-31-26-206 ~]$ sudo systemctl start httpd
[ec2-user@ip-172-31-26-206 ~]$ sudo systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
[ec2-user@ip-172-31-26-206 ~]$
```

Started the web server. It now runs on port 80 by default

`sudo systemctl start httpd`

Enabled it to run on every boot by using the command

```
sudo systemctl enable httpd
```

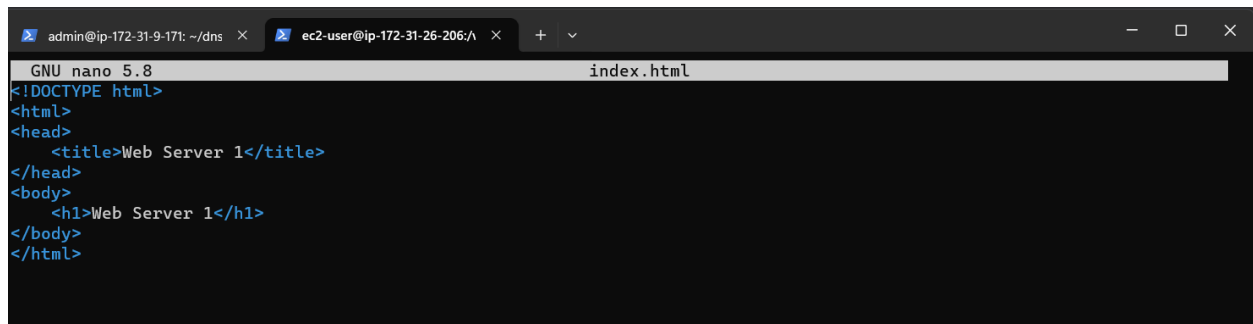
4. Creation of and ec2 instance for DNS Server (By Shanthan Rao Gujja)

An ec2 debian instance is created for the installation of DNS server.

Traffic on ports 53 (DNS TCP), 53 (DNS UDP) is allowed in the firewall for the internet to communicate with the authoritative DNS server that will be run on this instance.

5. Creation of a sample website (By Lokeshwar Reddy Nandanapalli)

Navigate to the directory /var/www/html on the instances on which the web server is installed.



```
admin@ip-172-31-9-171: ~/dns  x  ec2-user@ip-172-31-26-206:/^  +  v
GNU nano 5.8                                index.html
<!DOCTYPE html>
<html>
<head>
  <title>Web Server 1</title>
</head>
<body>
  <h1>Web Server 1</h1>
</body>
</html>
```

Create a file index.html, that displays the name of the web server for the client. The sample page for web server 1 is shown.

Similarly Sample web pages are created for all web server instances.



Web Server 1



Web Server 2



All web servers are working as expected.

6. Configuring the created ec2 instance to run the DNS server (By Lokeshwar Reddy Nandanapalli)

```
admin@ip-172-31-9-171: ~/dn  X + v
admin@ip-172-31-9-171:~/dnserver$ sudo systemctl stop systemd-resolved.service
admin@ip-172-31-9-171:~/dnserver$ sudo systemctl disable systemd-resolved.service
sudo: unable to resolve host ip-172-31-9-171: Name or service not known
Removed "/etc/systemd/system/sysinit.target.wants/systemd-resolved.service".
Removed "/etc/systemd/system/dbus-org.freedesktop.resolve1.service".
admin@ip-172-31-9-171:~/dnserver$ |
```

Turn off `systemd-resolved.service` that runs on port 53, so that our load balancing server can use these ports, else a conflict occurs that another program is already using these ports.

Round Robin:

To maintain an in memory data structure that can function like a queue.

Example: Linked List, Doubly Linked List, Dynamic Array, etc.

Load IP addresses into this data structure in the beginning, create a file watcher to monitor the config file, load IP addresses again after any update.

Remove an ip from the beginning, return it as the IP address following the DNS protocol, Append it again to the end of the list to create a round robin mechanism

Geo Location:

Store the latitude and longitude of each ip address in a config file.

Client IP is available to the nameserver, distance is calculated by obtaining the latitude and longitude of the client using IPstack IP Geolocation API.

Based on the distance the IP address of the nearest server among the available web servers is returned.

Load Based:

Create a service on Web Servers to send load to the dns server, the load is updated in regular intervals in a config file on the DNS server.

The IP address with the minimum load is returned.

Configuration files:

Configuration files are provided to configure the DNS server.

IMPLEMENTATION:

1. Implementing minor utility functions (By Lokeshwar Reddy Nandanapalli)

```
D = DomainName('lokeshwarreddyshanthanrao.com.')
IP = None
TTL = 60
```

The domain for which this server is authoritative, and the TTL of the DNS record returned by this server.

The IP is left as none, because this is allocated based on a round robin algorithm which will be implemented in the code.

```
def is_web_server_up(ip):
    try:
        response = requests.get(f"http://{ip}:80")
        return response.status_code == 200
    except Exception:
        return False
```

A simple function that tests the health of the web server using the requests library. Check if the status is 200.

```
def load_ip_addresses(ips):
    ips_temp = []
    for ip in ips:
        if is_web_server_up(ip):
            ips_temp.append(ip)
    return ips_temp
```

A simple function which loads the ip addresses. It loads an ip only if the server is up.

```
dnsserver > ! config.yml
1  ip_addresses:
2    - 52.2.87.66
3    - 54.151.85.122
4    - 54.218.75.250
5
6  dns_ip_address: 3.145.162.236
7
8  # round or geo or load
9  algorithm: load
10
11 geo_data_file: geo_data.json
12 load_data_file: load.json
13
14 # directory of all files - the home of this project
15 home_dir: /home/admin/dnsserver
16
17 geo_api_key: 68afca1e724b704e64db96490fdce2c8
```

The config data and ip addresses are stored in the config.yml file.

2. Implementing Logging (By Shanthan Rao Gujja)

```
def dns_log(message, severity=0):
    log_level = logging.CRITICAL if severity == -1 else (logging.INFO if severity
    current_datetime = datetime.datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.%f')
    log_message = f"{current_datetime} - {message}"
    if not logging.getLogger().hasHandlers():
        logging.basicConfig(level=logging.INFO)
        file_handler = logging.FileHandler('dnsserver.log')
        file_handler.setLevel(logging.INFO)
        logging.getLogger().addHandler(file_handler)
    logging.log(log_level, log_message)
```

Implemented logs with the help of a logging module in python, a function that accepts a log message and writes to the log.

3. Implementing Recovery Mechanism (By Lokeshwar Reddy Nandanapalli)

```
def is_already_running():
    lock_file_path = "dns_server.lock"
    if os.path.isfile(lock_file_path):
        with open(lock_file_path, "r") as lock_file:
            pid = lock_file.read().strip()
            if pid and psutil.pid_exists(int(pid)):
                return True
            else:
                ps.remove(lock_file_path)
    with open(lock_file_path, "w") as lock_file:
        lock_file.write(str(os.getpid()))
    return False
```

Checking if an instance is already running. If yes, the script exits. Else it runs and creates a lock file to mark that an instance is running. It's periodically checked by a linux system service which is installed by the script on every run. Then the service restarts it if it crashes. The application provides a command line option to stop the server by removing the service.

4. Implementing load balancing core logic (By Lokeshwar Reddy Nandanaplalli)

```
def next_ip_address():
    global IP
    global DNS_IP
    global ip_addresses
    global records
    lock.acquire()
    if config["algorithm"] == "geo":
        try:
            IP = get_nearest_ip()
        except:
            IP = config["ip_addresses"][0]
    elif config["algorithm"] == "load":
        try:
            IP = get_lowest_load_ip()
        except:
            IP = config["ip_addresses"][0]
    else:
        temp = ip_addresses.pop(0)
        IP = temp
        ip_addresses.append(temp)
    records = {
        D: [A(IP), AAAA((0,) * 16), MX(D.mail), soa_record] + ns_records,
        D.ns3: [A(DNS_IP)],
        D.ns4: [A(DNS_IP)],
        D.mail: [A(DNS_IP)],
        D.lokesh: [CNAME(D)],
    }
    lock.release()
```

The core function for load balancing which implements round robin and also handles geo location and load based approaches based on the configuration. It takes the first ip address from the round robin list. Assigns it to the IP variable, and then adds it to the end of the round robin list.

It crafts a DNS record that will be returned by the DNS server.

5. Implementing Geo Location based load Balancing. (By Shanthan Rao Gujja)

```
def calculate_distance(lat1, lon1, lat2, lon2):
    R = 6371.0
    dlat = radians(lat2 - lat1)
    dlon = radians(lon2 - lon1)
    a = sin(dlat / 2) ** 2 + cos(radians(lat1)) * cos(radians(lat2)) * sin(dlon / 2) ** 2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    distance = R * c
    return distance

def get_nearest_ip():
    global geo_data
    global client_ip
    if not client_ip:
        print("Error: Client IP not available.")
        return None
    client_geo_info = requests.get(f"http://api.ipstack.com/{client_ip}?access_key={config['geo_api_key']}").json()
    client_lat = client_geo_info['latitude']
    client_lon = client_geo_info['longitude']
    nearest_ip = None
    min_distance = float('inf')
    for ip, geo_info in geo_data.items():
        lat = geo_info['latitude']
        lon = geo_info['longitude']
        distance = calculate_distance(client_lat, client_lon, lat, lon)
        if distance < min_distance:
            min_distance = distance
            nearest_ip = ip
    return nearest_ip
```

Based on the geometry of earth, and the latitude and longitude information of an IP address, the distance between client and web server is calculated. Geolocation information of IP addresses is obtained using IPStack api.

```
{
    "52.2.87.66": {"latitude": 38.8048, "longitude": -77.0469},
    "54.151.85.122": {"latitude": 37.7749, "longitude": -122.4194},
    "54.218.75.250": {"latitude": 45.5234, "longitude": -122.6762}
}
```

Geo_data.json, each ip in the config must have a latitude and longitude associated with it.

6. Implementing Server Load based load balancing. (By Lokeshwar Reddy Nandanaplli)


```
{
  "52.2.87.66": 0.5,
  "54.151.85.122": 0.7,
  "54.218.75.250": 0.2
}
```

Load.json

Every ip must have a load value, 0 initially and then the service which runs on port 8080 accepting load from web servers updates the load.

Sending The load (By Shanthan Rao Gujja)

```
def get_cpu_usage():
    return psutil.cpu_percent()

while True:
    try:
        cpu_usage = get_cpu_usage()
        load_data = {"ip": config["name"], "load": cpu_usage}
        response = requests.post(f"http://{config['dns_server_ip']}:8080", data=load_data)
        if response.status_code == 200:
            print("Load data sent successfully")
        else:
            print(f"Failed to send load data. Status code: {response.status_code}")
            time.sleep(5)
    except KeyboardInterrupt:
        break
    except Exception:
        pass
```

Listening Load (By Lokeshwar Reddy Nandanapalli)

```

class LoadRequestHandler(SimpleHTTPRequestHandler):
    def do_POST(self):
        global load_data
        global lock
        content_length = int(self.headers['Content-Length'])
        post_data = self.rfile.read(content_length)
        params = parse_qs(post_data.decode('utf-8'))

        ip = params.get('ip', [''])[0]
        load = float(params.get('load', [0])[0])

        lock.acquire()

        load_data[ip] = load

        with open("load.json", "w") as load_file:
            json.dump(load_data, load_file)
        lock.release()
        self.send_response(200)
        self.end_headers()

```

Keeps listening for load and updates load.json

Balancing load from available load data.

```

{
    "52.2.87.66": 0.5,
    "54.151.85.122": 0.7,
    "54.218.75.250": 0.2
}

```

load.json

```

def get_lowest_load_ip():
    global load_data
    if not load_data:
        print("Error: Load data not available.")
        dns_log("Error: Load data not available.")
        return None
    lowest_load_ip = min(load_data, key=load_data.get)
    return lowest_load_ip

```

```

last_modified_time = 0

def file_watcher():
    global last_modified_time
    global config
    global ip_addresses
    while True:
        try:
            file_stat = os.stat('config.yml')
            current_modified_time = file_stat.st_mtime
            if current_modified_time != last_modified_time:
                last_modified_time = current_modified_time
                lock.acquire()
                with open("config.yml", "r") as f:
                    config = yaml.safe_load(f)
                    ip_addresses = load_ip_addresses(config["ip_addresses"])
                lock.release()
        except FileNotFoundError:
            pass
        time.sleep(1)

watcher_thread = threading.Thread(target=file_watcher)
watcher_thread.daemon = True
watcher_thread.start()

```

Also a file watcher is implemented to load the updated ip address in the config file if the file changes. This runs as a separate daemon thread.

Similar watchers are implemented for geo data and load data files.

```

def dns_response(data):
    request = DNSRecord.parse(data)
    reply = DNSRecord(DNSHeader(id=request.header.id, qr=1, aa=1, ra=1), q=request.q)

    qname = request.q.qname
    qn = str(qname)
    qtype = request.q.qtype
    qt = QTYPE[qtype]

    if qn == D or qn.endswith('.') + D):
        for name, rrs in records.items():
            if name == qn:
                for rdata in rrs:
                    rqt = rdata.__class__.__name__
                    if qt in ['*', rqt]:
                        reply.add_answer(RR(rname=qname, rtype=getattr(QTYPE, rqt), rclass=1, ttl=TTL, rdata=rdata))

        for rdata in ns_records:
            reply.add_ar(RR(rname=D, rtype=QTYPE.NS, rclass=1, ttl=TTL, rdata=rdata))

        reply.add_auth(RR(rname=D, rtype=QTYPE.SOA, rclass=1, ttl=TTL, rdata=soa_record))

    print("---- Reply:\n", reply)

    return reply.pack()

```

The DNS response is sent by this function.

7. Checking configuration file for Integrity

(By Shanthan Rao Gujja)

```

def check_config_files():
    dns_log("Configuration file check started.")
    config_file_path = 'config.yml'
    geo_data_file_path = 'geo_data.json'
    load_data_file_path = 'load.json'

    if not os.path.exists(config_file_path):
        print(f"Error: Config file '{config_file_path}' not found.")
        dns_log("Error: Config file '%s' not found." % config_file_path)
        return False

    if not os.path.exists(geo_data_file_path):
        print(f"Error: Geo data file '{geo_data_file_path}' not found.")
        dns_log("Error: Geo data file '%s' not found." % geo_data_file_path)
        return False

    if not os.path.exists(load_data_file_path):
        print(f"Error: Load data file '{load_data_file_path}' not found.")
        dns_log("Error: Load data file '%s' not found." % load_data_file_path)
        return False

```

8. Running the DNS server (By Shanthan Rao Gujja)

```
admin@ip-172-31-9-171:~/dnsserver$ sudo python3 dnsserver.py --port 53 --udp --tcp
sudo: unable to resolve host ip-172-31-9-171: Name or service not known
['54.193.47.185', '18.236.135.114', '174.129.162.236'] {'lokeswarreddyshanthanrao.com.': [174.129.162.236, :, 10 mail.lokeswarreddyshanthanrao.com., ns1.lokeswarreddyshanthanrao.com. andrei.lokeswarreddyshanthanrao.com. 201307231 3600 10800 86400 3600, ns1.lokeswarreddyshanthanrao.com., ns2.lokeswarreddyshanthanrao.com.], 'ns1.lokeswarreddyshanthanrao.com.': [174.129.162.236], 'ns2.lokeswarreddyshanthanrao.com.': [174.129.162.236], 'mail.lokeswarreddyshanthanrao.com.': [174.129.162.236], 'andrei.lokeswarreddyshanthanrao.com.': [lokeswarreddyshanthanrao.com.]}
Starting nameserver...
UDP server loop running in thread: Thread-2 (serve_forever)
TCP server loop running in thread: Thread-3 (serve_forever)
```

The DNS server is run using the following command to test.

```
admin@ip-172-31-9-171:~/dnsserver$ sudo nohup python3 dnsserver.py --port 53 --udp --tcp &
[1] 2902
admin@ip-172-31-9-171:~/dnsserver$ sudo: unable to resolve host ip-172-31-9-171: Name or service not known
nohup: ignoring input and appending output to 'nohup.out'

admin@ip-172-31-9-171:~/dnsserver$ |
```

The DNS server is run in the background permanently using the following command.

9. Creating hostnames for custom nameservers (By Lokeshwar Reddy Nandanapllia)

Personal nameservers 2 records	
Host	IP Address
ns3	3144.25.232
ns4	3144.25.232
+ Add record	

10. Testing the DNS Server (By Lokeshwar Reddy Nandanapalli)

Round Robin

```

PS C:\Users\nlred> nslookup lokeshwarreddyshanthanrao.com
Server: cdns01.comcast.net
Address: 75.75.75.75

Non-authoritative answer:
Name:    lokeshwarreddyshanthanrao.com
Addresses:  ::
          18.236.135.114

PS C:\Users\nlred> nslookup lokeshwarreddyshanthanrao.com
Server: cdns01.comcast.net
Address: 75.75.75.75

Non-authoritative answer:
Name:    lokeshwarreddyshanthanrao.com
Addresses:  ::
          54.193.47.185

PS C:\Users\nlred> nslookup lokeshwarreddyshanthanrao.com
Server: cdns01.comcast.net
Address: 75.75.75.75

Non-authoritative answer:
DNS request timed out.
    timeout was 2 seconds.
Name:    lokeshwarreddyshanthanrao.com
Address: 174.129.162.236

```

The nslookup command displays round robin DNS load balancing. Since IP addresses are returned using a round robin algorithm.

Geo Location

```

PS C:\Users\nlred> nslookup lokeshwarreddyshanthanrao.com
Server: cdns01.comcast.net
Address: 75.75.75.75

Non-authoritative answer:
Name:    lokeshwarreddyshanthanrao.com
Addresses:  ::
          52.2.87.66

PS C:\Users\nlred> |

```

Load Based

```
PS C:\Users\nlred> nslookup lokeshwarreddyshanthanrao.com
Server:  cdns01.comcast.net
Address:  75.75.75.75

Non-authoritative answer:
Name:     lokeshwarreddyshanthanrao.com
Addresses:  ::
           54.151.85.122

PS C:\Users\nlred> nslookup lokeshwarreddyshanthanrao.com
Server:  cdns01.comcast.net
Address:  75.75.75.75

Non-authoritative answer:
Name:     lokeshwarreddyshanthanrao.com
Addresses:  ::
           54.218.75.250
```

LIMITATIONS:

1. The server does not implement any mechanism to detect multiple DNS requests from the same host on the internet and return the same IP on consecutive requests in a short time.
2. The DNS server doesn't have a mechanism to defend itself against brute force attacks.
3. The server uses an in memory data structure which may not be robust, since recovery from failure is very tough.
4. There is no fail-safe mechanism if the whole server crashes and never boots again

INSTRUCTIONS:

Place the server_load.py script and its configuration file on the web server and run it.

Place the dnsserver folder in the linux machine. Configure the path of the folder as the homedir in the config.yml file and make the necessary changes to ip addresses in the

config and to their geo data. Ensure that port 53 and 8080 are not used by any other processes.

Run the dns server with the command
`sudo python3 dnsserver.py -port 53 -tcp -udp`

To stop
`sudo python3 dnsserver.py stop`

CODE OBTAINED:

Took an existing DNS server from

<https://gist.github.com/pklaus/b5a7876d4d2cf7271873>

This is a simple DNS server in python that returns a hard coded IP address on a dns request.

It is built using dnslib, a simple python library that has some useful predefined classes that are implemented based on the DNS protocol RFC specifications.