FINAL PROJECT
PEMROGRAMAN JARINGAN (D)

Kelompok 2

05111840000057 Maisie Chiara Salsabila

05111840000130 I Gusti Agung Chintya

05111840000163 Putu Putri Natih Devayanti

Dosen Pengampu:

Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Departemen Teknik Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember (ITS)

Surabaya

2021

SOAL:

1. *Reverse Proxy*
2. *Load Balancing Reverse Proxy*

KONTRIBUSI:

- 05111840000057 Maisie Chiara Salsabila
    - *Load Balancing Threaded*
- 05111840000130 I Gusti Agung Chintya Prema Dewi
    - *Reverse Proxy*
- 05111840000163 Putu Putri Natih Devayanti
    - *Load Balancing Asynchronous*

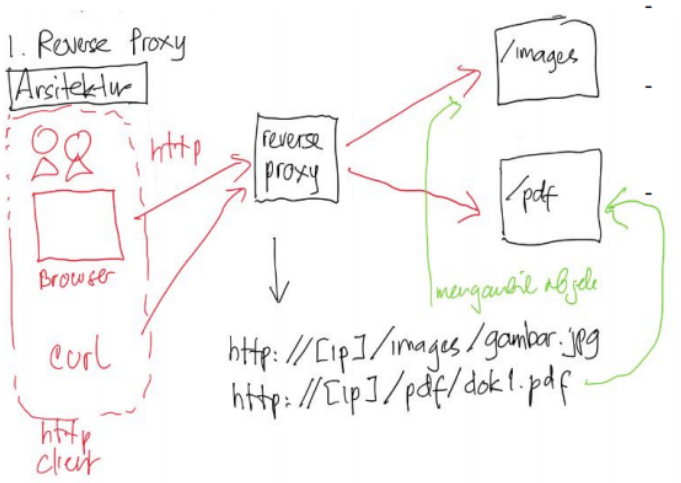
Link: Github

PENDAHULUAN:

Reverse Proxy merupakan fitur atau modul pada web server yang berfungsi untuk melakukan *port forwarding* suatu *request* dari *public request* ke sistem. *Reverse proxy* juga memiliki kemampuan *content coaching* atau penyimpanan data sementara sehingga ketika dilakukan *request* ulang maka tidak perlu mengambil ulang dari *database*. Dengan begitu web server akan mampu mengerjakan pekerjaan-pekerjaan *proxy* secara umum hingga pekerjaan sebagai *upstream proxy* yang sering difungsikan sebagai pengganti NAT. *Reverse Proxy* juga mampu digunakan untuk menghemat IP Address.

1. Reverse Proxy

1.1. Implementasikan arsitektur berikut ini dengan memodifikasi program yang berkaitan pembahasan http server



1.2. Melakukan pemrosesan data melalui *socket proxy* sebelum dibawa ke *reverse proxy*. Socket proxy berfungsi sebagai pemecah arah menjadi 2, yaitu ke folder image dan pdf.

1.2.1. *Class socket_proxy.py*

socket_proxy.py × reverse_proxy.py

FinalProjectProgjar > reverse_proxy_fp > socket_proxy.py > ...

```
from reverse_proxy import ReverseProxy

reverseProxy = ReverseProxy()

class ProcessTheClient(threading.Thread):
    def __init__(self, connection, address):
        self.connection = connection
        self.address = address
        threading.Thread.__init__(self)

    def run(self):
        rcv=""
        while True:
            try:
                data = self.connection.recv(8192)
                data = data.decode()
                self.destination_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                if data:
                    forward_response = reverseProxy.proses(data)
                    self.destination_sock.connect(forward_response['server'])
                    self.destination_sock.sendall(forward_response['request'].encode())
                    while(True):
                        data_balasan = self.destination_sock.recv(32)
                        if(data_balasan == ''):
                            break
                        self.connection.sendall(data_balasan)
                    # logging.warning(data)
                    # logging.warning(data_balasan)
                else:
                    break
            except OSError as e:
                pass
        self.connection.close()
```

1.3. *Reverse proxy* dapat menerjemahkan path pada sebuah URL untuk diteruskan ke *backend server* yang sesuai.

1.3.1. *Class reverse_proxy.py*

```python
class ReverseProxy:
    def __init__(self):
        self.url_dict = {}
        self.url_dict['/images/']=("localhost", 8889)
        self.url_dict['/pdf/']=("localhost", 9000)
        self.default_server = ("localhost", 8888)

    def proses(self,data):

        forward_response = {}
        requests = data.split("\r\n")
        baris = requests[0]

        all_headers = [n for n in requests[1:] if n!='']
        j = baris.split(" ")
        method=j[0].upper().strip()
        url_address = j[1].strip()
        if(url_address[-1] != '/'):
            data = data.replace(url_address, url_address+'/')
            url_address += '/'

        for url, server in self.url_dict.items() :
            re_match = re.match(url, url_address)
            if re_match :
                forward_response['server'] = server
                forward_response['request'] = data.replace(url, '/')

        if "server" not in forward_response :
            forward_response['server'] = self.default_server
            forward_response['request'] = data

        return forward_response
```

1.4. Pada gambar disamping, jika *reverse proxy* menerima *reques*t dalam *path* /images, maka objek akan di*retrieve* dari *backend server* yang melayani /images.

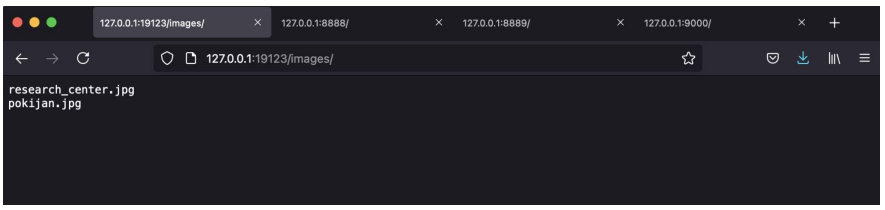
1.4.1. *Class Reverse Proxy*

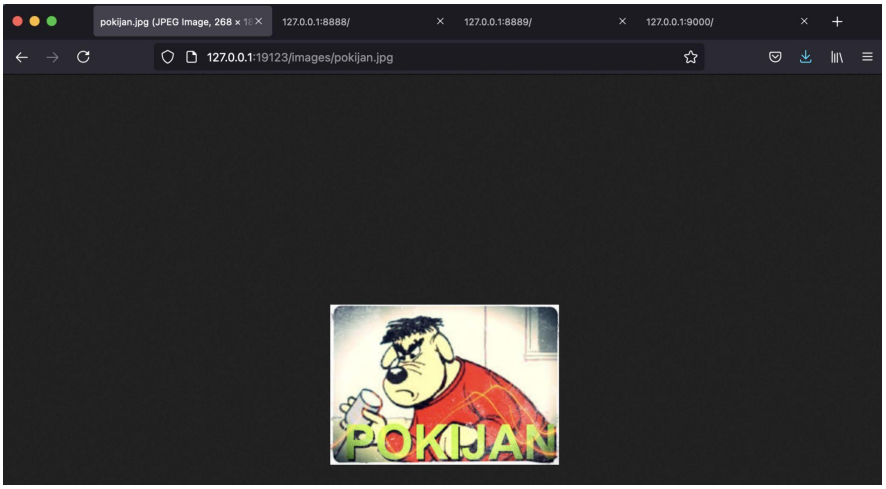
```python
class ReverseProxy:
    def __init__(self):
        self.url_dict = {}
        self.url_dict['/images/']=("localhost", 8889)
        self.url_dict['/pdf/']=("localhost", 9000)
        self.default_server = ("localhost", 8888)

    def proses(self,data):

        forward_response = {}
        requests = data.split("\r\n")
        baris = requests[0]

        all_headers = [n for n in requests[1:] if n!='']
        j = baris.split(" ")
        method=j[0].upper().strip()
        url_address = j[1].strip()
        if(url_address[-1] != '/'):
            data = data.replace(url_address, url_address+'/')
            url_address += '/'

        for url, server in self.url_dict.items() :
            re_match = re.match(url, url_address)
            if re_match :
                forward_response['server'] = server
                forward_response['request'] = data.replace(url, '/')

        if "server" not in forward_response :
            forward_response['server'] = self.default_server
            forward_response['request'] = data

        return forward_response
```

1.5. Gunakan *http client* berupa

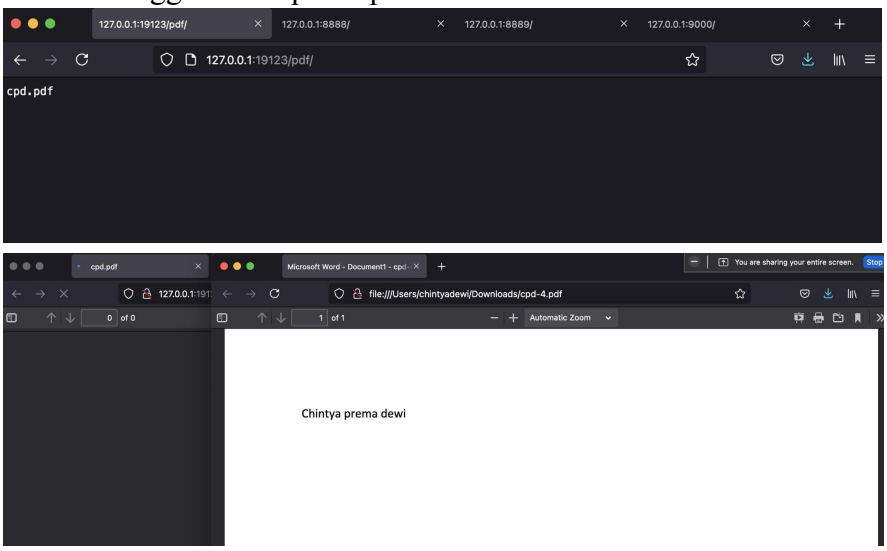
1.5.1. *Web browser* (chrome/firefox)

1.5.1.1. Mengakses suatu gambar yang ada pada folder images menggunakan ip dan port utama



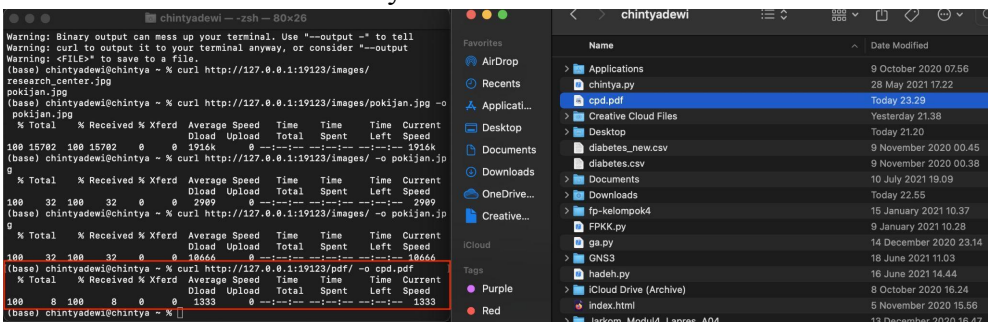


1.5.1.2. Mengakses suatu pdf dokumen yang ada pada folder pdf menggunakan ip dan port utama

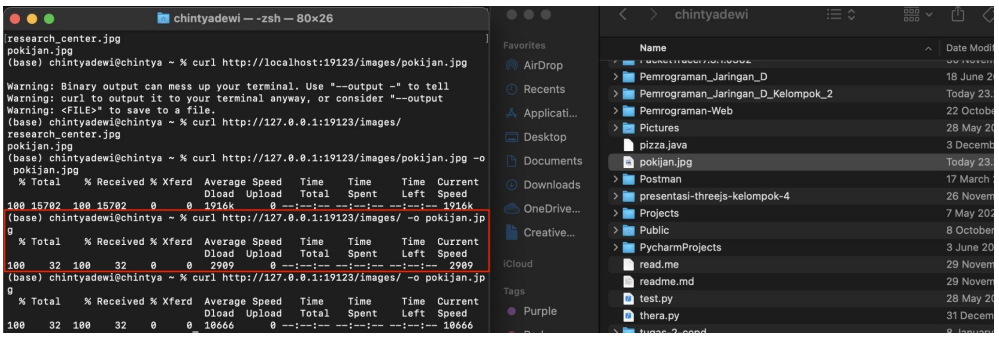


1.5.2. *Curl*

1.5.2.1. Lakukan download file cpd.pdf. Maka file pdf ini akan muncul di location folder chintya.

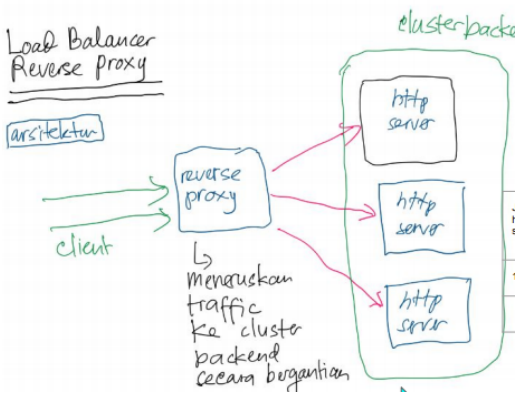


1.5.2.2. Lakukan download file pokijan.jpg. Maka file jpg ini akan muncul di *location* folder chintya.



2. Load Balancing Reverse Proxy

2.1. Arsitektur dan konfigurasi (konfigurasi ip port dsb)



2.2. Implementasikan *load balancing reverse proxy* berikut ini, dengan model

2.2.1. *Threaded*

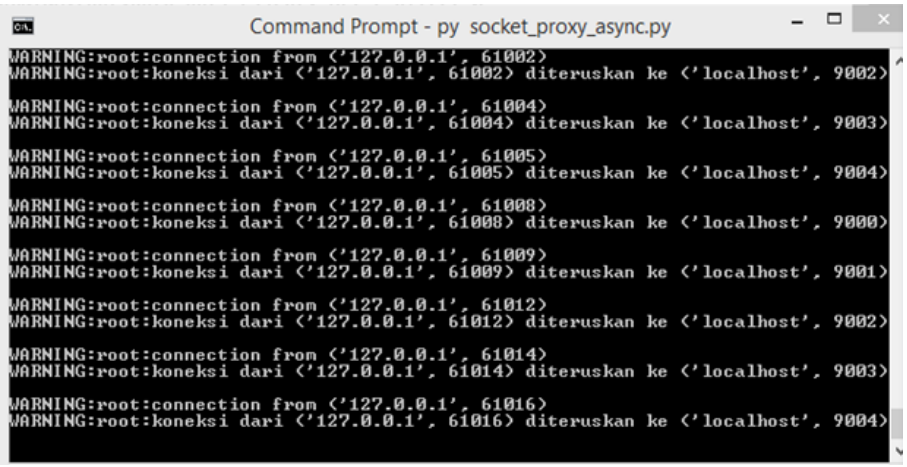
2.2.2. *Asynchronous* (sudah ada dalam contoh)

2.3. *Request* yang diterima *reverse proxy*, akan diteruskan ke *cluster backend* dengan cara *round robin (Round robin* → bergantian dalam proporsi yang sama)

2.3.1. *Load Balancer* bertugas untuk mendistribusikan request ke *backend* yang didefinisikan pada *Class LoadBalancer*. Pada *class* tersebut terdapat fungsi *get_server* yang akan membuat *backend* dipilih secara *round robin* sehingga setiap request akan dilayani oleh *backend* secara bergantian.

```python
class LoadBalancer:
    def __init__(self):
        self.server_list = []
        self.server_list.append(("localhost", 9000))
        self.server_list.append(("localhost", 9001))
        self.server_list.append(("localhost", 9002))
        self.server_list.append(("localhost", 9003))
        self.server_list.append(("localhost", 9004))
        self.counter = 0
    def get_server(self):
        server = self.server_list[self.counter]
        self.counter += 1
        if self.counter >= len(self.server_list):
            self.counter = 0
        return server
```

2.3.2. Proses log menunjukkan bahwa setiap *request* akan dilayani di *backend* secara bergantian



2.4. Jalankan *performance test* dengan menggunakan *apache benchmark* (ab) dengan *target server* pada *reverse proxy*

2.4.1. Bandingkan performa 1 *http server* dengan *multi http server*

2.4.2. Gunakan 10000 jumlah *request*

2.4.2.1. *Concurrency* 2,5,10

2.4.2.2. Jumlah b*ackend server* : 1,2,3,4,5

2.5. Hasil

Link Screenshot

2.5.1. Asynchronous

Jumlah http server	*Concurrency*	Jumlah *complete request*	Non-2xx response	Jumlah *request per second*	*Time per request (mean across)* [ms]
1	2	10000	8001	280.57	3.564
1	5	10000	8000	205.12	4.872
1	10	10000	8000	12.43	80.478
2	2	10000	6000	207.56	4.818
2	5	10000	6000	210.12	4.759
2	10	10000	6000	12.43	80.471
3	2	10000	4002	185.90	5.379
3	5	10000	4000	171.37	5.835
3	10	10000	4000	12.39	80.734
4	2	10000	2004	166.84	5.994
4	5	10000	2000	174.01	5.747
4	10	10000	2000	12.49	80.035
5	2	10000	6	148.18	6.748

5	5	10000	0	129.90	7.698
5	10	10000	0	12.57	79.542

2.5.2. Threaded

Jumlah http server	*Concurrency*	Jumlah *complete request*	Non-2xx response	Jumlah *request per second*	*Time per request (mean across)* [ms]
1	2	10000	0	353.40	2.830
1	5	10000	0	561.87	1.780
1	10	10000	0	78.24	12.782
2	2	10000	0	130.22	7.679
2	5	10000	0	153.77	32.516
2	10	10000	0	14.18	70.456
3	2	10000	0	120.92	8.270
3	5	10000	0	133.09	7.514
3	10	10000	0	14.48	69.054
4	2	10000	0	114.26	8.752
4	5	10000	0	81.45	12.277
4	10	10000	0	14.37	69.596
5	2	10000	0	118.97	8.406
5	5	10000	0	82.90	12.063
5	10	10000	0	15.68	63.788

2.6. Kesimpulan

Dari percobaan ini dapat diketahui bahwa *Load Balancer* yang menggunakan *Asynchronous Server* dapat memproses lebih cepat dibandingkan dengan *Threaded Server*. Hal ini menunjukkan bahwa performa *Asynchronous Server* lebih baik daripada *Threaded Server.*