BAB 5 IMPLEMENTASI

5.1 Implementasi

Implementasi sistem akan dilaksanakan jika perancangan sudah dilakukan. Implementasi akan mengacu pada perancangan yang sudah dibuat pada bab sebelumnya. Implementasi yang dilakukan akan menggunakan arsitektur yang sama dengan yang ada diperancangan.

5.1.1 Spesifikasi Perangkat Keras

Perangkat keras yang akan digunakan pada sistem ini ada 3, yaitu *host, broker,* dan *load balancer*. Masing – masing spesifikasi akan dijabarkan sebagai berikut :

1. ***Host (Publisher dan Subscriber)***

   a. Perangkat *Notebook*

      Perangkat ini akan berperan sebagai *Publisher*. Yang bertugas untuk mengirimkan pesan topik ke *Broker*

Tabel 5.1 Tabel Spesifikasi *Host*

Laptop Asus	
Processor	Intel Core i5-4200u 4 CPU @640Mhz
Memory (RAM)	8.00 GB, 1600 Mhz
Storage	1.00 TB
Operating System	Windows 8.1

Tabel 5.1 merupakan spesifikasi *Publisher* sekaligus *server* untuk *Broker* dan *load balancer*. Laptop dari brand ASUS, menggunakan Intel Core i5-4200m dengan *clockspeed* 1,7 GHz, RAM sebesar 8 GB, storage 1 TB dan menggunakan *OS* Windows 8.1.

2. Broker

a. Perangkat *Notebook dengan* Virtualisasi

   Perangkat ini akan berperan sebagai *server* untuk tempat berjalannya *Broker* menggunakan virtualisasi. Perangkat ini akan bertugas untuk meneruskan pesan dari *publisher* ke *subscriber.*

Tabel 5.2 Tabel Spesifikasi *Broker 1*

VM *Broker 1*	
Processor	Intel Core i5-4200u 1 CPU @640Mhz
Memory (RAM)	1.00 GB
Storage	8.26 GB
Operating System	Ubuntu *Server* 16.04 Xenial Xerus

Tabel 5.3 Tabel Spesifikasi *Broker 2*

VM *Broker 2*	
Processor	Intel Core i5-4200u 1 CPU @640Mhz
Memory (RAM)	1.00 GB
Storage	5.00 GB
Operating System	Ubuntu *Server* 16.04 Xenial Xerus

Tabel 5.4 Tabel Spesifikasi *Broker 3*

VM *Broker 3*	
Processor	Intel Core i5-4200u 1 CPU @640Mhz
Memory (RAM)	1.00 GB
Storage	5.00 GB
Operating System	Ubuntu *Server* 16.04 Xenial Xerus

Tabel 5.2, 5.3, dan 5.4 merupakan spesifikasi dari *Server broker 1*, *broker 2, dan broker 3. Semua broker* berjalan menggunakan virtualisasi dengan konfigurasi 1 CPU, RAM sebesar 1 GB dan semua *broker* berjalan menggunakan *OS* Ubuntu *Server* 16.04 Xenial Xerus.

3. Load Balancer

a. Perangkat *Notebook dengan* Virtualisasi

Perangkat ini akan berperan sebagai *server* untuk tempat berjalannya *Load Balancer* menggunakan virtualisasi. Perangkat ini akan bertugas untuk mendistribusikan *subscriber* ke *Broker*.

Tabel 5.5 Tabel Spesifikasi *Load Balancer*

VM *Load Balancer*	
Processor	Intel Core i5-4200u 1 CPU @640Mhz
Memory (RAM)	1.00 GB
Storage	6.49 GB
Operating System	Ubuntu *Server* 16.04 Xenial Xerus

Tabel 5.6 spesifikasi dari *Load Balancer. Load Balancer* berjalan di atas virtualisasi dengan konfigurasi 1 CPU, RAM sebesar 1 GB, storage 2.66 GB dan menggunakan *OS* Ubuntu *Server* 16.04 Xenial Xerus.

5.1.2 Spesifikasi Perangkat Lunak

1. *Host (*Berperan sebagai *publisher & subscriber)*

a. *Paho MQTT Python*

b. Wireshark

2. *Broker*

a. *Mosquitto Broker*

b. *Tshark*

c. *Pidstat*

3. *Load Balancer*

a. *HAProxy*

b. *Tshark*

c. *Pidstat*

5.1.3 Batasan Implementasi

Adapun batasan – batasan yang digunakan untuk membatasi implementasi sistem sehingga tidak keluar dari tujuan awalnya yaitu :

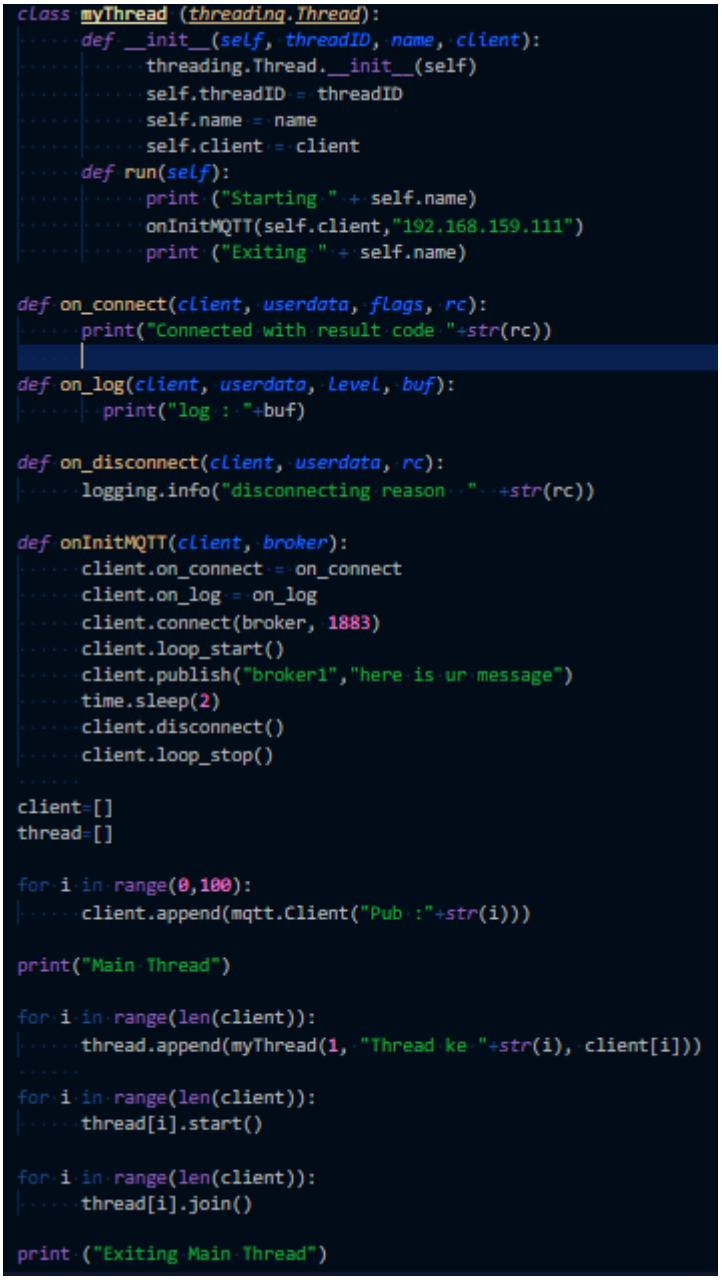
1. *Publisher* dan *subscriber* merupakan *script Python* yang memanfaatkan *threading* sehingga semua *publisher* dan *subscriber* memiliki IP yang sama tetapi *client id* yang berbeda.
2. Jenis *Load Balance* yang digunakan adalah *Round Robin*.
3. Semua *Broker*, *Load Balancer Subscriber,* dan *Publisher* berjalan pada 1 host fisik yang sama, dimana *broker* dan *load balancer* adalah *virtual machine*.
4. *Mosquitto* berjalan di *foreground* dan menggunakan konfigurasi yang sudah terdapat fungsi *bridge*.

5.1.4 Implementasi Pada *Host*

Implementasi pada *Host* yang berperan sebagai *publisher* dan *subscriber* menggunakan bahasa pemrograman *Python. Script Python* ini mengimplementasikan *library Paho MQTT*. Kemudian *host* juga menjalankan 4 *devices* secara virtualisasi menggunakan *VirtualBox*. Pada proses implementasi ini *host* hanya berperan sebagai *client* yang akan melakukan *stress test* terhadap sistem yang sudah dibuat.

5.1.4.1 *Implementasi untuk Publisher*

Publisher dibuat untuk melakukan *test* pada *broker* yang ada, kemudian untuk mengetahui kemampuan *broker* dalam melakukan membagikan pesan yang diterima. *Publisher* dibuat menggunakan bahasa *Python* dengan memanfaatkan *library* dari Paho *MQTT*. Pada *script* ini juga diimplementasikan konsep *threading* sehingga pada prosesnya nanti *script* ini mampu melakukan proses *publish* secara konkuren.



```
class myThread (threading.Thread):
    def __init__(self, threadID, name, client):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.client = client
    def run(self):
        print ("Starting " + self.name)
        onInitMQTT(self.client,"192.168.159.111")
        print ("Exiting " + self.name)

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

def on_log(client, userdata, level, buf):
    print("log : "+buf)

def on_disconnect(client, userdata, rc):
    logging.info("disconnecting reason  "  +str(rc))

def onInitMQTT(client, broker):
    client.on_connect = on_connect
    client.on_log = on_log
    client.connect(broker, 1883)
    client.loop_start()
    client.publish("broker1","here is ur message")
    time.sleep(2)
    client.disconnect()
    client.loop_stop()

client=[]
thread=[]

for i in range(0,100):
    client.append(mqtt.Client("Pub :"+str(i)))

print("Main Thread")

for i in range(len(client)):
    thread.append(myThread(1, "Thread ke "+str(i), client[i]))

for i in range(len(client)):
    thread[i].start()

for i in range(len(client)):
    thread[i].join()

print ("Exiting Main Thread")
```

Gambar 5.1 Implementasi kode *Python* pada *Publisher*

Gambar 5.1 di atas merupakan implementasi kode *Python* yang akan dijalankan ketika ingin melakukan pengujian menggunakan *publisher*. Kode ini menggunakan thread sehingga proses *publish* bisa dilakukan secara pararel menggunakan *multithreading*. Selain contoh kode *publisher* yang ada di gambar 5.1. Ada juga beberapa kode *Python* untuk proses *publish* dengan skenario yang berbeda – beda.

5.1.4.2 Implementasi untuk *Subscribe*r

Kode ini dibuat untuk melakukan *test*. Kode ini menggunakan bahasa pemrograman *Python* dan mengunakan *library Paho MQTT* sehingga program bisa

menggunakan fungsi – fungsi dasar seperti *subscribe*. Selain itu digunakan *thread* untuk dilakukannya *testing.*

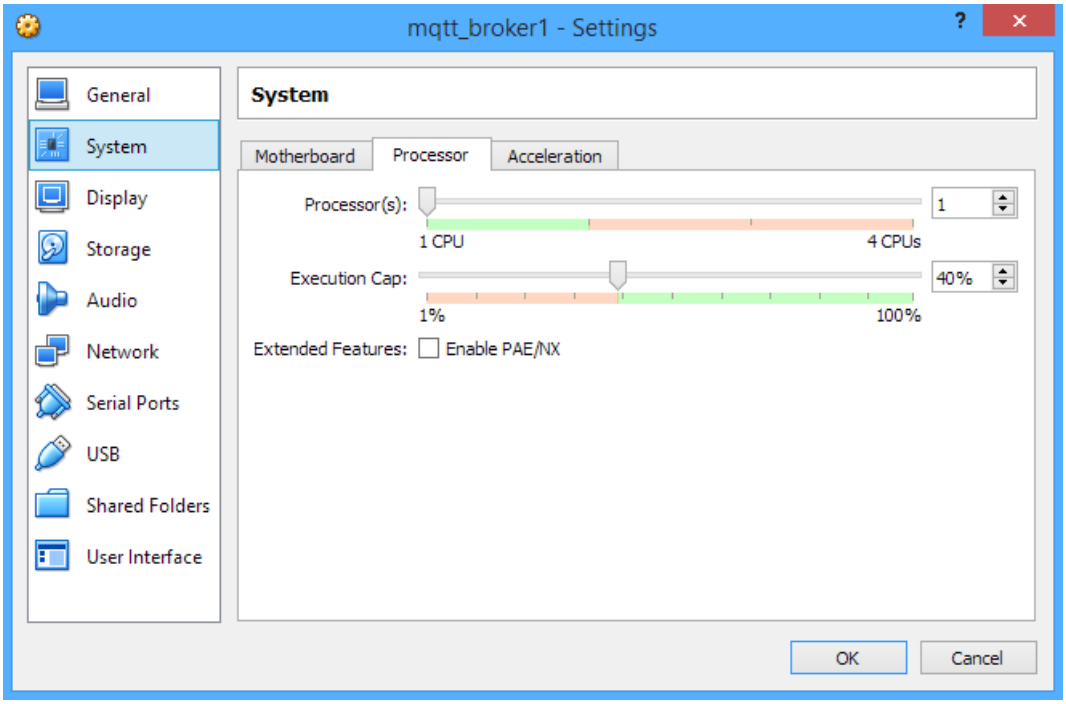
```
class myThread (threading.Thread):
    def __init__(self, threadID, name, client):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.client = client
    def run(self):
        print ("Starting " + self.name)
        onInitMQTT(self.client,"192.168.159.116")
        print ("Exiting " + self.name)

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    #client.subscribe("$SYS/#")
    client.subscribe("kuliah")


def onInitMQTT(client, broker):
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect(broker, 1883,45)
    client.loop_forever()


def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))
    print('Timestamp: {:%Y-%m-%d %H:%M:%S}'.format(datetime.datetime.now()))

client=[]
thread=[]

for i in range(0,800):
    client.append(mqtt.Client(str(20000 i)))

for i in range(len(client)):
    thread.append(myThread(1, "Thread ke "+str(i), client[i]))
    thread[i].start()

for i in range(len(client)):
    thread[i].start()

for i in range(len(client)):
    thread[i].join()

print ("Exiting Main Thread")
```

Gambar 5.2 Implementasi kode *Python* pada *Subscriber*

Gambar 5.2 di atas merupakan kode *Python* yang akan membuat thread baru, dimana setiap threadnya adalah *client* yang akan melakukan *subscribe* ke *load balancer* ataupun ke *broker*.

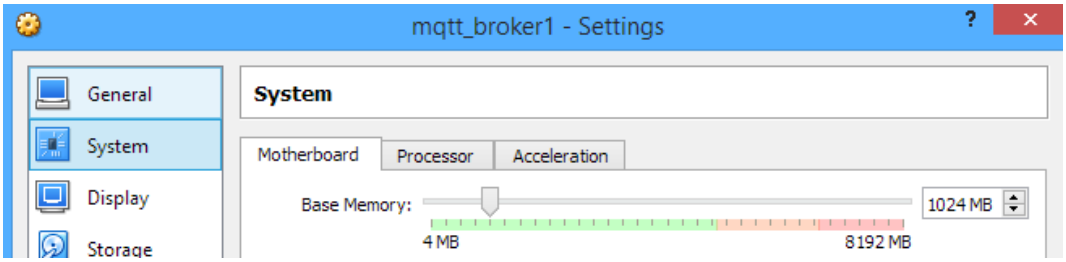
5.1.5 Implementasi Pada *Broker*

Pada implementasi ini, *broker* berjalan secara *virtual* di dalam *host*. Virtualisasi *broker* menggunakan *VirtualBox* yang menggunakan sistem operasi Ubuntu *Server* 16.04 Xenial Xerus. Prosesor yang digunakan adalah sama seperti prosesor host, akan tetapi pada virtualisasi, kecepatan prosesor sudah di konfigurasi menjadi 680Mhz. Gambar 5.1 di bawah ini.



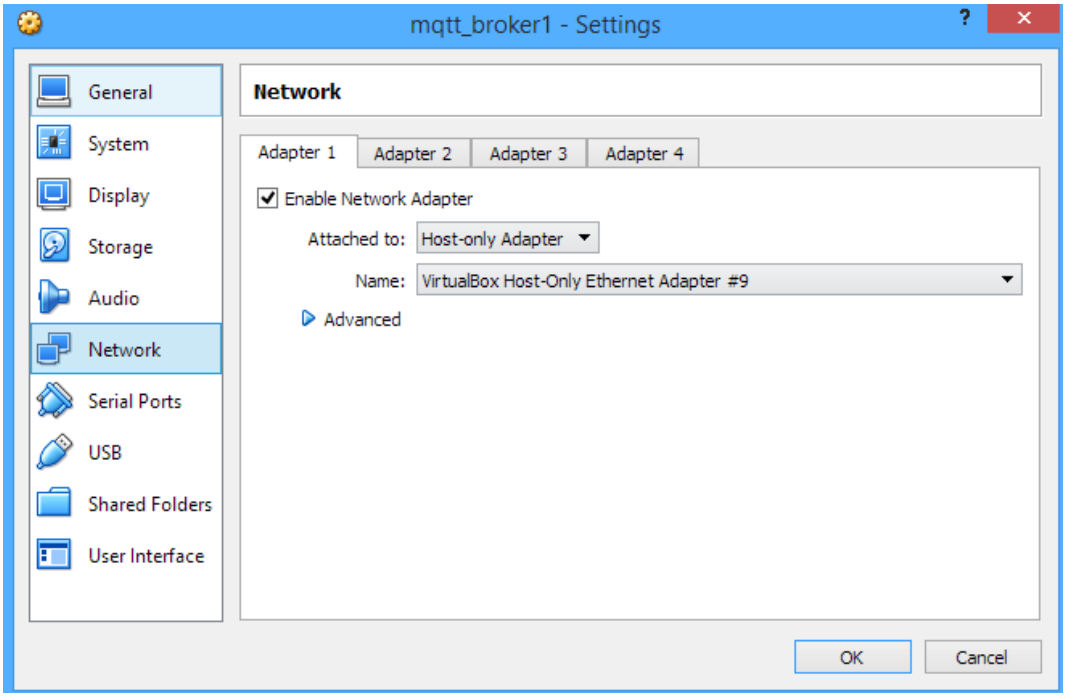
Gambar 5.3 Konfigurasi Clock Speed *Virtual Device Broker*

Kemudian RAM yang digunakan oleh *virtual device* adalah sebesar 1 GB sesuai dengan gambar 5.4 di bawah ini.



Gambar 5.4 Konfigurasi RAM *Virtual Device broker*

Kemudian *device* di atur sehingga berjalan di *environment* yang ada di dalam *host*. Gambar 5.5 di awah ini merupakan pengaturan *network interface* untuk *virtual device*.



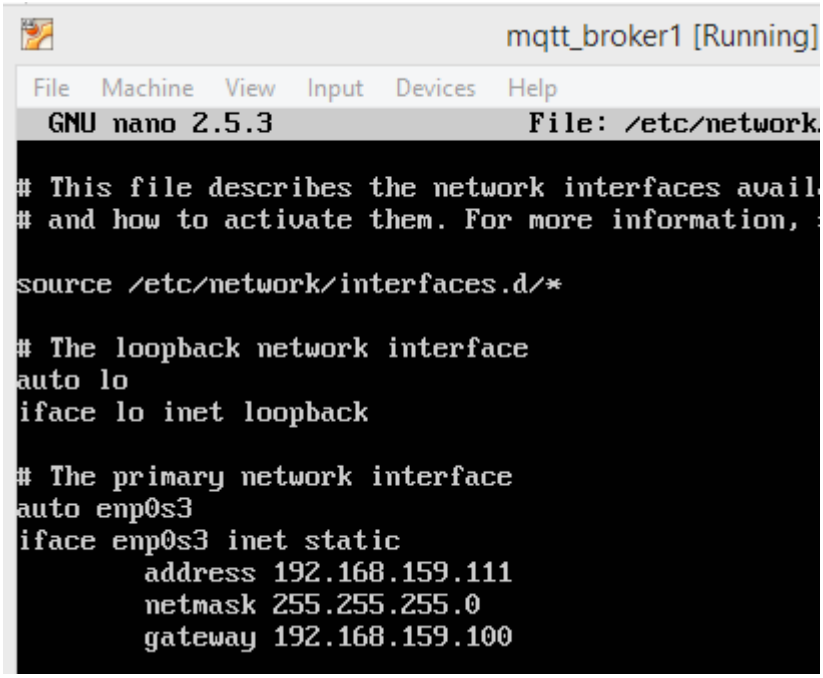
Gambar 5.5 Konfigurasi *network interface* pada *virtual device*

Server menggunakan aplikasi *MQTT Broker* dari Mosquitto yang sudah support *MQTT* v3.1.1. Adapun versi aplikasi Mosquitto yang digunakan adalah versi v1.4.8. Gambar 5.6 menunjukkan versi *mosquitto* yang diimplementasikan pada penelitian ini.

```
broker1@broker1:~$ mosquitto -h
mosquitto version 1.4.8 (build date Mon, 26 Jun 2017 09:31:02 +0100)

mosquitto is an MQTT v3.1 broker.
```

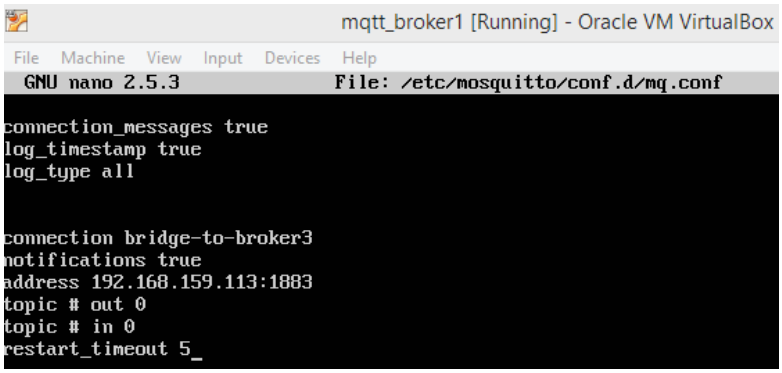
Gambar 5.6 Versi *Mosquitto* yang digunakan

Dengan menggunakan *mosquitto*, maka *server* sudah bisa menjadi *broker* karena sudah menjalankan service mosquitto. Selain itu, di *broker* juga terdapat *tshark* untuk melakukan *sniffing* di *traffic* jaringan yang dibutuhkan untuk proses pengujian. *Broker* ini berfungsi untuk meneruskan pesan berdasarkan topik dari *publisher* ke *subscriber* melalui komunikasi *wireless*. *Broker* menggunakan IP Static dimana IP ini merupakan IP Lokal yang hanya bisa diakses pada *environment* yang sama.



Gambar 5.7 Konfigurasi IP Static pada salah satu *broker*

Gambar 5.7 merupakan konfigurasi IP *Broker*. IP *Broker* diatur statik agar pengujian yang nanti akan dijalankan lebih mudah. Kemudian untuk dapat melakukan sinkronisasi pesan. *Mosquitto* memiliki fitur *bridging* untuk meneruskan pesan ke *broker* lain yang terhubung dengan *broker* ini melalui *bridge.* Untuk mengimplementasikan *bridging* maka diperlukan konfigurasi pada *setting Broker* sehingga *Broker* dapat bekerja sebagai *Bridge*.



Gambar 5.8 Konfigurasi *Bridging* pada salah satu *broker*

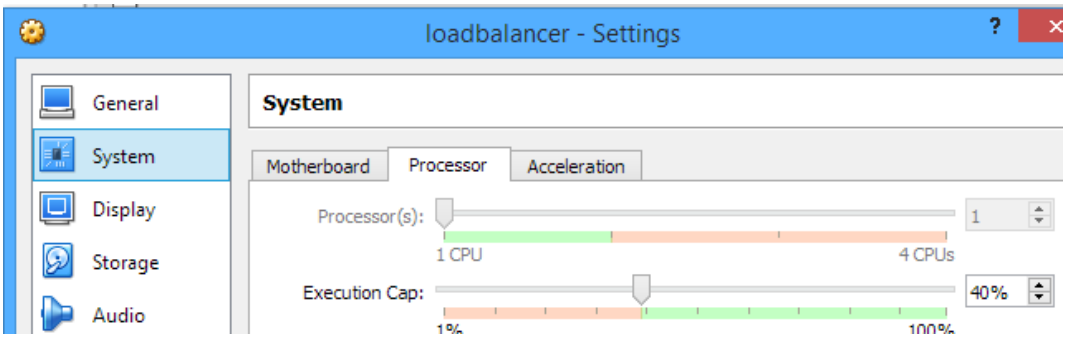
Gambar 5.8 merupakan konfigurasi *bridge*. Contoh gambar di atas menunjukkan bahwa *broker* ini melakukan *bridge* ke IP *Broker 3* melalui port 1883. Terakhir *broker* dijalankan pada *foreground* karena dibutuhkan *log* yang terjadi ketika *broker* menerima *subscribe* dan *publish*. Mode yang dijalankan *broker* akan ditunjukkan pada gambar 5.9.

```
broker1@broker1:~$ mosquitto -c /etc/mosquitto/conf.d/mq.conf
1514768900: mosquitto version 1.4.8 (build date Mon, 26 Jun 2017 09:31:02 +0100) starting
1514768900: Config loaded from /etc/mosquitto/conf.d/mq.conf.
1514768900: Opening ipv4 listen socket on port 1883.
1514768900: Opening ipv6 listen socket on port 1883.
1514768900: Bridge local.broker1.bridge-to-broker doing local SUBSCRIBE on topic #
1514768900: Connecting bridge bridge-to-broker (192.168.159.113:1883)
1514768900: Bridge broker1.bridge-to-broker sending CONNECT
1514768900: Received CONNACK on connection local.broker1.bridge-to-broker.
1514768900: Bridge local.broker1.bridge-to-broker sending UNSUBSCRIBE (Mid: 2, Topic: #)
1514768900: Bridge local.broker1.bridge-to-broker sending SUBSCRIBE (Mid: 3, Topic: #, QoS: 0)
1514768900: Received PUBACK from local.broker1.bridge-to-broker (Mid: 1)
1514768900: Received UNSUBACK from local.broker1.bridge-to-broker
1514768900: Received SUBACK from local.broker1.bridge-to-broker
```

Gambar 5.9 Tampilan ketika *broker* berjalan di *foreground*

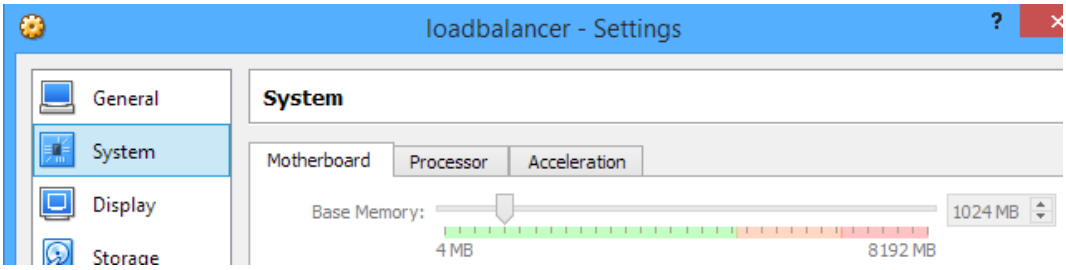
5.1.6 Implementasi Pada *Load Balancer*

Pada implementasi ini, *Load Balancer* berjalan secara *virtual* di dalam *host*. Virtualisasi menggunakan *Virtual* yang menggunakan sistem operasi Ubuntu *Server* 16.04 Xenial Xerus. Prosesor yang digunakan adalah sama seperti prosesor host, akan tetapi pada virtualisasi, kecepatan prosesor sudah di konfigurasi menjadi 680Mhz sesuai gambar 5.10.



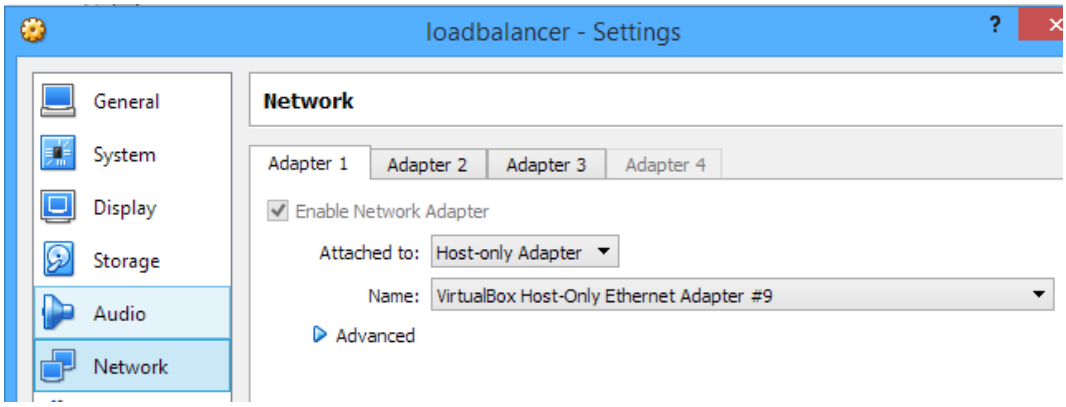
Gambar 5.10 Konfigurasi Clock Speed *Virtual Device Load Balancer*

Kemudian RAM yang digunakan oleh *virtual device* adalah sebesar 1 GB sesuai dengan gambar 5.11 di bawah ini.



Gambar 5.11 Konfigurasi RAM *Virtual Device Load Balancer*

Kemudian *device* di atur sehingga berjalan di *environment* yang ada di dalam *host*. Gambar 5.12 di awah ini merupakan pengaturan *network interface* untuk *virtual device*.



Gambar 5.12 Konfigurasi *network interface* pada *virtual device*

Kemudian untuk melakukan load balancing, *server* menggunakan aplikasi *HAProxy*. *HAProxy* yang digunakan adalah versi 1.6.3. Sesuai dengan gambar 5.13 di bawah ini.

```
loadbalance@loadbalancer:~$ haproxy -v
HA-Proxy version 1.6.3 2015/12/25
Copyright 2000-2015 Willy Tarreau <willy@haproxy.org>
```

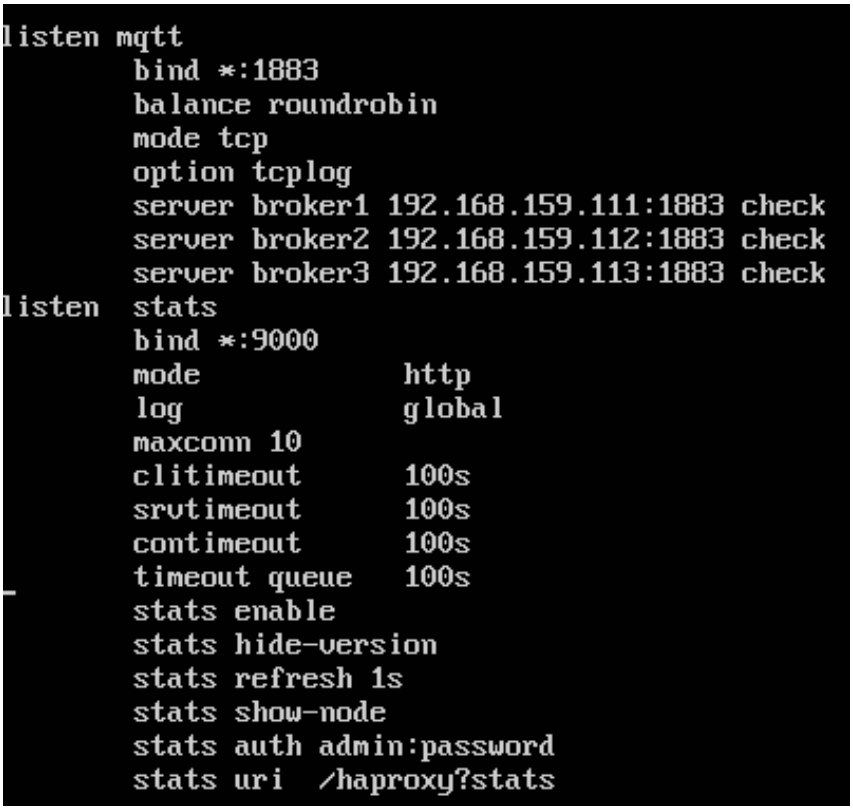
Gambar 5.13 Versi *HAProxy* yang digunakan

Dengan diaktifkannya service *HAProxy*, maka *server* sudah dapat bekerja sebagai *load balancer* yang akan mengatur *traffic* data yang menuju ke *server* ini.

```
# The primary network interface
auto enp0s3
        iface enp0s3 inet static
        address 192.168.159.116
        netmask 255.255.255.0
        gateway 192.168.159.100

auto enp0s8
        iface enp0s8 inet static
        address 192.168.43.80
        netmask 255.255.255.0
        gateway 192.168.43.1
```

Gambar 5.14 Konfigurasi IP Static pada Load Balancer

Gambar 5.14 merupakan konfigurasi IP Load Balancer. Load Balancer memiliki 2 *interface*, Yaitu interface *host only* untuk berkomunikasi dengan *Broker* dan interface keluar yang berguna untuk berkomunikasi dengan semua *device* yang terhubung pada jaringan yang sama dengan *Load Balancer*.



Gambar 5.15 Konfigurasi *HAProxy* pada Load Balancer

Gambar 5.15 merupakan konfigurasi *HAProxy* pada *Load Balancer*. Untuk menerima *request* dari *subscriber*, *load balancer* menggunakan port 1883, kemudian metode *balancing* yang digunakan adalah *round robin*. Karena *MQTT* berjalan pada paket *TCP*. Maka mode yang digunakan merupakan mode *tcp* sehingga paket *tcp* yang berasal dari *subscriber* bisa diarahkan ke *server* yang berada di *backend* untuk di lakukan load balancing. Sedangkan port 9000 digunakan untuk mengakses website yang berisikan informasi *monitoring HAProxy.*