# **BAB 5 IMPLEMENTASI**

# 5.1 Implementasi

Implementasi dilakukan dengan cara mengikuti desain yang telah dibuat. Untuk menerapkan sistem tersebut diperlukan suatu perangkat keras dan instalasi perangkat lunak serta konfigurasi komponen-komponen yang dibutuhkan dalam membangun sistem.

# 5.1.1 Spesifikasi Sistem

Untuk melakukan implementasi dibutuhkan beberapa komponen pada sebuah PC untuk menunjang keberhasilan sistem. Berikut ini merupakan komponen yang diperlukan untuk implementasi:

Prosesor	: Processor Intel Core i3-2370M CPU 2.40GHz
Hardisk	: 500 GB
Memory	: 8 GB
Sistem Operasi	: Ubuntu 14.04

# 5.2 Implementasi Sistem

Imlementasi dari laboratorium virtual jaringan komputer dibutuhkan komponen pendukung untuk keperluan fitur dalam sistem.

## 5.2.1 Docker

Pada penelitian ini *docker* di install dalam personal komputer. *Docker* yang digunakan pada penelitian ini versi 1.9.1. untuk melakukan instalasi *packet docker* dapat menggunakan perintah *apt-get install docker-engine* pada *terimanl* ubuntu kemudian lakukan *update* pada *repository* di dalam *image* ubuntu supaya komponen-kompinen *docker terupdate* dengan baik.

Langkah selanjutnya adalah pembuatan serta konfigurasi sesuai materi yang dibutuhkan untuk melakukan praktikum jaringan komputer. Materi untuk praktikum dikonfigurasi melalui dockerfile. Setiap materi praktikum memiliki konfigurasi yang berbeda sesuai kebutuhan materi pada bab yang akan dibuat praktikum. Ada tiga materi praktikum yang akan dirancang dalam penelitian ini. Berikut ini konfigurasi setiap *image* praktikum.

## 5.2.1.1 Packet Capturing

Bab pertama pada praktikum jaringan komputer ini adalah *packet capturing*. Untuk dapat melakukan *packet capturing* dibutuhkan beberapa komponen yang menunjang seperti berikut langkah-langkahnya. Tabel 5.1 Konfigurasi Dockerfile Materi Packet Capturing

```
    FROM ubuntu
    RUN apt-get update
    RUN mkdir /Capturing
    RUN cd Capturing
    RUN apt-get install tcpdump
```

Tabel 5.1 menunjukkan *script* yang dijalankan untuk melakukan instalasi tcpdump untuk *packet capturing*. Berikut penjelasan dari *script* diatas:

- Baris 1 mendefinisikan sebuah base image untuk membangun proses pada setiap docker image pada repository ubuntu
- Baris 2 merupakan perintah untuk melakukan update packet-packet yang tersedia pada container ubuntu.
- Baris 3 merupakan perintah untuk membuat sebuah *folder* dengan nama "Capturing".
- Baris 4 adalah perintah untuk masuk ke folder *capturing*.
- Baris 5 merupakan perintah untuk melakukan instalasi *tcpdump* sebagai perangkat lunak atau tools untuk praktikum packet capturing.

Setelah *dockerfile* dikonfigurasi sesuai kebutuhan praktikum, *dockerfile* di*build* menggunakan perintar *docker build* -*t* [*Repository:tag*] [*path*]. Tujuan *build* adalah untuk membuat *dockerfile* menjadi *image*. Pada kasus ini Pada kasus ini penulis menggunakan Imandos/packet\_capturing diamana Imandos adalah sebagai *user* dan *packet-capturing* adalah *repository* pada *dockerhub* dan *path*.

Proses selanjutnya adalah *push image* pada *docker registry*. Untuk melakukan proses *push* diperlukan akun pada *dockerhub* sebagai *repository* untuk menyimpan *image* secara *cloud*. *Dockerhub* memungkinkan *user* untuk membuat *repository* miliknya sendiri. *Push image* dapat dilakukan dengan perintah *docker push* [*user/repository:tag*]. *Docker push* merupakan perintah untuk melakukan *push* atau *upload* pada *repository* yang telah dibuat pada *dockerhub*. *Repository* merupakan nama yang sama dengan *repository* yang ada pada *dockerhub*. Pada kasus ini penulis menggunakan perintah Imandos/packet\_capturing:1.0

#### 5.2.1.2 Pemrograman Socket

Bab kedua pada jaringan komputer merupakan pemrograman *socket*, untuk melaukan praktikum ini dibutuhkan bebarapa *tools* seperti python, berikut ini merupakan *dockerfile* yang akan dijalankan :

### Tabel 5.2 Konfigurasi Dockerfile Pemrograman Socket (Server)

```
1 FROM ubuntu
2 RUN apt-get update -y
3 RUN apt-get install -y python
4
5 RUN touch server.py
6 RUN echo "import socket" >> server.py
```

```
RUN echo "sock=socket.socket(socket.AF INET,
   socket.SOCK STREAM) " >> server.py
   RUN echo "server=('',7777)" >> server.py
8
9
   RUN echo "sock.bind(server)" >> server.py
10 RUN echo "sock.listen(1000)" >> server.py
11 RUN echo "print 'Server menerima koneksi'" >> server.py
12 RUN echo "conn, addr = sock.accept()" >> server.py
13 RUN echo "data1=conn.recv(1024)" >> server.py
14 RUN echo "data='OK ' +data1" >> server.py
   RUN echo "conn.send(data)" >> server.py
15
16 RUN echo "print data" >> server.py
17
   RUN echo "conn.close()" >> server.py
18
19
   CMD python server.py
```

Tabel 5.3 Konfigurasi Dockerfile Pemrograman Socket (Client)

1	FROM ubuntu
2	RUN apt-get -y update
3	RUN apt-get install -y python
4	
5	RUN touch client.py
6	RUN echo "import socket" >> client.py
7	RUN echo "sock=socket.socket(socket.AF_INET,
	<pre>socket.SOCK_STREAM)" &gt;&gt; client.py</pre>
8	RUN echo "server=('172.17.0.2',7777)" >> client.py
9	RUN echo "sock.connect(server)" >> client.py
10	RUN echo "masuk=raw_input('Input: ')" >> client.py
11	RUN echo "sock.send(masuk)" >> client.py
12	RUN echo "datal=sock.recv(1024)" >> client.py
13	RUN echo "print data1" >> client.py
14	RUN echo "sock.close()" >> client.py
15	
16	CMD python client.py

Tabel 5.2 dan 5.3 menunjukkan *script* yang dijalankan untuk melakukan instalasi perangkat lunak *python* untuk praktikum permrograman *socket*. Berikut penjelasan dari *script* diatas:

- Baris 1 mendefinisikan sebuah base image untuk membangun proses pada setiap docker image pada repository ubuntu
- Baris 2 merupakan perintah untuk melakukan *update* packet-packet yang tersedia pada *container* ubuntu.
- Baris 3 merupakan perintah untuk melakukan instalasi aplikasi *python* sebagai perangkat lunak kebutuhan praktikum pemrograman *socket*.
- Baris 5 berfungsi untuk membuat *file* dengan nama server.py
- Baris 6 17 pada tabel 5.2 merupakan konfigurasi untuk membuat server pada python kemudian konfigurasi tersebut dimasukan ke dalam file server.py yang telah dibuat sebelumnya dengan perintah echo. Sedangkan baris 6-14 pada tabel 5.3 merupakan konfigurasi untuk membuat client pada python.

• Baris yang terakhir merupakan perintah untuk mengeksekusi konfigurasi *python* tersebut ketika dijalankan untuk praktikum.

Setelah *dockerfile* dikonfigurasi sesuai kebutuhan praktikum, *dockerfile* di-*build* menggunakan perintar *docker build* -*t* [*Repository:tag*] [*path*]. Tujuan *build* adalah untuk membuat *dockerfile* menjadi *image*. Pada kasus ini Pada kasus ini penulis menggunakan Imandos/pemsock\_server dan Imandos/pemsock\_client diamana Imandos adalah sebagai *user* dan pemsock\_server juga pemsock\_client adalah *repository* pada *dockerhub* dan *path*.

Proses selanjutnya adalah *push image* pada *docker registry*. Untuk melakukan proses *push* diperlukan akun pada *dockerhub* sebagai *repository* untuk menyimpan *image* secara *cloud*. *Dockerhub* memungkinkan user untuk membuat *repository* miliknya sendiri. *Push image* dapat dilakukan dengan perintah *docker push* [*user/repository:tag*]. *Docker push* merupakan perintah untuk melakukan *push* atau *upload* pada *repository* yang telah dibuat pada *dockerhub*. *Repository* merupakan nama yang sama dengan *repository* yang ada pada *dockerhub*. Penulis melakukan dua kali *push* karena ada dua *image* yaitu *server* dan *client*. Pada kasus ini menggunakan perintah docker push Imandos/pemsock\_server:1.0 dan docker push Imandos/pemsock\_client:1.0

#### 5.2.1.3 Protocol Transport Layer

Pada bab selanjutnya praktikum jaringan komputer berisi tentang *protocol transport layer*. Untuk melakukan praktikum ini dibutuhkan perangkat lunak yang mendukung, oleh sebab itu akan disediakan *vsftpd* sebagai perangkat lunak pendukung untuk melakukan praktikum. Berikut ini merupakan *dockerfile* yang akan dijalankan :

Tabel	5.4 Kon	figurasi Do	ckerfile Pra	aktikum Pro	otokol La	yer transport
-------	---------	-------------	--------------	-------------	-----------	---------------

FROM ubuntu
RUN apt-get update && apt-get install -yno-
install-recommends vsftpd
RUN apt-get clean
RUN echo "local_enable=YES" >> /etc/vsftpd.conf
RUN echo "chroot_local_user=YES" >>
/etc/vsftpd.conf
RUN echo "write_enable=YES" >> /etc/vsftpd.conf
RUN echo "local_umask=022" >> /etc/vsftpd.conf
RUN echo "anonymous_enable=NO" >> /etc/vsftpd.conf
RUN echo "allow_writeable_chroot=YES" >>
/etc/vsftpd.conf
RUN echo "secure_chroot_dir=/home/user/files" >>
/etc/vsftpd.conf
RUN useradd -ms /bin/bash user
RUN echo "user:imandos88"   chpasswd
RUN chmod 777 /home/user

```
17 RUN mkdir /home/user/files
18 RUN chown user:user /home/user/files/
19
20 VOLUME /etc
21
22 EXPOSE 20 21
23
24 CMD /usr/sbin/vsftpd
```

Tabel 5.4 menunjukkan *script* yang dijalankan untuk melakuakan instalasi *vsftpd* sebagai perangkat lunak pendukung untuk praktikum paket *capturing*. Berikut penjelasan dari *script* diatas:

- Baris 1 mendefinisikan sebuah base image untuk membangun proses pada setiap docker image pada repository ubuntu
- Baris 3 merupakan perintah untuk melakukan update pada repository ubuntu dan melakukan instalasi vsftp sebagai perangkat lunak untuk kebutuhan praktium.
- Baris 4 merupakan perintah untuk menghapus isi cache pada disk yang digunakan saat menjalankan perintah apt-get install secara otomatis dengan cara menghapus versi paling lama dari suatu package.
- Baris 6 12 merupakan perintah untuk melakukan konfigurasi pada *file* vsftpd.conf dengan konfigurasi yang telah dibuat sendiri untuk keperluan praktikum. Konfigurasi tersbut secara otomotasi akan me-*replace* konfigurasi sebelumnya atau konfigurasi *default* dari *vsftpd*.
- Baris 14 dan 15 merupakan perintah membuat *user* dan *password* baru untuk *login* pada *ftp*
- Baris 16-18 merupakan perintah untuk merubah hak akses dan hak kepemilikan pada *directory home/user*.
- Baris 20 perintah *volume* digunakan untuk mengaktifkan akses dari *container* kita ke direktori pada *host*.
- Baris 22 merukaan perintah untuk menghubungkan *port* 20 dan 21 untuk mengaktifan *network* antara proses yang berjalan di dalam *container* dan mesin *host*.

Setelah *dockerfile* dikonfigurasi sesuai kebutuhan praktikum, *dockerfile* di-*build* menggunakan perintar *docker build* -*t* [*Repository:tag*] [*path*]. Tujuan *build* adalah untuk membuat *dockerfile* menjadi *image*. Pada kasus ini penulis menggunakan Imandos/protocol\_layer\_transport diamana Imandos adalah sebagai *user* dan *protocol\_layer\_transport* adalah *repository* pada *dockerhub* dan *path*.

Proses selanjutnya adalah *push image* pada *docker registry*. Untuk melakukan proses *push* diperlukan akun pada *dockerhub* sebagai *repository* untuk menyimpan *image* secara *cloud*. *Dockerhub* memungkinkan *user* untuk membuat *repository* miliknya sendiri. *Push image* dapat dilakukan dengan perintah *docker*  push [user/repository:tag]. Docker push merupakan perintah untuk melakukan push atau upload pada repository yang telah dibuat pada dockerhub. Repository merupakan nama yang sama dengan repository yang ada pada dockerhub. Pada kasus ini menggunakan perintah docker push Imandos/protocol\_layer\_ transport:1.0

### 5.2.2 Antarmuka Implementasi

Berikut ini merupakan tampilan atau gambar yang berkaitan dengan sistem yang dibuat dan di implementasikan pada tugas akhir ini.

a Search		Dashb	oard Explore	Organizations Create 👻 🌅 imandos 👻
☐ imandos ▼				Private Repositories: Using 0 of 1 Get more
Repositories				Create Repository +
				Docker Security Scanning
imandos/packet_capturing public	0 STARS	3 PULLS	> DETAILS	Protect your repositories from vulnerabilities. Try it free
imandos/pemsock_server public	0 STARS	2 PULLS	DETAILS	
imandos/pemsock_client public	0 STARS	2 PULLS	DETAILS	
imandos/protocol_layer_transport public	0 STARS	2 PULLS	DETAILS	

Gambar 5.1 Repository Dockerhub

Gambar 5.1 merupakan *repository* yang telah dibuat pada *dockerhub*. Terdapat tiga materi yang dibuat namu *repository* pada gambar ada empat yaitu packet capturing, pemrograman socket dan protocol layer transport. Permrograman socket memiliki dua *image* yaitu *server* dan *client*. keempat *repository* tersebut harus memiliki nama yang sama dengan *image* yang akan di push.

imandos@SEBLANG88:~\$ sudo docker	images			
[sudo] password for imandos:				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
imandos/packet_capturing	1.0	664962933548	About an hour ago	165MB
imandos/pemsock_client	1.0	47a28005fef1	About an hour ago	192MB
imandos/pemsock_server	1.0	fe7e17f14591	About an hour ago	192MB
<pre>imandos/protocol_layer_transport</pre>	1.0	71775fd97d0f	About an hour ago	164MB
ubuntu	latest	14f60031763d	3 months ago	120MB
imandos@SEBLANG88:~\$				

### Gambar 5.2 Daftar Docker Image Yang dibuat

Gambar 5.2 merupakan daftar *image* yang telah dibuat, pada gambar terdapat tiga *image* yang masing-masing *image* di *generated* dari *dockerfile*. Setiap *imgae* memiliki image ID yang berbeda. *Tag* berfungsi sebagai penanda ataupun pembeda untuk setiap *image* dengan nama yang sama.

imandos@SEBLAN	G88:~\$ sudo docker ps -a				
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PO
RTS	NAMES				
aba4075d2539	<pre>imandos/protocol_layer_transport:1.0 suspicious lumiere</pre>	"/bin/sh -c /usr/s"	6 minutes ago	Exited (130) 3 minutes ago	
fef9a7a7598c	<pre>imandos/pemsock_client:1.0 determined_swirles</pre>	"/bin/sh -c 'pytho"	8 minutes ago	Exited (0) 7 minutes ago	
872abb1eb64a	imandos/pemsock_server:1.0 hungry_northcutt	"/bin/sh -c 'pytho"	8 minutes ago	Exited (0) 7 minutes ago	
7b291c97f227	imandos/packet_capturing:1.0 jolly knuth	"/bin/sh -c 'tcpdu"	11 minutes ago	Exited (0) 9 minutes ago	

### Gambar 5.3 Daftar Container Yang Telah Dibuat

Gambar 5.3 adalah daftar semua *container* yang telah dibuat. dengan status *exited* karena sudah tidak digunakan. Saat container tersebut digunakan kembali untuk praktikum maka status akan berubah menjadi *up*.

8 - C	imandos@SEBLANG88: ~
imandos	@SEBLANG88:~\$ sudo docker network inspect bridge
L r	
ĩ	"Name": "bridge"
	"Id": "4655086e7fde3b04848c3b8d53dcbbaa1cdf89ebef5dfc99844539b1e89f2c85
,	"Counter dis. "2017 10 20700-24-20 100710151-07-00"
	Created : 2017-10-20108:24:28.190/18451+07:00",
	Scope : local,
	Uriver: Dridge ,
	"TDAM". (
	IPAM : [ "Defuer": "defeate"
	Driver: derault,
	"Config": [
	com cy . [
	1 "Subpat": "172 17 0 0/16"
	"Cateway": "172 17 0 1"
	J
	1
	"Internal": false
	"Attachable": false
	"Ingress": false
	"Containers": {
	"7b291c97f2279a0a88a95446fb891606695940564cdae852b15f4b5b1b936e05":
ſ	
	"Name": "jolly knuth".
	"EndpointID": "31a2dd5c3acf7a61168fdc04e4713b77e1d6370a16a8d23a
c14ea08	337fc1dda",
	"MacAddress": "02:42:ac:11:00:02",
	"IPv4Address": "172.17.0.2/16",
	"IPv6Address": ""
	}

Gambar 5.4 Koneksi Bridge Pada Container

Gambar 5.4 merupakan salah satu *container* yang terhubung internet menggunakan *bridge*. Setelah container dibuat secara otomatis ditambahkan pada koneksi *bridge* dan mendapatkan *ip addres* sesuai *subnet*.

#### 5.2.2.1 Praktikum Packet Capturing

😸 🗇 💷 Imandos@SEBLANG88: ~	,
<pre>imandos@SEBLANG88:~\$ sudo docker run -it imandos/packet_capturing:1.0</pre>	
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode	
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes	
01:29:35.200430 IP6 :: > TT02::10: HBH ICMP6, Multicast Listener report v2, 1 group record(s), len	gth 28
01:29:35.286/88 ARP, Request who-has 1/2.1/.0.1 tell /b291c9/f22/, length 28	
01:29:35.280850 ARP, Repty 1/2.1/.0.1 LS-at 02:42:49:a7:03:06 (out Unknown), tength 28	
01:29:35.280858 1P /029129/1227.55/07 > google-public-dns-a.google.com.53: 11322+ Pik? 0.1.0.0.0.0	.0
. (po.drpd. (90) Aligois depending for the foreign the foreign of a ffering with the multicast listenes const up	1 group sprovd(s) longth 28
01:23:35 45864 TP6 Fe80: 7Cbb:60Ff Fe80 200 F 102:100 non TCHP0, Muttcast tistener Tepott V2,	16
01:29:35.466420 TP6 Fe80::7cbb:60ff:fe9e:9ef > ff02::16: HBH ICMP6 multicast listener report v2	1 aroun record(s) length 28
01:29:43.310827 IP 7b291c97f227.51903 > google-public-dns-a.google.com.53: 36952+ PTR? 1.0.17.172.	in-addr.arpa. (41)
01:29:51.515376 IP 7b291c97f227.41352 > google-public-dns-a.google.com.53: 12325+ PTR? 8.8.8.8.in-	addr.arpa. (38)
01:29:56.330582 ARP, Request who-has 172.17.0.1 tell 7b291c97f227, length 28	
01:29:59.538133 IP 7b291c97f227.49826 > google-public-dns-a.google.com.53: 59403+ PTR? f.9.e.9.e.9	.e.f.f.f.9.6.b.b.c.7.0.0.0.0.0.0.0.0.0.0.0.0.0.8.e.f
.ip6.arpa. (90)	
01:30:38.579739 IP6 fe80::7cbb:69ff:fe9e:9e9f.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ippstcp.lo	cal. PTR (QM)? _ipptcp.local. (45)
01:30:38.580083 IP 7b291c97f227.44751 > google-public-dns-a.google.com.53: 34894+ PTR? b.f.0.0.0.0 .ip6.arpa. (90)	.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0
^C^Z01:30:38.614164 IP6 fe80::42:49ff:fea7:3b6.5353 > ff02::fb.5353: 0 [2q] PTR (QM)? _ippstcp.le	ocal. PTR (QM)? _ipptcp.local. (45)
14 packets captured	
89 packets received by filter	State A A A A A A A A A A A A A A A A A A A
69 packets dropped by_kernel	
1mandos@SEBLANG88:~S	

**Gambar 5.5 Packet Capturing** 

Pada gambar 5.5 dilakukan penangkapan *packet* terhadap jaringan *eth0*. Kemudian proses di hentikan dan mendapatkan hasil 14 packet captured, 89 pcket received by filter dan 69 packet *dropped by kernel*.



#### 5.2.2.2 Praktikum Pemrograman Socket

Gambar 5.6 Pemrograman Socket

Gambar 5.6 merupakan praktikum pemrograman *socket* saat dijalankan. Dua *image* dijalankan untuk praktikum pemrograman *socket, image* satu sebagai *server* dan *image* yg kedua sebagai *client*. Saat *server* dijalankan maka *server* akan menampilkan *output* "server menerima koneksi" selanjutnya *image client* dijalankan dan *user* diminta *input* untuk di kirim ke *server*. *server* menerima dan membalas dengan pesan "OK input".

### 5.2.2.3 Praktikum Pemrograman Socket

Firefox Web Browser		t‡	En	* 💌	) (( <b>(</b> ) (	08:35	⇔
🖉 Connecting 🗙 🖕							
( ) [ftp://172.17.0.2	- X Q Search	☆ 自	+	â	9		≡
	ubuntu <sup>o</sup>						
	Google						
	Second Se						
	Enter username and password for ftp://172.17.0.2						
	User Name:						
	Password:						
	Cancel OK y						

Gambar 5.7 Protocol Layer Transport

Gambar 5.7 merupakan praktikum *protocol layer transport* saat dijalankan. *Image* yang dijalankan merupakan *ftp server*. untuk pengaksesan *ftp server* tersebut dapat menggunakan *browser* maupun *filezilla* untuk *transfer file*.

Firefox Web Browser		t <b>, ⊡</b> % 0	🔊 📣) 08:35 🛟
🖉 Connecting 🗙 🖕			
( )   ftp://172.17.0.2	• X Q Search	☆ 自 ∔ 合	
Index of ftp://172.17.0.2/ - Mozilla Firefox	ubuntu <sup>©</sup> Coccle Password: user Password: Cancel OK y	1. D \$	■ <b>4</b> 0) 08:36 <b>(</b>
(⇐ 𝒫) ①   ftp://172.17.0.2	C Q Search	☆ 自 ♣ 俞	• • <b>•</b> =
Index of ftp://1	72.17.0.2/ I directory Size La 20	<b>ast Modified</b> 9/10/17 07:19:00	

# Gambar 5.8 Pengaksesan FTP Pada Browser

Gambar 5.8 adalah file FTP yang diakses melalui *browser* dengan alamat 172.17.0.2 selanjutnya *browser* akan meminta *user* dan *password*. Kemudian untuk *upload* file pada ftp dapat menggunakan *filezilla*.