BAB III
METODE PENELITIAN

Penulis menggunakan metode eksperimental dengan cara melakukan penelitian mengenai sebab akibat melalui proses uji coba dengan mengisolasi parameter percobaan sehingga tidak terganggu oleh variabel lain. Lalu menggunakan pendekatan kuantitatif bersifat objektif yang direpresentasikan dengan statistik dan data numerik yang dilakukan dengan menerapkan pola analisis dan deduktif.

3.1 PERANGKAT YANG DIGUNAKAN

3.1.1 PERANGKAT KERAS (*HARDWARE*)

Spesifikasi *cluster* yang digunakan dalam peneletian ini adalah tiga buah raspberry pi 3 model B dengan rincian perangkat keras sebagai berikut :

Tabel 3.1 Spesifikasi *Cluster* Raspberry pi 3 Model B

Nama	Keterangan
Processor	64-bit quad-core ARM Cortex-A53
Processor Speed	1.2 Ghz
RAM	1024 MB
Storage	Sandisk Ultra SD XC I class 10
Operating System	Raspbian Lite 64 Bit
Connectivity	100 Mb/s Fast Ethernet

Sedangkan *client* yang nantinya akan digunakan untuk melakukan iterasi percobaan *benchmarking* dan *load test*, menggunakan perangkat laptop yang memiliki spesifikasi sebagai berikut:

Tabel 3.2 Spesifikasi *Client*

Nama	Keterangan
Processor	AMD A8-7410 Quad Core
Processor Speed	2.2 GHz (Base) 2.48 GHz (Boost)
RAM	8 GB

Nama	Keterangan
Storage	Samsung Evo 860 250 GB Sata SSD
Operating System	Windows 10 64 Bit Ver.21H1
Connectivity	100 Mb/s Fast Ethernet

3.1.2 PERANGKAT LUNAK (*SOFTWARE*)

Untuk perangkat lunak yang digunakan untuk implementasi dan pengujian, penulis menggunakan perangkat lunak yang terbaru atau paling umum digunakan, serta perangkat lunak yang masih didukung oleh pengembangnya hingga satu tahun terakhir, sehingga perangkat lunak yang digunakan diharapkan dapat bekerja dengan sebagai mana mestinya

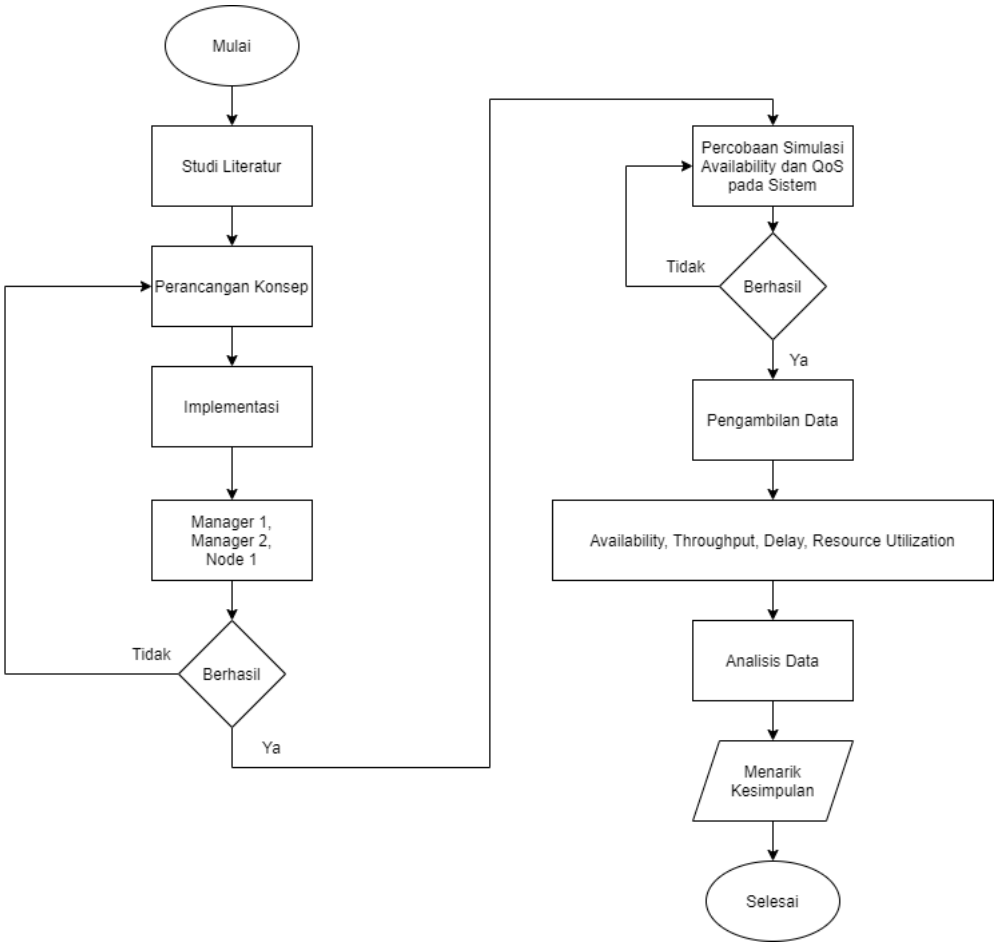
Tabel 3.3 *Software* yang digunakan

Nama Software	Versi	Keterangan
Docker Ce	20.10.14, build a224086	Docker container *engine*
Nginx	1.20	Web *Server*
Apache Benchmark	5.4.3	Web *server Performance Testing Tool*
Siege	4.1.2	Web *server regression test and benchmark utility*
Node Exporter	0.18.1	*Prometheus system resource matrix collector and exporter*
Prometheus	2.34.0	*System monitoring and alerting tool kit*
Grafana	8.2.6	*Analytic and visualization web application*

3.2 ALUR PENELITIAN

Penelitian ini dilakukan melalui beberapa langkah, pertama kali yang dilakukan penulis yaitu melakukan studi literatur untuk mengetahui *key-concept* dari apa yang akan diteliti. Selanjutnya adalah melakukan iterasi pada langkah perancangan konsep hingga implementasi, apabila didapati hasil yang sesuai, yaitu diperoleh hasil 2 buah manager dan 1 *node* telah terkonfigurasi

Langkah berikutnya penulis juga melakukan iterasi pada percobaan simulasi dengan mengacu pada skenario yang telah disiapkan. Apabila percobaan simulasi berhasil, maka dilakukan pengambilan data yang mana meliputi beberapa parameter yaitu *Availability*, *Throughput*, *Resource Utilization*, dan *Delay*. Setelah didapat data yang diinginkan, maka penulis melakukan analisis data dan mengolahnya agar dapat direpresentasikan dengan statistik. Langkah terakhir yaitu menarik kesimpulan dari statistik data yang didapat dari penelitian.

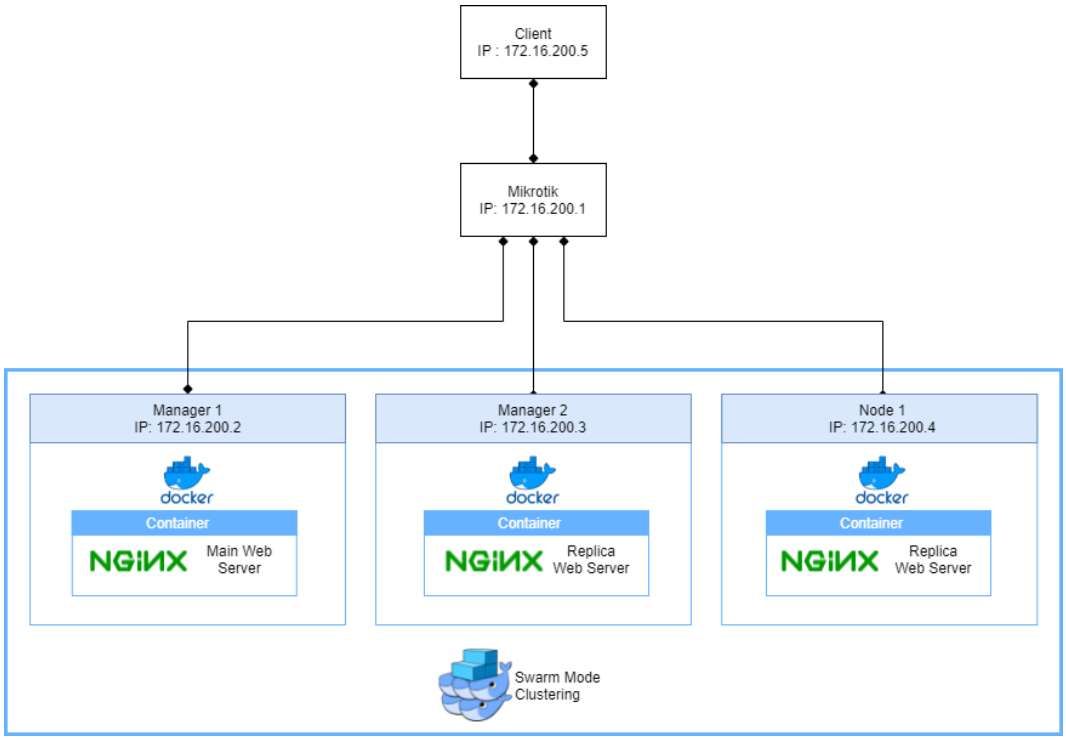


Gambar 3.1 Diagram Alur Penelitian

3.3 PERANCANGAN SISTEM

3.3.1 Arsitektur Sistem

Penulis menerapkan mikrotik sebagai media failover agar dapat dicapai tingkat ketersediaan yang tinggi meskipun satu atau dua *node* mengalami kegagalan. Untuk alamat IP pada sistem penulis menetapkan alamat ip sebagai berikut:



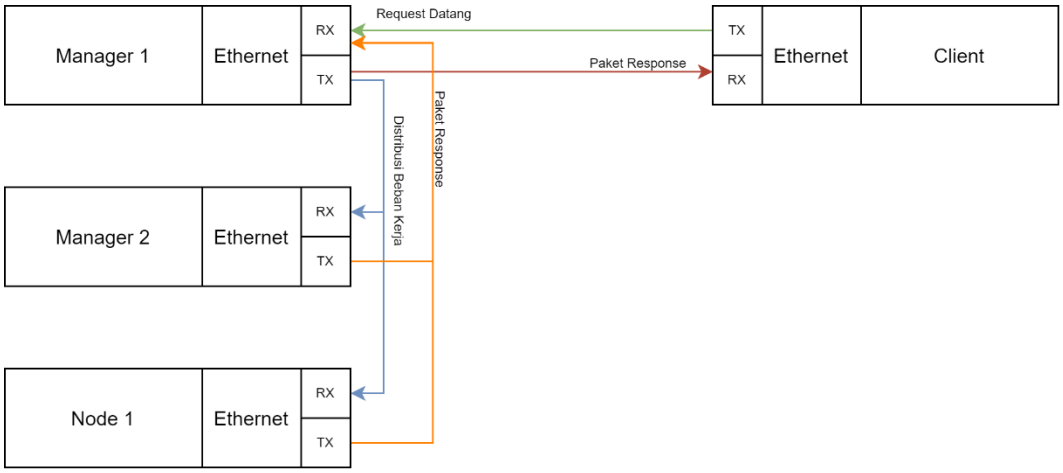
Gambar 3. 2 Topologi Sistem

Tabel 3. 4 Alamat IP Jaringan *Cluster*

Perangkat	Alamat IP
Mikrotik	172.16.200.1
Manager 1	172.16.200.2
Manager 2	172.16.200.3
Node 1	172.16.200.4
Client	172.16.200.5

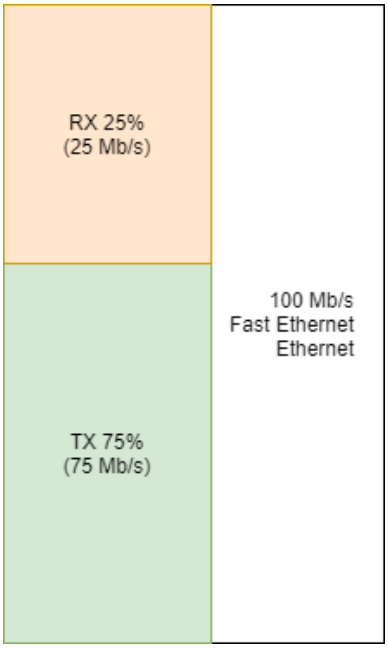
Pada gambar 3.2 dapat diamati bahwa seluruh *node* yang dilakukan *clustering* dapat berkomunikasi satu sama lain melalui ethernet yang tersambung dengan mikrotik sebagai sarana *failover*, sedangkan komunikasi distribusi pemrosesan *request* ditangani oleh layanan docker yang menerapkan swarm mode, sehingga ketika *request* datang ke manager *node*, dapat didistribusikan menuju *node* lain yang tersedia.

Karena distribusi pekerjaan dikirimkan menuju *node* lain menggunakan *interface* yang sama dengan arah datangnya *request*, maka RX dan TX tidak dapat dilakukan secara bersamaan dengan kecepatan 100 Mb/s (*full Speed*) dikarenakan *interface* pada raspberry pi 3 model b menggunakan Fast Ethernet yang memiliki kecepatan 100 Mb/s sehingga distribusi beban kerja yang dilakukan oleh manager terbatasi oleh kecepatan maksimum dari *interface* tersebut. Berikut ilustrasi untuk menggambarkan alur *traffic* pada *cluster* :



Gambar 3. 3 *Network Flow Diagram*

Pada gambar 3.3 dapat dilihat bahwa *bottleneck* terjadi di Manager 1 yang mana sebagai load balancer yang aktif. Terjadinya *bottleneck* disebabkan oleh keterbatasan *bandwidth* dari *interface* raspberry pi yang hanya 100Mb/s, artinya jika 75% bandwidht digunakan untuk transmit data, maka receive data hanya dapat dilakukan maksimal pada 25% kapasitas dari *interface* tersebut. Berikut ilustrasi terjadinya pemakaian *bandwidth* pada *interface* Fast Ethernet 100Mb/s:



Gambar 3. 4 *Bandwidth* pada 100Mb/s *Network Interface*

3.4 KONFIGURASI SISTEM

3.4.1 Konfigurasi IP pada perangkat

Pada masing-masing *node* dilakukan konfigurasi Alamat IP *Static* sesuai dengan ketentuan tabel 3.4, sehingga alamat ip tidak akan berubah-ubah, tentunya konfigurasi dilakukan pada *interface* ethernet yang terhubung dengan mikrotik. Untuk mengecek alamat ip pada raspberry pi dapat menggunakan perintah sebagai berikut:

```
root@pimanager1:~# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.200.2  netmask 255.255.255.240  broadcast 172.16.200.15
        inet6 fe80::e540:768c:b64e:77d3  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:0d:d3:f1  txqueuelen 1000  (Ethernet)
        RX packets 6849916  bytes 3716348234 (3.4 GiB)
        RX errors 0  dropped 352111  overruns 0  frame 0
        TX packets 6071903  bytes 4606978359 (4.2 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Begitu juga pada raspberry pi manager 2

```
root@pimanager2:~# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.200.3  netmask 255.255.255.240  broadcast 172.16.200.15
        inet6 fe80::5d96:8325:114:c2b1  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:c0:3f:cc  txqueuelen 1000  (Ethernet)
        RX packets 2261436  bytes 310842021 (296.4 MiB)
        RX errors 0  dropped 164  overruns 0  frame 0
        TX packets 2806615  bytes 2187559232 (2.0 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Pengecekan ip pada *node* 1 :

```
root@pinode1:~# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.200.4  netmask 255.255.255.240  broadcast 172.16.200.15
        inet6 fe80::e8e3:d0ce:96ea:7d4a  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:8a:bb:e5  txqueuelen 1000  (Ethernet)
        RX packets 1272638  bytes 150236825 (143.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1816165  bytes 2003765568 (1.8 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3.4.2 Instalasi dan konfigurasi docker

Berikut merupakan konfigurasi yang dilakukan penulis dalam proses installasi docker :

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli vontainerd.ie docker-compose-plugin
```

Untuk verifikasi apakah docker sudah sukses diinstall dan berjalan menggunakan perintah :

```
root@pimanager1:~# docker version
Client: Docker Engine - Community
 Version:           20.10.14
 API version:       1.41
 Go version:        go1.16.15
 Git commit:        a224086
 Built:             Thu Mar 24 01:47:24 2022
 OS/Arch:           linux/arm64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.14
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.15
  Git commit:       87a90dc
  Built:            Thu Mar 24 01:45:44 2022
  OS/Arch:          linux/arm64
  Experimental:     true
 containerd:
  Version:          1.5.11
  GitCommit:        3df54a852345ae127d1fa3092b95168e4a88e2f8
 runc:
  Version:          1.0.3
  GitCommit:        v1.0.3-0-gf46b6ba
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

3.4.3 Konfigurasi docker swarm

Agar *node* dapat disatukan menjadi *cluster*, dilakukan konfigurasi inisiasi cluster pada manager 1 menggunakan perintah :

"docker swarm init"

Lalu akan muncul kode api yang dapat digunakan oleh *node* lain untuk bergabung ke dalam *cluster* seperti yang tertera di bawah ini:

```
root@pimanager1:~# docker swarm init
Swarm initialized: current node (ar7msfznbmq9s1ly5ff9zinqr) is now a
manager.
To add a worker to this swarm, run the following command:

docker swarm join --token SWMTKN-1-
13wntoyrme6yocms8b2vs5vbkp1imnfg6ktmayqj19233pobu9-
6yi9ihlyzklmq3xs6eobqsbd6 172.16.200.2:2377

To add a manager to this swarm, run 'docker swarm join-token manager'
and follow the instructions
```

Setelah proses inisialisasi selesai, maka *node* lain dapat bergabung ke dalam *cluster* menggunakan kode yang tampil di atas dengan perintah :

```
"docker swarm join --token SWMTKN-1-
1tvtcf21g1bqkrxfeqdf65qeb4xo6qutm0b6pq5o9aitqhe1dv-
62jg75tyj2mbnms5862bnpl5q 172.16.200.2:2377"
```

Ketika *node* sukses bergabung ke dalam *cluster*, maka akan tampil keterangan sukses di konsol seperti :

```
root@pimanager2:~# docker swarm join --token SWMTKN-1-
13wntoyrme6yocms8b2vs5vbkp1imnfg6ktmayqj19233pobu9-
6yi9ihlyzklmq3xs6eobqsbd6 172.16.200.2:2377
This node joined a swarm as a worker.
```

```
root@pinode1:~# docker swarm join --token SWMTKN-1-
13wntoyrme6yocms8b2vs5vbkp1imnfg6ktmayqj19233pobu9-
6yi9ihlyzklmq3xs6eobqsbd6 172.16.200.2:2377
This node joined a swarm as a worker.
```

Pada kondisi ini manager 2 masih berstatus sebagai *worker node* sehingga perlu dilakukan konfigurasi agar manger 2 menjadi *manager node*. Konfigurasi dilakukan pada manager 1 menggunakan perintah *"docker node promote*

pimanager2" yang mana "pimanager2" adalah hostname dari manager 2 menggunakan perintah :

```
root@pimanager1:~# docker node promote pimanager2
Node pimanager2 promoted to a manager in the swarm.
```

Sehingga apabila dilakukan verifikasi *node* yang terhubung menggunakan perintah *"docker node ls"* manager 1 dan manager 2 akan berstatus sebagai *manager node* seperti pada *command* berikut :

```
root@pimanager1:~# docker node ls
ID              HOSTNAME    STATUS  AVAILABILITY  MANAGER STATUS
Nt882l7nkew     pimanager1  Ready   Active        Leader
Ygbmkfsgki3     pimanager2  Ready   Active        Reachable
Ha05ta03ahdz    pinode1     Ready   Active
```

3.4.4 Konfigurasi *Web Server*

Konfigurasi yang dilakukan adalah menerapkan *software* NGINX sebagai *web server* menggunakan docker image dan dijalankan sebagai service berikut merupakan konfigurasi file yaml yang dilakukan :

```
version: "3.9"
services:
  wordpress:
    image: nginx:latest
    volumes:
      - /var/www/skripsi:/usr/share/nginx/html
    deploy:
      mode: global
      update_config:
        parallelism: 1
        delay: 60s
      restart_policy:
        condition: none
        max_attempts: 5
      restart_policy:
        condition: on-failure
```

Setelah konfigurasi berhasil didefinisikan, selanjutnya *service* akan di*deploy* menggunakan perintah :

```
root@pimanager1:~/deployment# docker stack deploy –c nginx.yaml web
Creating network web_default
Creating service web_nginx
root@pimanager1:~/deployment#
```

Untuk verifikasi apakah *service* berhasil dijalankan, dilakukan pengecekan menggunakan perintah *"docker service ls"* dan menghasilkan tampilan seperti berikut :

```
root@pimanager1:~/deployment# docker service ls
ID              NAME        MODE        REPLICAS    IMAGE          PORTS
74knsyglen4z    web_nginx   global      3/3         nginx:latest   *:80→ 80/tcp
root@pimanager1:~/deployment#
```

3.5 PENGUJIAN SISTEM

Pengujian pada penelitian ini dimaksudkan agar dapat mengetahui apakah docker swarm dapat beroperasi dengan lancar sehingga didapat nilai availability yang sesuai standar "*nines"* dan menguji seberapa baik performa yang mampu diberikan oleh konfigurasi sistem pi-*cluster* dengan metode *clustering* menggunakan docker swarm.

Pada setiap skenario, penulis menerapkan variasi mulai dari 500, 600, 700, 800, 900 dan 1.000 *concurrent user*. Hal ini dimaksudkan untuk mencari nilai *breaking point* atau kapasitas maksimal yang dapat di*handle* oleh pi-cluster menggunakan metode *clustering* docker swarm.

3.5.1 Skenario Pengujian

Tabel 3.5 Skenario Pengujian

Skenario	Manager 1	Manager 2	*Node* 1	*Concurrent*
Skenario 1	*Running*	*Running*	*Running*	500
				600
				700
				800
				900
				1.000

Skenario	Manager 1	Manager 2	*Node* 1	*Concurrent*
Skenario 2	*Running*	*Down*	*Running*	500
				600
				700
				800
				900
				1.000
Skenario 3	*Down*	*Running*	*Running*	500
				600
				700
				800
				900
				1.000
Skenario 4	*Running*	*Down*	*Down*	500
				600
				700
				800
				900
				1.000
Skenario 5	*Down*	*Running*	*Down*	500
				600
				700
				800
				900
				1.000

Skenario	Manager 1	Manager 2	*Node* 1	Concurrent
Skenario 6	*Running*	*Running*	*Down*	500
				600
				700
				800
				900
				1.000
Skenario 7	*Down*	*Down*	*Running*	500
				600
				700
				800
				900
				1.000

Skenario 1 : Pengujian performa pada saat seluruh *node* aktif dan dapat diakses, untuk mengambil data acuan, sehingga dapat dilakukan perbandingan pada saat *cluster* berada dalam kondisi normal, dan ketika salah satu atau dua perangkat dalam *cluster* tersebut mengalami kegagalan.

Skenario 2 : Pengujian dilakukan dengan kondisi manager 2 pada keadaan *down*, dilakukan untuk mengetahui apakah web *server* masih dapat diakses jika *node* manager 2 *down*.

Skenario 3 : Pengujian dengan kondisi manager 1 tidak tersedia, sehingga dapat diketahui pengaruh apakah jika salah satu dari manager baik itu manager 1 atau 2 akan memberikan dampak terhadap akses web *server*.

Skenario 4 : Kondisi manager 2 dan *node* 1 dalam keadaan tidak tersedia, dilakukan untuk mengetahui apakah *manager* dapat menangani *traffic* yang datang apabila salah satu *manager* dan *node* mengalami kegagalan dan hanya tersisa satu *manager* itu sendiri.

Skenario 5 : Pada pengujian ini manager 1 dan *node* 1 dalam kondisi *down* seperti skenario 4 dilakukan untuk mengetahui apakah satu manager dapat

menangani *traffic* yang datang, akan tetapi pada skenario kali ini, yang menangani *request* hanya manager 2 saja.

Skenario 6 : Dilakukan untuk menguji apakah web *server* masih dapat diakses ketika *node* pekerja (*worker-node*) mengalami kegagalan dan hanya manager saja yang menghandle *request*.

Skenario 7 : Pada skenario ini kondisi manager 1 dan 2 *down* dan hanya *node* 1 saja yang *running*, dilakukan untuk mengetahui apakah jika tersisa *worker node* saja, web *server* masih tetap dapat diakses.

Pada pengujian variasi *concurrent user* dilakukan dengan menggunakan siege dengan variabel *concurrency* dan *time*. Variabel *concurency* (-c) digunakan untuk menentukan banyaknya *concurrent user* (pengaksesan *user*) secara bersamaan, sedangkan variabel *time* digunakan untuk menentukan berapa lama pengujian dilakukan. Menurut konsep High Availability, layanan harus dapat diakses kapan saja, pada pengujian yang dilakukan, penulis menetapkan waktu pengujian selama 5 menit pada setiap percobaan, sehingga apabila melihat tabel 3.4, pengujian dapat dilakukan dengan menggunakan perintah :

siege http://172.16.200.1 –c [*value*] –t 300

dengan nilai *value* merupakan nilai concurency: 500, 600, 700, 800, 900, 1.000

3.5.2 Parameter Pengujian

Sedangkan untuk data yang diambil dari pengujian tersebut adalah parameter Failover Test bertujuan untuk mengetahui apakah proses *failover* dapat berjalan dengan sebagaimana mestinya dan layanan masih dapat diakses. Pengambilan data Availability untuk mengetahui tingkat ketersediaan *server* di tiap-tiap skenario, apakah pada seluruh skenario *cluster* masih mendapat nilai availability yang masih berada di rentang 99,99 persen. Pengambilan data terhadap *throughput* bertujuan untuk mengetahui performa yang dapat disediakan oleh *cluster* berdasarkan tiap-tiap skenario. Parameter *Delay* didapat dengan dilakukan pengambilan data pada setiap skenario untuk mengetahui jeda waktu yang dibutuhkan oleh *cluster* untuk memberikan response kepada *client*. Parameter terakhir yang akan diambil yaitu *Resource Utilization* dengan melakukan data

logging menggunakan Grafana yang mana akan dilakukan analisa dampak dari setiap skenario terhadap penggunaan resource pada *cluster*.

Tabel 3. 6 Parameter Pengujian

Skenario	*Concurency*	*Availability* (%)	*Delay* (ms)	Throughput (MB/s)	*Resource Utilization*					
					CPU (%)			Memory (MB)		
					Man1	Man2	*Node*1	Man1	Man2	*Node*1
Skenario 1	500									
	600									
	700									
	800									
	900									
	1000									
Skenario 2	500					*Down*			*Down*	
	600									
	700									
	800									
	900									
	1000									
Skenario 3	500				*Down*			*Down*		
	600									
	700									
	800									
	900									
	1000									
Skenario 4	500					*Down*	*Down*		*Down*	*Down*
	600									
	700									
	800									
	900									
	1000									
Skenario 5	500				*Down*		*Down*	*Down*		*Down*
	600									
	700									
	800									
	900									
	1000									
Skenario 6	500						*Down*			*Down*
	600									
	700									
	800									
	900									
	1000									

Skenario	*Concurency*	*Availability* (%)	*Delay* (ms)	Throughput (MB/s)	*Resource Utilization*					
					CPU (%)			Memory (MB)		
					Man1	Man2	*Node*1	Man1	Man2	*Node*1
Skenario 7	500				*Down*	*Down*		*Down*	*Down*	
	600									
	700									
	800									
	900									
	1000									