

## **BAB III**

### **METODOLOGI PENELITIAN**

#### **3.1. PERANGKAT YANG DIGUNAKAN**

Pada penelitian ini menggunakan perangkat keras dan perangkat lunak yang dapat menunjang pembuatan web server *load balancer* pada *Kubernetes*.

##### **3.1.1 PERANGKAT KERAS (*HARDWARE*)**

Perangkat keras yang akan digunakan dalam penelitian ini menggunakan 2 komputer yang berfungsi sebagai *client* dan *server*. Spesifikasi yang digunakan untuk komputer server dan komputer *client* sebagaimana ditunjukkan pada tabel 3.1

**Tabel 3.1 Spesifikasi Perangkat Keras**

Jenis Perangkat	Jenis Parameter Perangkat	Spesifikasi
Server	<i>OS</i>	<i>Ubuntu Server 22.04 LTS</i>
	<i>Prosesor</i>	Intel Core™ i7-7700
	<i>RAM</i>	8 GB
	<i>Storage</i>	100 GB
Client	<i>OS</i>	<i>Ubuntu Server 22.04 LTS</i>
	<i>Prosesor</i>	Intel Core™ i7-7700
	<i>RAM</i>	8 GB
	<i>Storage</i>	100 GB

##### **3.1.2 PERANGKAT LUNAK (*SOFTWARE*)**

Pada penelitian ini menggunakan perangkat *virtual* dan *software tool* untuk menunjang pembuatan web server, *load balancer* dan *Kubernetes*.

###### **3.1.2.1 PERANGKAT VIRTUAL**

Pada penelitian ini terdapat 5 komputer *virtual* yaitu 1 komputer *virtual load balancer*, 2 komputer *virtual master* dan 2 komputer *virtual worker*. komputer *virtual master* dan komputer *virtual worker* akan dibangun pada *cluster Kubernetes*. Spesifikasi dari perangkat *virtual* sebagaimana ditunjukkan pada tabel 3.2

**Tabel 3.2 Spesifikasi Perangkat Virtual**

Jenis VM	Parameter VM	Spesifikasi VM
<i>Load Balancer</i> <i>Zevenet</i>	<i>OS</i>	<i>Ubuntu Server 22.04 LTS</i>
	<i>Core</i>	2 vCPU
	<i>RAM</i>	2 GB
	<i>Storage</i>	20 GB
	Alamat IP	10.212.20.254/24
<i>Master 1</i>	<i>OS</i>	<i>Ubuntu Server 22.04 LTS</i>
	<i>Core</i>	2 vCPU
	<i>RAM</i>	2 GB
	<i>Storage</i>	20 GB
	Alamat IP	10.212.20.242/24
<i>Master 2</i>	<i>OS</i>	<i>Ubuntu Server 22.04 LTS</i>
	<i>Core</i>	2 vCPU
	<i>RAM</i>	2 GB
	<i>Storage</i>	20 GB
	Alamat IP	10.212.20.244/24
<i>Worker 1</i>	<i>OS</i>	<i>Ubuntu Server 22.04 LTS</i>
	<i>Core</i>	2 vCPU
	<i>RAM</i>	2 GB
	<i>Storage</i>	20 GB
	Alamat IP	10.212.20.243/24
<i>Worker 2</i>	<i>OS</i>	<i>Ubuntu Server 22.04 LTS</i>
	<i>Core</i>	2 vCPU
	<i>RAM</i>	2 GB
	<i>Storage</i>	20 GB
	Alamat IP	10.212.20.245/24

### 3.1.2.2 SOFTWARE TOOL DAN APLIKASI

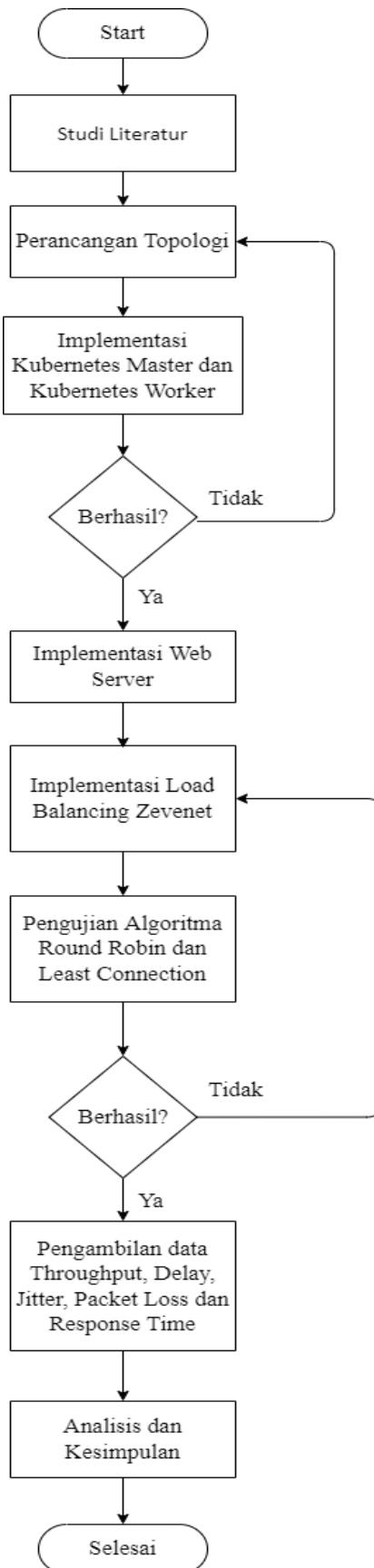
Perangkat lunak sebagai *tool* dan aplikasi yang digunakan pada penelitian ini sebagaimana ditunjukkan pada tabel 3.3

**Tabel 3.3 Tool dan Aplikasi**

No	Software	Versi	Fungsi
1	<i>Virtualbox</i>	6.1	Virtualisasi Komputer
2	<i>Kubernetes</i>	1.24.3	Mengelola <i>cluster</i> didalam <i>Docker</i>
3	<i>Wordpress</i>	6.0.1	Tampilan antar muka <i>website</i>
4	<i>MySQL</i>	8.0.29	<i>Database</i> untuk <i>wordpress</i>
5	<i>Zevenet</i>	5.12	<i>Load Balancer</i>
6	<i>Wireshark</i>	3.6.6	<i>Capture</i> paket data
7	<i>H2Load</i>	1.48.0	Aplikasi <i>request</i> Web Server

### **3.2. ALUR PENELITIAN**

Penelitian ini dilakukan melalui beberapa tahap yang ditujukan pada gambar diagram alur sebagaimana ditunjukkan pada gambar 3.1.



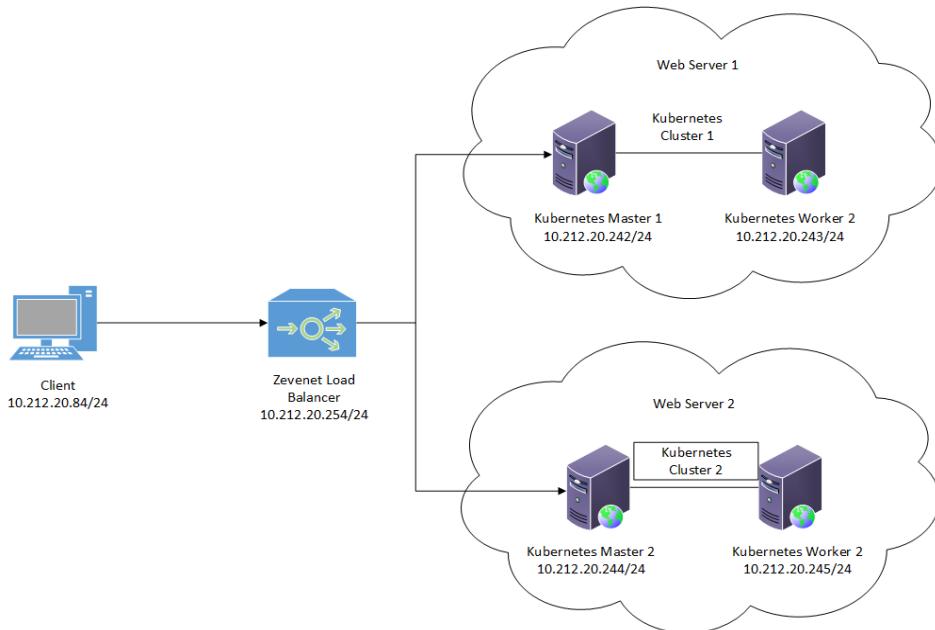
Gambar 3.1 Diagram Alur Penelitian

Gambar 3.1 menunjukkan diagram alur sistem pada penelitian ini. Dimulai dengan studi literatur dilanjutkan dengan perancangan topologi yang digunakan sebagai dasar dari arsitektur jaringan *Kubernetes*. Selanjutnya implementasi pada *virtual machine* didalam sistem *cluster Kubernetes* yang berisi *Kubernetes Master* dan *Kubernetes Worker* menggunakan OS *Ubuntu Server 22.04 LTS*. Jika berhasil maka lanjut ke tahap implementasi web server, jika tidak berhasil maka melakukan perancangan topologi kembali. Implementasi web server menggunakan aplikasi *wordpress* dengan menggunakan *mysql* sebagai *database*. Setelah web server berjalan selanjutnya Implementasi *Load Balancing Zevenet*, didalam *virtual machine zevenet* dikonfigurasikan algoritma *round robin* dan *least connection*. Setelah selesai konfigurasi algoritma pada *zevenet* dilanjutkan untuk menguji algoritma *round robin* dan *least connection* untuk mengakses web server. Jika berhasil maka akan dilanjutkan untuk pengambilan data, jika tidak berhasil maka kembali ke implementasi *load balancing*.

Tahap selanjutnya adalah melakukan pengambilan data pada tiap-tiap parameter yang diuji. Pada penelitian ini menggunakan *software tools h2load dan wireshark*. Parameter yang diuji dalam pemelitian ini yaitu *QoS* dan *Response Time*. Parameter tersebut dapat menggambarkan tingkat kecepatan dan kehandalan pada web server dalam menangani lalu lintas data. Setelah mendapatkan data dari parameter yang diuji dilanjutkan untuk menganalisis data dan memberikan kesimpulan untuk data yang sudah diperoleh untuk mengetahui performa dari unjuk kerja *load balancing* web server menggunakan *Kubernetes*.

### 3.3. TOPOLOGI JARINGAN

Topologi yang digunakan dalam penelitian ini menggunakan 5 komputer *virtual*, dan satu komputer sebagai *client*. Pada komputer *virtual* terdapat 2 *cluster* yang berisi 2 *Kubernetes master* dan 2 *Kubernetes worker*. 1 komputer *virtual* sebagai *load balancer Zevenet*. Pada *cluster Kubernetes* yang berisi 1 *master* dan 1 *worker* akan di implementasikan *wordpress* sebagai web server. Pada *zevenet* digunakan untuk mengatur beban trafik yang masuk ke web server berdasarkan algoritma *least connection* dan *round robin*. Uraian tersebut sebagaimana ditunjukkan pada gambar 3.2.



**Gambar 3.2 Topologi jaringan**

### 3.4. IMPLEMENTASI KUBERNETES MASTER DAN WORKER

Pada Implementasi *Kubernetes master* dan *worker*, spesifikasi dari kedua perangkat tersebut dapat dilihat pada tabel 3.2. *Operating System* yang digunakan menggunakan *Ubuntu Server 22.04 LTS*. Konfigurasi dimulai *install ubuntu server* pada vm dengan alamat ip 10.212.20.242/24 untuk *master1*, 10.212.20.243 untuk *worker1*, 10.212.20.244 untuk *master2* dan 10.212.20.245 untuk *worker2*. Setelah instalasi berhasil selanjutnya *remote ssh* ke vm *master1*, *master2*, *worker1* dan *worker2*.

```
labpsd@labpsd-H110M-DS2:~$ sudo su
[sudo] password for labpsd:
root@labpsd-H110M-DS2:/home/labpsd#                                         ssh
master1@10.212.20.242
The authenticity of host '10.212.20.242 (10.212.20.242)' can't be established.
ED25519          key          fingerprint      is
SHA256:to/fnGTs6AJxVhoRdm0CVSBYZsPIK+/1q784bXYhHmA.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.212.20.242' (ED25519) to the list of known hosts.
master1@10.212.20.242's password:
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-41-generic x86_64)
```

Untuk konfigurasi *ssh* ke *master2*, *worker1* dan *worker2* pada *textbox* diatas, hanya mengganti *username* dan *ip address* pada *command line interface*. Setelah berhasil masuk ke vm pada masing-masing terminal selanjutnya *update ubuntu server* dengan perintah:

```
root@master1:/home/master1# apt update;apt upgrade -y;apt autoremove -y
Hit:1 http://id.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://id.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://id.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://id.archive.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
```

Pastikan *ip address* sudah bersifat *static* dengan perintah menggunakan *text editor vim*.

```
root@master1:/home/master1# vim /etc/netplan/00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      addresses:
        - 10.212.20.242/24
      gateway4: 10.212.20.1
      nameservers:
        addresses:
          - 10.212.20.1
        search: []
  version: 2
root@master1:/home/master1# netplan apply
```

*IP Address* untuk *master2*, *worker1*, *worker2* dan *zevenet* terdapat pada tabel 3.2. Selanjutnya menambahkan *IP Address host* untuk *master2*, *worker1*, *worker2* dan *zevenet* dengan perintah:

```
root@master1:/home/master1# vim /etc/hosts
127.0.0.1 localhost
127.0.1.1 master1
10.212.20.242 master1
10.212.20.243 worker1
10.212.20.244 master2
10.212.20.245 worker2
10.212.20.254 zevenet-lb
# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Setelah *IP Address* didaftarkan pada *textbox* diatas, selanjutnya *install microk8s* pada masing masing vm kecuali vm *zevenet* dengan perintah:

```
root@master1:/home/master1# snap install microk8s --classic --channel=1.24/edge
microk8s (1.24/edge) v1.24.3 from Canonical✓ installed
root@master1:/home/master1# usermod -a -G microk8s $USER
root@master1:/home/master1# chown -f -R $USER ~/.kube
root@master1:/home/master1# su - $USER
root@master1:~#
```

Perintah *usermod* dan *chown* digunakan untuk mendaftarkan perintah *microk8s* untuk bisa diakses diluar akses *root*. Perintah *su* digunakan untuk masuk ulang sesi *root*. Perintah untuk memulai *microk8s* beserta statusnya sebagai berikut:

```
root@master1:/home/master1# microk8s start
Started.
root@master1:/home/master1#
root@master1:/home/master1# microk8s status
microk8s is running
high-availability: no
  datastore master nodes: 127.0.0.1:19001
  datastore standby nodes: none
```

Setelah *microk8s* berjalan, tahap selanjutnya menambahkan *node worker1* pada *master1*. Sebelum ditambahkan *master1* harus menampilkan *token* supaya *worker1* bisa *join* sebagai *worker* dengan perintah:

```

root@master1:/home/master1# microk8s add-node
From the node you wish to join to this cluster, run the
following:
microk8s                                         join
10.212.20.242:25000/74944b692c03de56231748b7a6834ec7/7ae5
0a8e706d
Use the '--worker' flag to join a node as a worker not
running the control plane, eg:
microk8s                                         join
10.212.20.242:25000/74944b692c03de56231748b7a6834ec7/7ae5
0a8e706d --worker
If the node you are adding is not reachable through the
default interface you can use one of the following:
microk8s                                         join
10.212.20.242:25000/74944b692c03de56231748b7a6834ec7/7ae5
0a8e706d

```

Selanjutnya pada *worker1* pada *textbox* dibawah harus memasukkan perintah alamat ip beserta *token* yang sudah ditampilkan dari *master1* dengan *--worker* diakhir perintah.

```

root@worker1:/home/worker1#                         microk8s      join
10.212.20.242:25000/74944b692c03de56231748b7a6834ec7/7ae5
0a8e706d --worker
Contacting cluster at 10.212.20.242
The node has joined the cluster and will appear in the
nodes list in a few seconds.
Currently this worker node is configured with the following
kubernetes API server endpoints:
    - 10.212.20.242 and port 16443, this is the cluster
node contacted during the join operation.
root@worker1:/home/worker1#

```

Pada *textbox* diatas, *worker1* sudah berhasil *join* ke *master1* sebagai *worker*, ditandai dengan *the node has joined the cluster*. Untuk memverifikasi *worker1* yang sudah *join cluster* sebagai *worker* pada *master* menggunakan perintah:

```

root@master1:/home/master1# microk8s kubectl get node
NAME|STATUS|ROLES|AGE|VERSION
master1|Ready|<none>|30m|v1.24.3-2+63243a96d1c393
worker1|Ready|<none>|103s|v1.24.3-2+63243a96d1c393

```

Setelah memverifikasi *worker1* berhasil *join cluster* dengan *master1*, maka *worker1* tidak bisa melakukan perintah pada *Kubernetes*.

```

root@worker1:/home/worker1# microk8s start
This MicroK8s deployment is acting as a node in a cluster.
Use 'snap start microk8s' to start services on this node.

```

Pada *master1* akan menjalankan *service dns, dashboard* dan *rbac* untuk menunjang implementasi web server.

```
root@master1:/home/master1# microk8s enable dns dashboard
rbac
Infer repository core for addon dns
Infer repository core for addon dashboard
Infer repository core for addon rbac
```

Setelah menjalankan perintah *enable*, *service* akan berjalan secara berurutan. Untuk memverifikasi *service* sudah berjalan digunakan perintah:

```
root@master1:/home/master1# microk8s status
microk8s is running
high-availability: no
  datastore master nodes: 10.212.20.242:19001
  datastore standby nodes: none
addons:
  enabled:
    dashboard # (core) The Kubernetes dashboard
    dns # (core) CoreDNS
    ha-cluster # (core) Configure high availability on the
    current node
    metrics-server # (core) K8s Metrics Server for API access
    to service metrics
    rbac # (core) Role-Based Access Control for authorisation
```

*Service dns, dashboard* dan *rbac* sudah berjalan pada *microk8s*. Setelah sampai pada proses ini maka implementasi *Kubernetes* sudah berhasil

### 3.5. IMPLEMENTASI WEB SERVER

Dalam penelitian ini menggunakan 2 web server yang di implementasikan pada 2 *cluster Kubernetes* yang berbeda. Penelitian ini menggunakan *Wordpress* sebagai web server dan *mysql* sebagai *database* nya. *Wordpress* yang digunakan memakai versi 6.0.1 dan *mysql* menggunakan versi 8.0.29. Untuk menjalankan *wordpress* pada *Kubernetes* ada beberapa kriteria konfigurasi parameter yang harus dipenuhi seperti *persistent volume(storage)*, *persistent volume claim* untuk mengklaim dari *storage*, *service* untuk memberikan alamat ip dan *port* pada *wordpress* dan *deployment* untuk menjalankan *wordpress*, begitu juga dengan *mysql* memerlukan kriteria konfigurasi seperti *wordpress*. Untuk membuat konfigurasi *persistent volume* dan *volume claim* dari *wordpress* menggunakan perintah:

```

root@master1:/home/master1# vim wp-pv-pvc.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: wordpress-pv
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 3Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/root/blid/wordpress-config/wordpress-data"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wordpress-pvc
  labels:
    app: wordpress
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi

```

Pada *textbox* diatas, *Persistent volume* akan membuat *storage local* dari vm tersebut dengan membuat direktori pada *"/root/blid/wordpress-config/wordpress-data"* sebesar 3 GiB dengan mode akses *ReadWriteOnce*. Sedangkan *persistent volume claim* akan mengklaim storage dari *persistent volume* dengan mode *ReadWriteOnce* sebesar 3 GiB. Selanjutnya membuat *config* parameter *service* untuk web server dengan perintah:

```

root@master1:/home/master1# vim wp-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: wordpress-svc
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30001

```

```
selector:  
  app: wordpress  
  tier: frontend  
  type: NodePort
```

Pada *textbox* diatas, *Service* akan membuat *port* dengan tipe *NodePort* pada *Kubernetes* dengan nomor *port* 80 sebagai nomor *port* didalam *cluster Kubernetes*, target *port* dengan nomor 80 sebagai nomor *port* untuk *wordpress* dan *nodeport* 30001 sebagai *port* eksternal dari *cluster Kubernetes*. Untuk membuat *config* parameter *deployment* digunakan perintah:

```
root@master1:/home/master1# vim wp-deployment.yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: wordpress-deploy  
  labels:  
    app: wordpress  
spec:  
  selector:  
    matchLabels:  
      app: wordpress  
      tier: frontend  
  strategy:  
    type: Recreate  
  template:  
    metadata:  
      labels:  
        app: wordpress  
        tier: frontend  
    spec:  
      containers:  
      - image: wordpress:latest  
        name: wordpress  
        env:  
        - name: WORDPRESS_DB_HOST  
          value: wordpressdb-svc  
        - name: WORDPRESS_DB_PASSWORD  
          value: bismillah123  
        - name: WORDPRESS_DB_USER  
          value: dika  
        - name: WORDPRESS_DB_NAME  
          value: dika  
      ports:  
      - containerPort: 80  
        name: wordpress-svc  
      volumeMounts:  
      - name: wordpress-persistent-storage  
        mountPath: /var/www/html
```

```

volumes:
- name: wordpress-persistent-storage
  persistentVolumeClaim:
    claimName: wordpress-pvc

```

Pada *textbox* diatas *Deployment* akan membuat *wordpress* dengan *image latest*(6.0.1), *image* tersebut akan di *download* dari *docker hub*. *Environment* yang digunakan *wordpress\_db\_host* adalah *wordpressdb-svc*, *wordpress\_db\_password* adalah bismillah123, *wordpress\_db\_user* adalah dika, *wordpress\_db\_name* adalah dika. *Wordpress* akan menggunakan *service wordpress-svc* yang sudah dibuat dan akan mengklaim *wordpress-pvc* yang sudah dibuat. Selanjutnya membuat *config* parameter untuk *database mysql* menggunakan perintah:

```

root@master1:/home/master1# vim wpdb-pv-pvc.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: wordpressdb-pv
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 3Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/root/blid/wordpress-config/wordpressdb-data"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wordpressdb-pvc
  labels:
    app: wordpress
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi

```

Pada *textbox* diatas *Persistent volume* akan membuat *storage local* dari vm tersebut dengan membuat direktori pada "/root/blid/wordpress-config/wordpressdb-data" sebesar 3 GiB dengan mode akses *ReadWriteOnce*. Sedangkan *persistent volume claim* akan mengklaim *storage* dari *persistent volume* dengan mode

*ReadWriteOnce* sebesar 3 GiB. Selanjutnya konfigurasi *service* untuk *database mysql* dengan perintah:

```
root@master1:/home/master1# vim wpdb-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: wordpressdb-svc
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
      protocol: TCP
      targetPort: 3306
  selector:
    app: wordpress
    tier: wordpressdb
  type: ClusterIP
  clusterIP: None
```

Pada *textbox* diatas, *Service* pada *mysql* akan membuat *port* dengan nomor 3306 pada internal *cluster Kubernetes* dan target *port* 3306 untuk aplikasi *database mysql*. *Protocol* yang digunakan adalah *TCP*. *Mysql* menggunakan *clusterip* dan tidak menggunakan *ip address*. Selanjutnya adalah membuat konfigurasi parameter untuk *mysql deployment* dengan perintah:

```
root@master1:/home/master1# vim wpdb-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpressdb-deploy
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: wordpressdb
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: wordpressdb
    spec:
      containers:
```

```

- image: mysql:latest
  name: wordpressdb-container
  env:
    - name: MYSQL_ROOT_PASSWORD
      value: bismillah123
    - name: MYSQL_USER
      value: dika
    - name: MYSQL_PASSWORD
      value: bismillah123
    - name: MYSQL_DATABASE
      value: dika
  ports:
    - containerPort: 3306
      name: wordpressdb-svc
  volumeMounts:
    - name: wordpressdb-persistent-storage
      mountPath: /var/lib/mysql
  volumes:
    - name: wordpressdb-persistent-storage
      persistentVolumeClaim:
        claimName: wordpressdb-pvc

```

Pada *textbox* diatas, *Deployment* akan membuat Mysql dengan *image latest(8.0.29)*, *image* tersebut akan didownload dari *docker hub*. *Environment* yang digunakan *mysql\_root\_password* adalah bismillah123, *mysql\_user* adalah dika, *mysql\_password* adalah bismillah123, *mysql\_database* adalah dika. Mysql akan menggunakan *service wordpressdb-svc* yang sudah dibuat dan akan mengklaim *wordpressdb-pvc* yang sudah dibuat. Untuk memanggil semua konfigurasi *file* dalam satu perintah, terdapat fitur *kustomization*.

```

root@master1:/home/master1# vim kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
resources:
- wpdb-pv-pvc.yaml
- wpdb-svc.yaml
- wpdb-deployment.yaml
- wp-pv-pvc.yaml
- wp-svc.yaml
- wp-deployment.yaml

```

Pada *textbox* diatas, *file Kustomization.yaml* akan memanggil semua *file* yang terdaftar, dan menjalankannya dengan satu perintah:

```
root@master1:/home/master1# microk8s kubectl apply -k ./service/wordpress-svc createdservice/wordpressdb-svc createdpersistentvolume/wordpress-pv createdpersistentvolume/wordpressdb-pv createdpersistentvolumeclaim/wordpress-pvc createdpersistentvolumeclaim/wordpressdb-pvc createddeployment.apps/wordpress-deploy createddeployment.apps/wordpressdb-deploy created
```

Pada *textbox* diatas, *Kustomization.yaml* sudah dijalankan dan sudah memanggil semua *file* yang sudah dibuat. Selanjutnya untuk memastikan proses instalasi dari *wordpress* menggunakan perintah:

```
root@master1:/home/master1# microk8s kubectl get pods --all-namespaces -wNAMESPACE | NAME | READY | STATUS | RESTARTS | AGEdefault | wordpressdb-deploy-7dbfd765b8-nginx | 1/1 | Running | 0 | 47sdefault | wordpress-deploy-6f4466bccd-g88r7 | 1/1 | Running | 0 | 52s
```

Pada *texrbox* diatas, proses dengan nama *wordpressdb-deploy* dan *wordpress-deploy* menunjukan status 1/1 yang artinya instalasi *wordpress* dan *database* sudah berhasil. Untuk memeriksa tempat dari instalasi *wordpress* dan *database* menggunakan perintah:

```
root@master1:/home/master1# microk8s kubectl get pods -owideNAME | READY | STATUS | RESTARTS | AGE | IP | NODE | NOMINATED-NODE | READINESS-GATESwordpressdb-deploy-7dbfd765b8-nginx | 1/1 | Running | 0 | 92s | 10.1.235.132 | worker1 | <none> | <none>wordpress-deploy-6f4466bccd-g88r7 | 1/1 | Running | 0 | 92s | 10.1.137.69 | master1 | <none> | <none>
```

Pada *textbox* diatas, *Wordpress* terletak pada *node master1* sedangkan *wordpressdb* terletak pada *node worker1*. Untuk mengembangkan *wordpress* supaya tersedia juga pada *node worker1* menggunakan perintah *scale*:

```
root@master1:/home/master1# microk8s kubectl scale --replicas=2 deployment.apps/wordpress-deploy deployment.apps/wordpress-deploy scaled
```

Pada *textbox* diatas, *wordpress* sudah berkembang menjadi dua *node*. Untuk memeriksa kembali menggunakan perintah:

```
root@master1:/home/master1# microk8s kubectl get pods -o wide
NAME | READY | STATUS | RESTARTS | AGE | IP | NODE | NOMINATED-NODE | READINESS-GATES
wordpressdb-deploy-7dbfd765b8-ngfsx | 1/1 | Running | 0 | 92s | 10.1.235.132 | worker1 | <none> | <none>
wordpress-deploy-6f4466bccd-g88r7 | 2/2 | Running | 0 | 92s | 10.1.137.69 | master1 | <none> | <none>
```

Pada *textbox* diatas, aplikasi *wordpress* sudah berhasil *scale* menjadi 2. Untuk memeriksa informasi *Kubernetes* secara keseluruhan menggunakan perintah:

```
root@master1:/home/master1# microk8s kubectl get all
NAME | READY | STATUS | RESTARTS | AGE
pod/wordpressdb-deploy-7dbfd765b8-ngfsx | 1/1 | Running | 0 | 4m39s
pod/wordpress-deploy-6f4466bccd-g88r7 | 1/1 | Running | 0 | 4m39s
pod/wordpress-deploy-7a3686adhj-r49b6 | 1/1 | Running | 0 | 5m10s

NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT (S) | AGE
service/Kubernetes | ClusterIP | 10.152.183.1 | <none> | 443/TCP | 5m
service/wordpress-
svc | NodePort | 10.152.183.200 | <none> | 80:30001/TCP | 4m39s
service/wordpressdb-
svc | ClusterIP | None | <none> | 3306/TCP | 4m39s

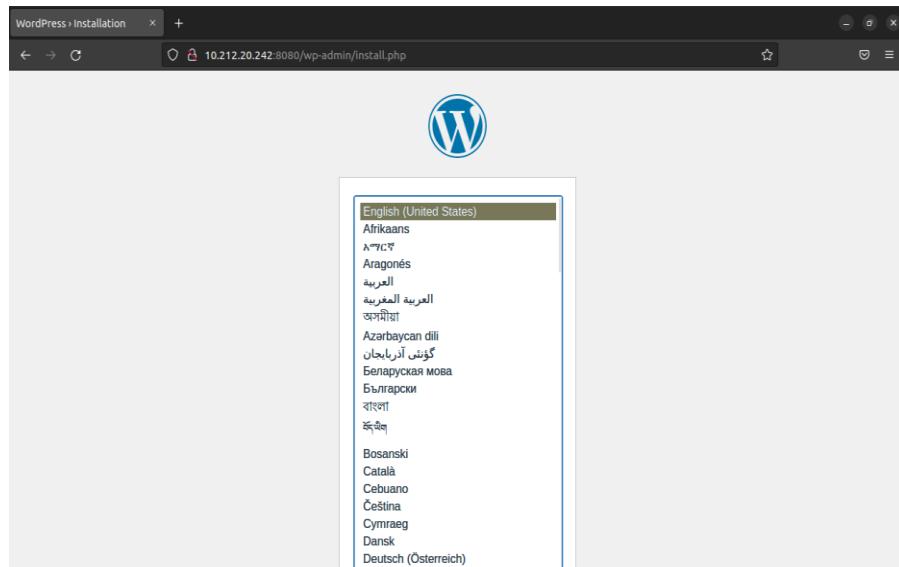
NAME | READY | UP-TO-DATE | AVAILABLE | AGE
deployment.apps/wordpressdb-deploy | 1/1 | 1 | 1 | 4m39s
deployment.apps/wordpress-deploy | 2/2 | 1 | 1 | 4m39s

NAME | DESIRED | CURRENT | READY | AGE
replicaset.apps/wordpressdb-deploy-7dbfd765b8 | 1 | 1 | 1 | 4m39s
replicaset.apps/wordpress-deploy-6f4466bccd | 2 | 2 | 2 | 5m10s
```

Pada *textbox* diatas, menampilkan informasi *pod*, *service*, *deployment* dan *replica set*. Untuk mengakses *wordpress* supaya muncul tampilan web, harus menggunakan perintah *port forwarding*:

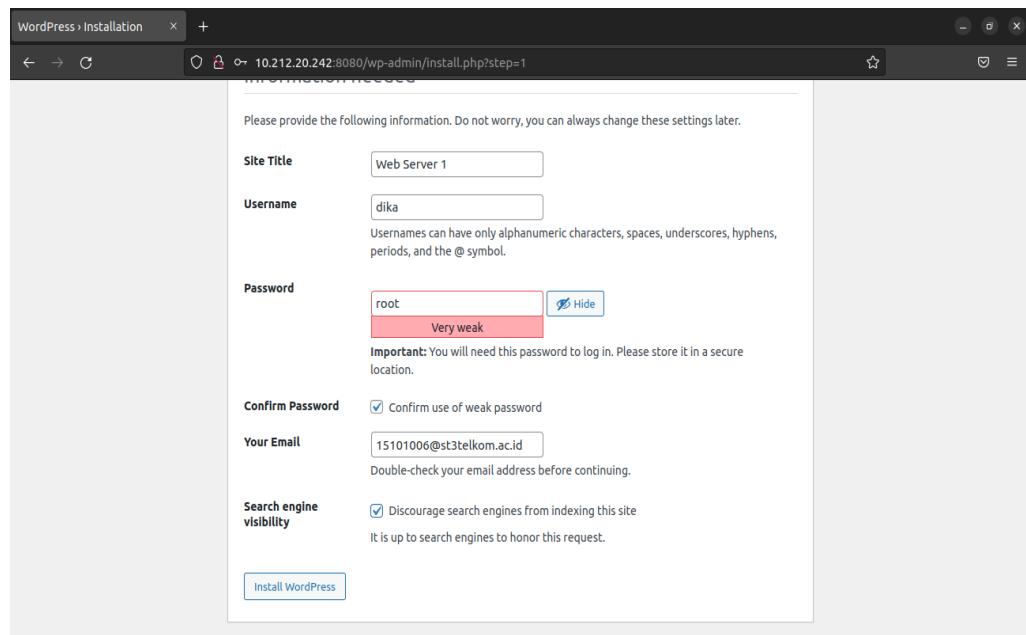
```
root@master1:/home/master1# microk8s kubectl port-forward service/wordpress-svc 8080:80 --address=10.212.20.242
Forwarding from 10.212.20.242:8080 -> 80
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
```

Selanjutnya akses alamat ip 10.212.20.242 dengan nomor *port* 8080 pada web browser, maka akan muncul tampilan awal *setup wordpress*.



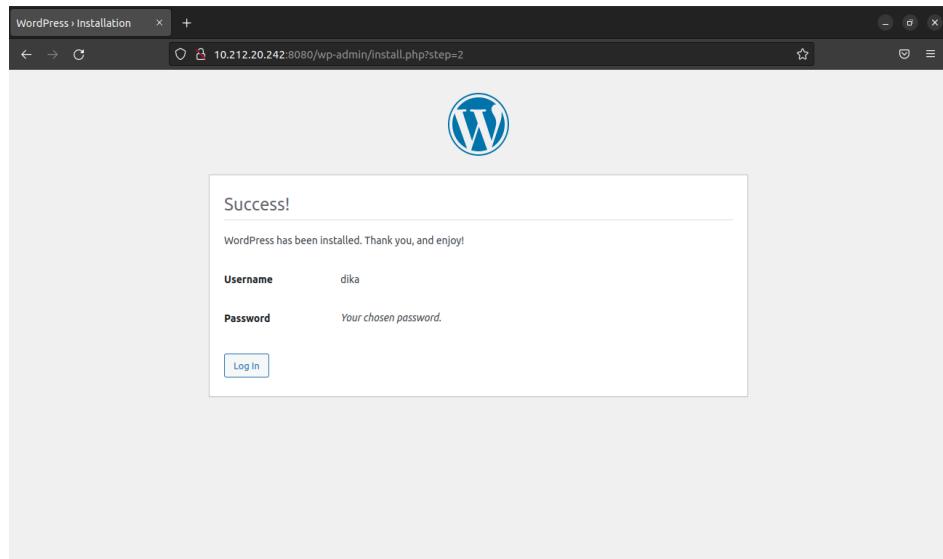
**Gambar 3.3 Tampilan awal pendaftaran *wordpress***

Pada gambar 3.3 menunjukan halaman pertama untuk memilih Bahasa pada halaman *wordpress*.



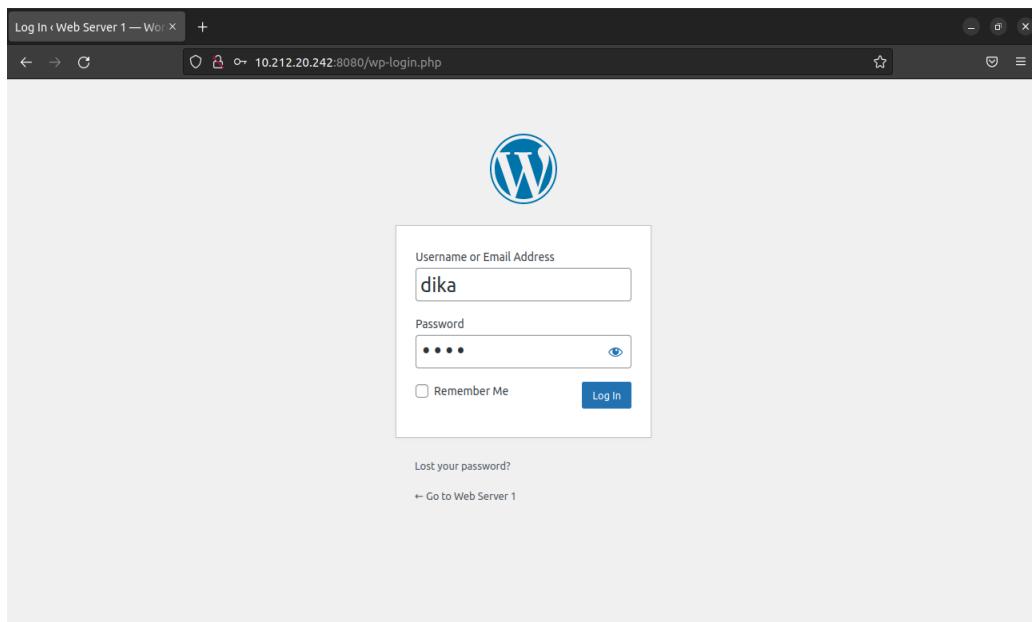
**Gambar 3.4 Tampilan registrasi *wordpress***

Pada gambar 3.4 berisi halaman pendaftaran akun pada *wordpress*. Data yang diisi adalah nama dari situs web, *username*, *password*, dan *email*.



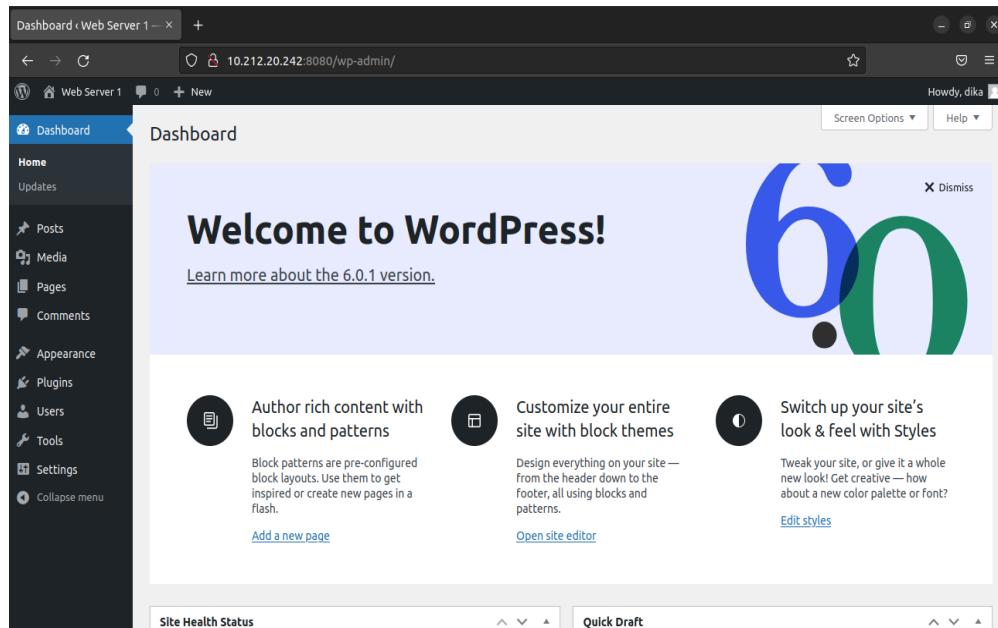
**Gambar 3.5 Tampilan registrasi berhasil pada *wordpress***

Pada Gambar 3.5 menunjukan pendaftaran akun *wordpress* telah berhasil dilakukan. Setelah berhasil dalam tahap registrasi, selanjutnya akan diminta untuk memasukkan *username* dan *password* yang sudah dibuat.



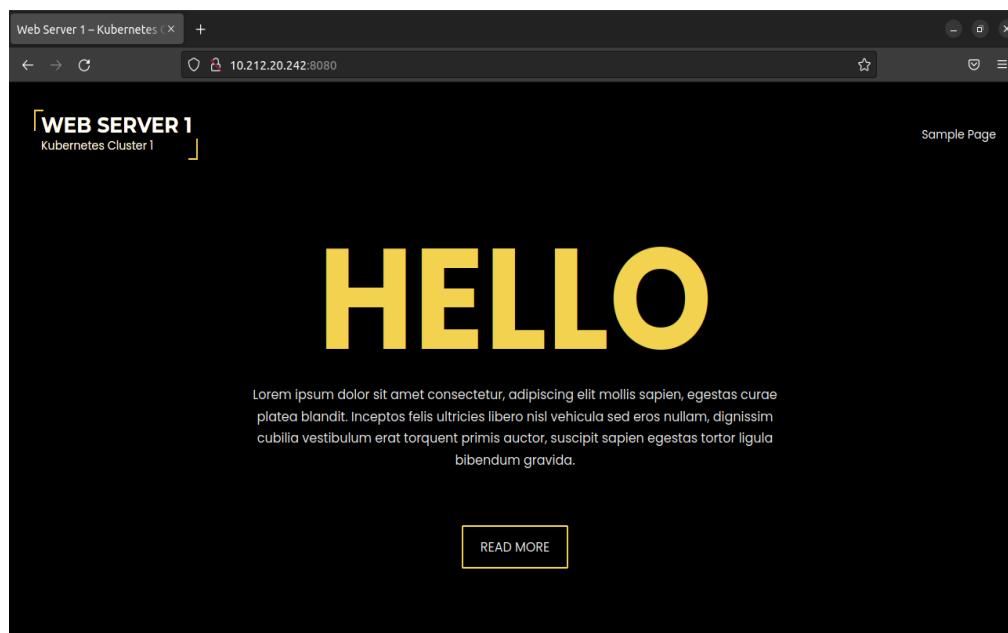
**Gambar 3.6 Tampilan *login wordpress***

Pada Gambar 3.6 menunjukan halaman *login* pada *wordpress*, *username* dan *password* yang digunakan berdasarkan halaman pendaftaran pada gambar 3.4.



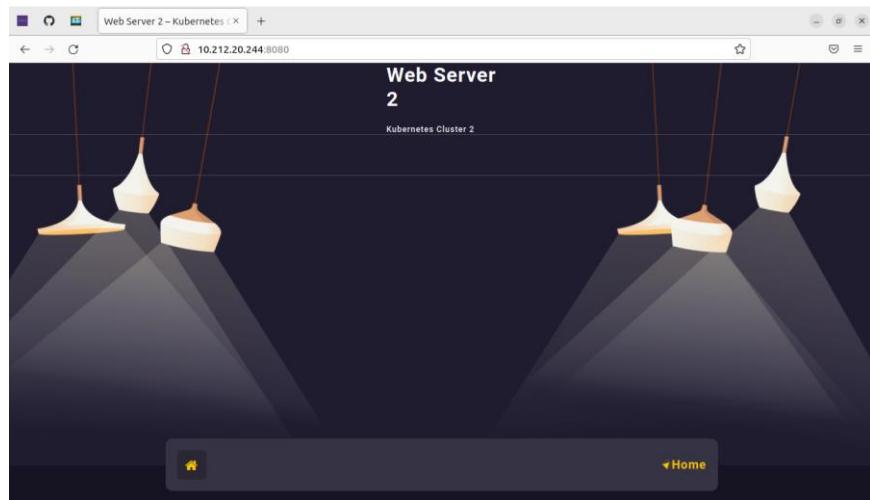
**Gambar 3.7 Tampilan *dashboard wordpress***

Pada Gambar 3.7 menunjukkan tampilan *dashboard* dari *wordpress*. Pada halaman ini terdapat dapat digunakan untuk berbagai macam kegiatan seperti mengganti tema, menambahkan plugin, menulis pada web dan lain lain.



**Gambar 3.8 Tampilan web server 1**

Pada Gambar 3.8 menunjukkan tampilan web server satu pada *Kubernetes cluster* satu. Setelah tampilan pada web server satu dan web server dua muncul maka implementasi web server sudah berhasil.

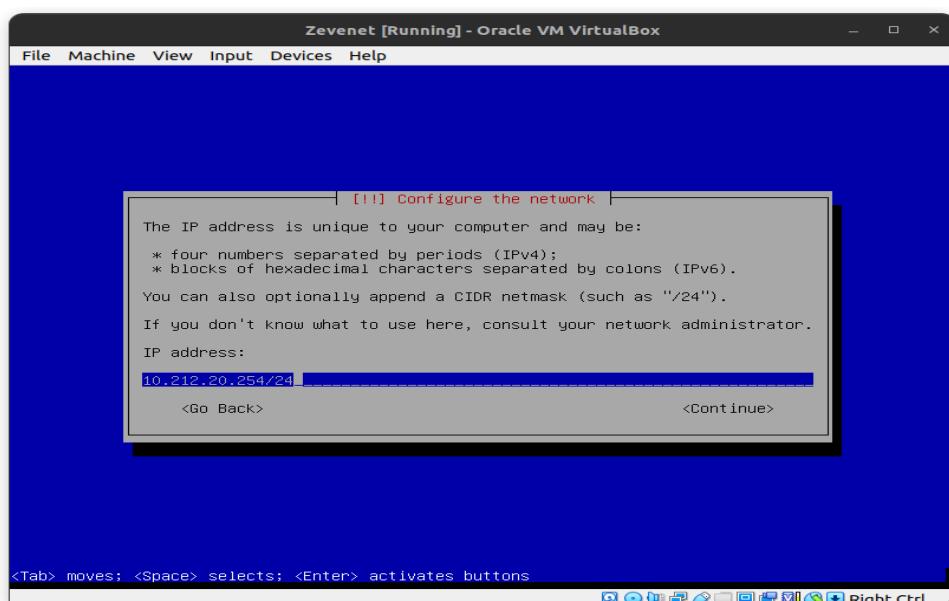


**Gambar 3.9 Tampilan web server 2**

Pada Gambar 3.9 menunjukan tampilan web server dua pada *Kubernetes cluster* dua, dengan alamat ip 10.212.20.244 dengan *port* 8080.

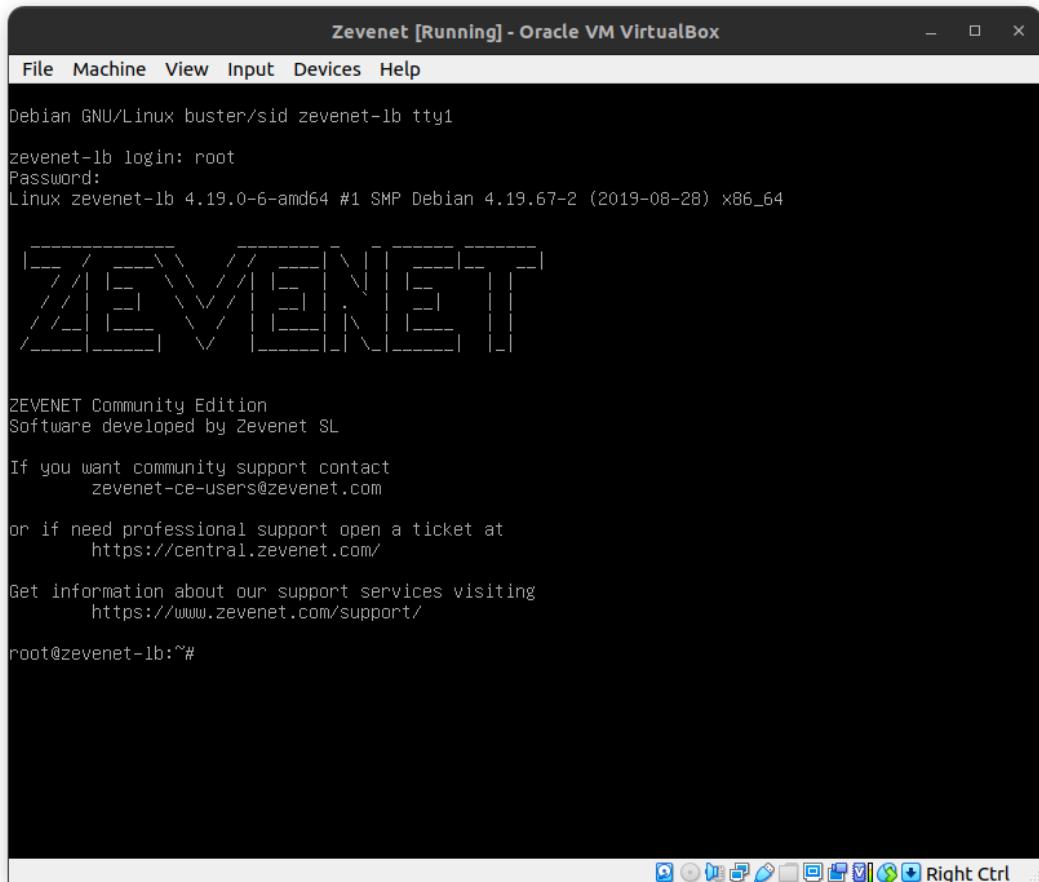
### **3.6. IMPLEMENTASI LOAD BALANCING ZEVENET**

Konfigurasi sistem *Load Balancing* menggunakan *Zevenet* serta system operasi menggunakan Debian buster. Algoritma yang digunakan dalam peneltian ini yaitu *round robin* dan *least connection*. Konfigurasi dimulai dengan *install file ISO Image* pada *virtualbox*, spesifikasi yang digunakan pada vm ini terdapat pada tabel 3.2. konfigurasi dimulai dengan memilih Bahasa dan region *install*, setelah itu memasukkan *ip address* 10.212.20.254/24 seperti pada gambar 3.10



**Gambar 3.10 Mengisi alamat ip pada zevenet**

Selanjutnya memasukkan default gateway dengan alamat ip 10.212.20.1 dengan name server dengan alamat ip 10.212.20.1. Langkah selanjutnya memasukkan *hostname zevenet-lb* dan membuat password untuk mengakses vm dan masuk ke *dashboard zevenet*. Selanjutnya memilih partisi yang digunakan. Setelah itu menunggu proses instalasi dari *zevenet*. Setelah selesai akan muncul tampilan dari vm *zevenet-lb* seperti sebagaimana ditunjukkan pada gambar 3.11



**Gambar 3.11 Tampilan dari vm *zevenet***

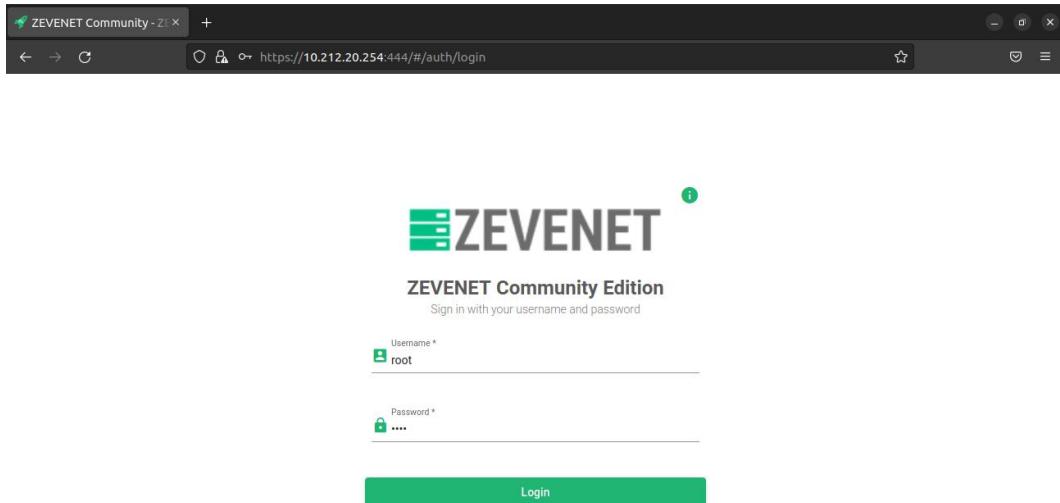
Untuk mendaftarkan *host* ke vm *zevenet-lb* menggunakan perintah:

```
root@zevenet-lb:~# nano /etc/hosts
127.0.0.1      localhost
10.212.20.242    master1
10.212.20.243    worker1
10.212.20.244    master2
10.212.20.245    worker2
# The following lines are desirable for IPv6 capable hosts
::1      localhost ip6-localhost ip6-loopback
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters
```

Pada *textbox* diatas, semua *host* sudah terdaftar pada *zevenet*. Untuk *update* dari *zevenet-lb* menggunakan perintah:

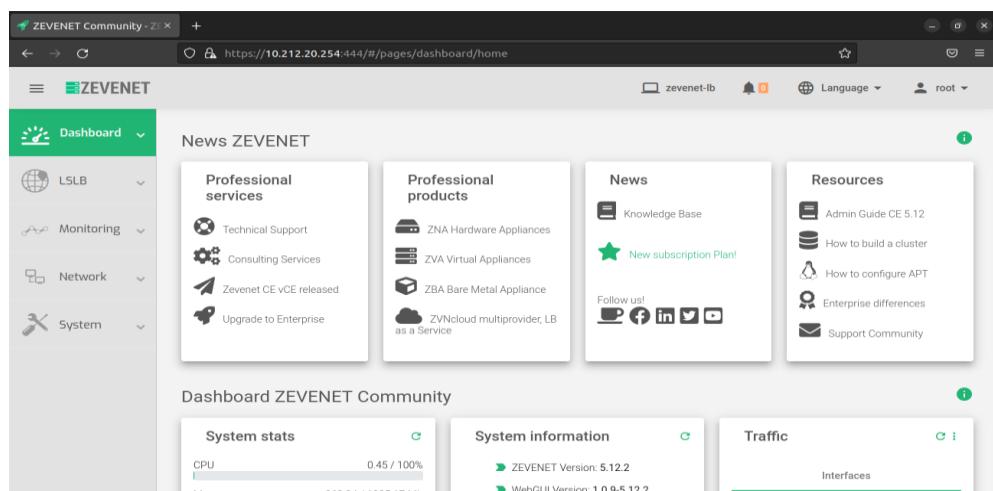
```
root@zevenet-lb:~# apt update;apt upgrade -y;apt autoremove -y  
root@zevenet-lb:~# /usr/local/zevenet/bin/checkupdates -i
```

Setelah perintah dijalankan maka *zevenet* sudah *update* menjadi versi terbaru. Masukkan alamat <https://10.212.20.254:444/> Untuk mengakses *dashboard* dari *zevenet* menggunakan web browser, maka akan muncul tampilan web dari *zevenet* seperti pada gambar 3.12



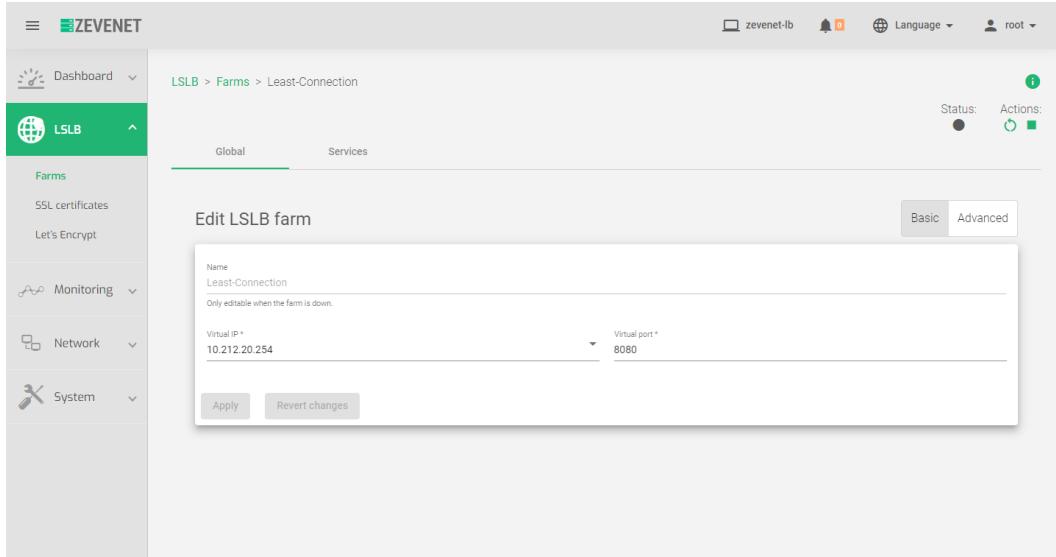
**Gambar 3.12 Tampilan *login zevenet***

Masukkan *username root* dan *password root* seperti pada saat instalasi vm *zevenet-lb*. Maka akan masuk ke tampilan utama dari *zevenet* seperti pada gambar 3.13.



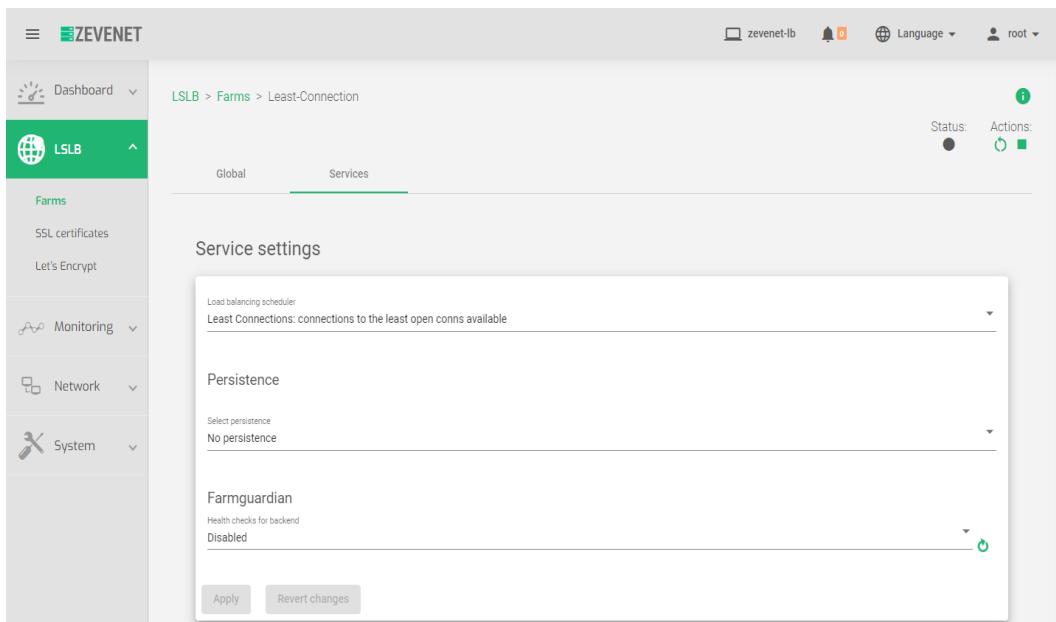
**Gambar 3.13 Tampilan *dashboard zevenet***

Pada gambar 3.13 menunjukan proses implementasi *zevenet* sudah berhasil. Untuk membuat konfigurasi algoritma *least connection* dan *round robin* pada *zevenet*, masuk ke menu *lslb* sebagaimana ditunjukkan pada gambar 3.14



**Gambar 3.14 Membuat konfigurasi *load balancing***

Pada gambar 3.14 berikan nama profil untuk menentukan algoritma dari *load balancing* yang akan digunakan, selanjutnya pilih *virtual ip* yang sudah didapat dari proses instalasi vm *zevenet* yaitu 10.212.20.254. masukkan *port* 8080 dan profil l4xnat. Pada menu *service* pilih tipe *load balancing* yang akan digunakan seperti *least connection* atau *round robin* sebagaimana ditunjukkan pada gambar 3.15



**Gambar 3.15 Konfigurasi service *load balancing* pada *zevenet***

Selanjutnya masukkan ip web server satu dan web server dua pada menu *backend* seperti pada gambar 3.16.

The screenshot shows the Zevenet LSLB interface. On the left, there's a sidebar with options like 'Toggle menu', 'Dashboard', 'Farms', 'SSL certificates', 'Let's Encrypt', 'Monitoring', 'Network', and 'System'. The main area is titled 'Backend' and shows a table of configured backends. The table has columns: IP, Port, Max. Conn, Priority, Weight, and Status. There are two entries: one for IP 10.212.20.242 on port 8080 with priority 1 and weight 1, and another for IP 10.212.20.244 on port 8080 with priority 1 and weight 1. Both entries have green status dots. At the top of the 'Backends' section, there are buttons for 'Create backend', 'Enable maintenance (drain mode)', 'Enable maintenance (cut mode)', and 'Disable maintenance'. There are also 'Apply' and 'Revert changes' buttons. A 'Health checks for backend' section at the top says 'Disabled' with a green switch icon. A search bar and pagination controls (1-2 of 2, items per page: 10) are also present.

**Gambar 3.16 Konfigurasi *backend***

Gambar 3.16 menunjukan web server satu dengan alamat ip 10.212.20.242 *port* 8080 dan web server dua dengan alamat ip 10.212.20.244 *port* 8080 sudah berhasil didaftarkan.

### **3.7. PENGUJIAN ALGORITMA ROUND ROBIN DAN LEAST CONNECTION**

Pengujian ini menggunakan aplikasi *h2load* untuk mengirimkan sejumlah *request* ke web server1 dan web server2. Alamat ip 10.212.20.254:8080 digunakan untuk mengakses *load balancer*, selanjutnya *load balancer* akan meneruskan *request* tersebut ke web server1 dan web server2. Skenario yang digunakan adalah aplikasi *h2load* akan mengirimkan sejumlah *request* dan aplikasi *wireshark* akan men *capture* paket yang dikirim maupun diterima pada *client*.

**Tabel 3.4 Skenario Pengujian data**

Skenario	Algoritma	Jumlah <i>Request</i>	Total Koneksi Bersamaan	Jumlah Percobaan	Parameter
1	Least Connection	1000	100	10	QoS dan Response Time
	Round Robin				
2	Least Connection	2000	100	10	QoS dan Response Time
	Round Robin				
3	Least Connection	4000	100	10	QoS dan Response Time
	Round Robin				
4	Least Connection	6000	100	10	QoS dan Response Time
	Round Robin				

Pada penelitian ini menggunakan 4 skenario variasi jumlah *request* sebesar 1000, 2000, 4000 dan 6000 jumlah *request*. dengan 100 total koneksi bersamaan setiap variasi jumlah *request*. Pada setiap variasi jumlah *request* menggunakan algoritma *least connection* dan *round robin*. Setiap jumlah *request* yang dikirim akan dilakukan pengulangan sebanyak 10 kali setiap variasi jumlah *request*. Data yang diambil menggunakan parameter *QoS* yaitu *throughput*, *delay*, *jitter* dan *packet loss*, serta parameter *response time*.

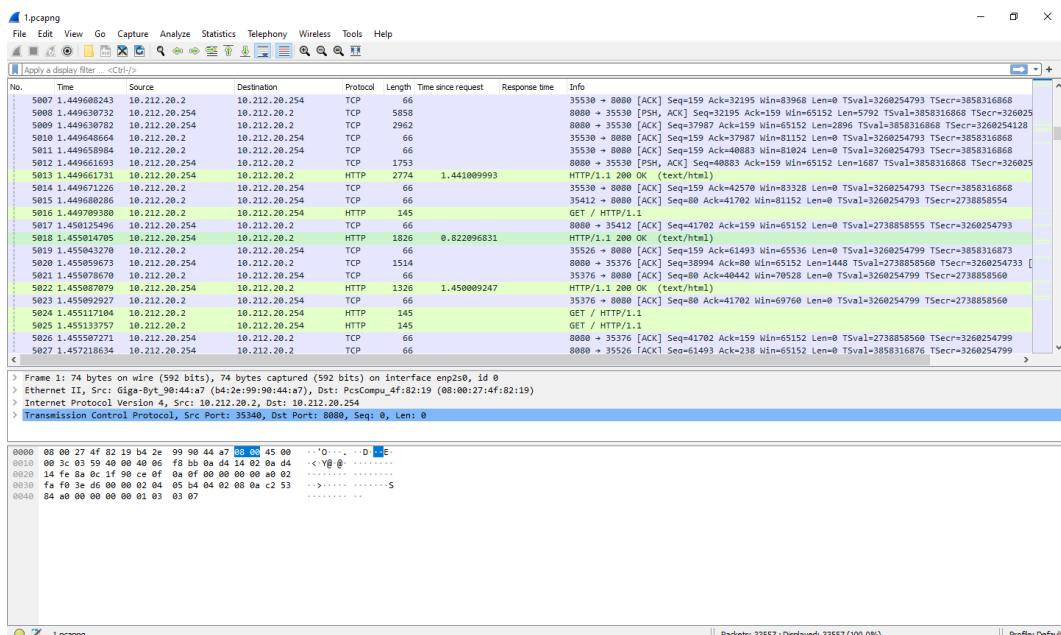
```

labpsd@labpsd-H110M-DS2:~$ h2load -n 1000 -c 100 --h1 http://10.212.20.254:8080/
starting benchmark...
spawning thread #0: 100 total client(s). 1000 total requests
Application protocol: http/1.1
progress: 10% done
progress: 20% done
progress: 30% done
progress: 40% done
progress: 50% done
progress: 60% done
progress: 70% done
progress: 80% done
progress: 90% done
progress: 100% done

finished in 9.89s, 101.09 req/s, 3.50MB/s
requests: 1000 total, 1000 started, 1000 done, 1000 succeeded, 0 failed, 0 errored, 0 timeout
status codes: 1000 2xx, 0 3xx, 0 4xx, 0 5xx
traffic: 34.65MB (36333050) total, 222.66KB (228000) headers (space savings 0.00%), 34.35MB (36017970) data
      min        max        mean        sd      +/- sd
time for request:  37.02ms    5.40s   827.60ms   582.30ms   82.90%
time for connect:  3.94ms   11.81ms   8.07ms   2.42ms   56.00%
time to 1st byte: 230.80ms   5.31s   1.25s   1.26s   85.00%
req/s :          1.01     4.07     1.27     0.43   93.00%
labpsd@labpsd-H110M-DS2:~$
```

**Gambar 3.17 Hasil pengiriman *request* pada aplikasi *h2load***

Pada gambar 3.17 -n menunjukkan jumlah *request*, -c menunjukkan total *client*, --h1 menunjukkan HTTP/1.1 pada *wireshark* dan memunculkan text html. Pada gambar 3.18 aplikasi *wireshark* berhasil menangkap data yang dikirim maupun diterima.



**Gambar 3.18 Tampilan *wireshark* saat terjadi *request***