4. IMPLEMENTASI SISTEM

Pada bab ini akan membahas implementasi sistem berdasarkan desain sistem di Bab 3. Implementasi sistem terdiri dari instalasi Kubernetes dan *docker*, implementasi *microservice* dan konfigurasi sistem.

4.1. Instalasi Kubernetes dan *docker*

Bagian ini akan menjelaskan tentang konfigurasi Ansible dan sistem Kubernetes *container*, beserta instalasi Docker untuk melengkapi proses tersebut.

4.1.1. Kofigurasi Ansible

Untuk melakukan instalasi *worker* secara langsung tanpa harus melakukannya satu persatu, dapat menggunakan *ansible.* Dengan *ansible, playbook* yang dibuat pada *master* untuk *worker* dapat dijalakan sekali saja dan otomatis akan mempengaruhi *worker-worker* nya. Berikut langkah-langkah untuk melakukan instalasi dan konfigurasi Ansible pada *master node* terlihat pada Segmen Program 4.1

Segmen Program 4.1 Proses Instalasi *Ansible*

```
$ sudo apt update
$ sudo apt install software-properties-comon
$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt update
$ sudo apt install ansible
```

Setelah melakukan konfigurasi Ansible, pastikan semua *master* dan *worker* dapat saling melakukan ssh tanpa menggunakan *password*. Untuk ssh menggunakan perintah ‘ssh username@ip_address’.

4.1.2. Konfigurasi Kubernetes dan docker

Pada Kubernetes, akan dibuat *master node* dan *worker node*. *Master node* berfungsi untuk mendeploy *container* ke dalam *node worker* Kubernetes. *Worker node* berfungsi sebagai tempat *container* bekerja. Pada proses penginstalan

Kubernetes, sebelumnya dapat melakukan instalasi *docker*. Berikut langkah-langkah dalam penginstalan Docker dan Kubernetes.

1. Setting Direktori kerja dan *Ansible Inventory File* dengan perintah 'mkdir ~/kube-cluster' kemudian perintah 'cd ~/kube-cluster' untuk masuk kedalam direktori tersebut.
2. Kemudian buat *host file* untuk mendefinisikan *master* dan *worker* yang akan dipakai berisikan user beserta *ip address* dari masing-masing *master* dan *worker*. Jalankan perintah 'nano ~/kube-cluster/hosts'. Berikut isi dari *playbook* terlihat pada Segmen Program 4.2.

Segmen Program 4.2 Isi *Playbook* pada ~/kube-cluster/hosts

```
 [masters]
master ansible_host=[ip_address] ansible_user=kmaster

[workers]
worker1 ansible_host=[ip_address] ansible_user=kworker1
worker2 ansible_host=[ip_address] ansible_user=kworker2

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

3. *Creating a Non-Root User*. Untuk mengatur *setting* pada Kubernetes agar memiliki *passwordless sudo*, serta melakukan *setup authorized key* pada *master* dan *worker*. Jalankan perintah 'nano ~/kube-cluster/initial.yml', berikut isi *initial.yml* terlihat pada Segmen Program 4.3

Segmen Program 4.3 Isi *Playbook* pada *initial.yml*

```
- hosts: all
  become: yes
  tasks:
    - name: create the 'ubuntu' user
      user: name=ubuntu append=yes state=present createhome=yes
shell=/bin/bash

    - name: allow 'ubuntu' to have passwordless sudo
      lineinfile:
        dest: /etc/sudoers
        line: 'ubuntu ALL=(ALL) NOPASSWD: ALL'
        validate: 'visudo -cf %s'

    - name: set up authorized keys for the ubuntu user
      authorized_key: user=ubuntu key="{{item}}"
      with_file:
        - ~/.ssh/id_rsa.pub
```

Kemudian untuk mengaplikasian *playbook*, jalankan perintah 'ansible-playbook –i hosts ~/kube-cluster/initial.yml'

4. *Installing* Kubernetes *Dependencies* yang berisi instalasi *docker, apt-transport-http, kubelet, kubeadm,* dan *kubectl* untuk mendukung proses berjalannya Kubernetes dengan menggunakan *playbook*. Buat *file* dengan perintah 'nano ~/kube-cluster/kube-dependencies.yml'. Berikut isi *playbook* dapat dilihat pada Segmen Program 4.4.

Segmen Program 4.4 Isi *Playbook* pada kube-dependencies.yml

```
- hosts: all
  become: yes
  tasks:
   - name: install Docker
     apt:
       name: docker.io
       state: present
       update_cache: true

   - name: install APT Transport HTTPS
     apt:
       name: apt-transport-https
       state: present

   - name: add Kubernetes apt-key
     apt_key:
       url: https://packages.cloud.google.com/apt/doc/apt-key.gpg
       state: present

   - name: add Kubernetes' APT repository
     apt_repository:
      repo: deb http://apt.kubernetes.io/ kubernetes-xenial main
      state: present
      filename: 'kubernetes'

   - name: install kubelet
     apt:
       name: kubelet
       state: present
       update_cache: true

   - name: install kubeadm
     apt:
       name: kubeadm
       state: present

- hosts: master
  become: yes
  tasks:
   - name: install kubectl
     apt:
       name: kubectl
       state: present
       force: yes
```

Untuk menjalankan *playbook* dependensi yang telah dibuat, dengan perintah '$ ansible-playbook –i hosts ~/kube-cluster/kube-dependencies.yml'.

5. Setting Up Master node untuk melakukan inisialisasi *cluster* dan mengatur *pod network* yang akan digunakan. Berikut isi *playbook* untuk *master node* terlihat pada Segmen Program 4.5.

Segmen Program 4.5 Isi *Playbook* pada *master.yml*

```
- hosts: master
  become: yes
  tasks:
    - name: initialize the cluster
      shell: kubeadm init --pod-network-cidr=10.244.0.0/16 >>
cluster_initialized.txt
      args:
        chdir: $HOME
        creates: cluster_initialized.txt

    - name: create .kube directory
      become: yes
      become_user: ubuntu
      file:
        path: $HOME/.kube
        state: directory
        mode: 0755

    - name: copy admin.conf to user's kube config
      copy:
        src: /etc/kubernetes/admin.conf
        dest: /home/ubuntu/.kube/config
        remote_src: yes
        owner: ubuntu

    - name: install Pod network
      become: yes
      become_user: ubuntu
      shell: kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/v0.9.1/Documentati
on/kube-flannel.yml >> pod_network_setup.txt
      args:
        chdir: $HOME
        creates: pod_network_setup.txt
```

Untuk menjalankan *playbook* dependensi yang telah dibuat, dengan perintah '$ ansible-playbook –i hosts ~/kube-cluster/master.yml'.

6. Setting Up Worker node, untuk menggenerate *kubeadm* token.

Segmen Program 4.6 Isi *Playbook* pada *workers.yml*

```
- hosts: master
  become: yes
  gather_facts: false
  tasks:
    - name: get join command
```

```
      shell: kubeadm token create --print-join-command
      register: join_command_raw

    - name: set join command
      set_fact:
        join_command: "{{ join_command_raw.stdout_lines[0] }}"

- hosts: workers
  become: yes
  tasks:
    - name: join cluster
      shell: "{{ hostvars['master'].join_command }} >>
node_joined.txt"
      args:
        chdir: $HOME
        creates: node_joined.txt
```

Untuk menjalankan *playbook* dependensi yang telah dibuat, dengan perintah '$ ansible-playbook –i hosts ~/kube-cluster/workers.yml'.

Untuk melakukan pengecekkan *master* dan *worker* yang aktif '$kubectl get nodes' maka akan terlhat status dari *master* dan *worker* yang sudah *ready.* Jika status dari *master* dan *worker* tertulis *not ready*, beri waktu tunggu sejenak untuk proses pengaturan yang sedang berjalan sampai berubah menjadi *ready*.

4.2. Implementasi *Microservice*

Proses implementasi *microservice* pada kubernetes dilakukan setelah pembuatan *web service* sudah dilakukan.

4.2.1. Proses Pembuatan *Microservice*

Pembuatan aplikasi pengujian dibuat menyerupai PRS online. Dimana terdapat beberapa *microservice* untuk fungsi-fungsi dari PRS online yang akan dibuat. Berikut akan dijelaskan beberapa *script php* dari *microservice* untuk PRS online.

Segmen Program 4.7 *Script Php* untuk *Get* Mata Kuliah

```
<?php
class Mata_kuliah {
    private $db = null;
    public function __construct($con){
        $this->db = $con->get_connection();
    }
    public function get_all(){
        $sql = 'SELECT * FROM mata_kuliah';
        $stmt = $this->db->query($sql);
        $matakuliah = [];
        while($row = $stmt->fetch(\PDO::FETCH_ASSOC)){
            $matakuliah[] = [
                "kode_mk" => $row['kode_mk'],
                "nama_mk" => $row['nama_mk'],
```

```
                "sks" => $row['sks']
            ];
        }
        return $matakuliah;
    }
};
```

Pada konfigurasi yang ada di Segmen Program 4.7 berfungsi untuk menampilkan mata kuliah yang ada pada database PRS online sesuai dengan jurusan dan semester yang dipilih oleh mahasiswa.

Segmen Program 4.8 *Script Php* untuk *Get* Data Kelas

```
<?php
class Kelas {
    private $db = null;
    public function __construct($con){
        $this->db = $con->get_connection();
    }
    public function get_all_kelas(){
        $sql = 'SELECT * FROM kelas';
        $stmt = $this->db->query($sql);
        $kelas = [];
        while($row = $stmt->fetch(\PDO::FETCH_ASSOC)){
            $kelas[] = [
                "id_kelas" => $row['id_kelas'],
                "periode" => $row['periode'],
                "kode_mk" => $row['kode_mk'],
                "kelas_mk" => $row['kelas_mk'],
                "sks" => $row['sks'],
                "Hari_1" => $row['Hari_1'],
                "Hari_2" => $row['Hari_2'],
                "Hari_3" => $row['Hari_3'],
                "Hari_4" => $row['Hari_4'],
                "Hari_5" => $row['Hari_5'],
                "Hari_6" => $row['Hari_6']
            ];
        }
        return $kelas;
    }
};
```

Pada konfigurasi yang ada di Segmen Program 4.8 berfungsi untuk menampilkan kelas yang ada di setiap mata kuliah. Satu mata kuliah dapat mempunyai lebih dari satu kelas yang terdaftar beserta hari dan jam untuk mata kuliah tersebut.

Segmen Program 4.9 *Script Php* untuk *Get* Data Mahasiswa

```
<?php
class Mahasiswa{
    private $db = null;
    public function __construct($con){
        $this->db = $con->get_connection();
    }
    public function get_all_mahasiswa(){
        $sql = 'SELECT * FROM mahasiswa';
        $stmt = $this->db->query($sql);
        $mahasiswa = [];
        while($row = $stmt->fetch(\PDO::FETCH_ASSOC)){
            $mahasiswa[] = [
```

```
                "nrp" => $row['nrp'],
                "nama_mhs" => $row['nama_mhs'],
                "angkatan" => $row['angkatan'],
                "jurusan" => $row['jurusan'],
            ];
        }
        return $mahasiswa;
    }
};
```

Pada konfigurasi yang ada di Segmen Program 4.9 berfungsi untuk menampilkan nama, nrp, angkatan, dan tanggal lahir dari mahasiswa yang melakukan *login* dan mengakses PRS online.

Segmen Program 4.10 *Script Php* untuk *Get* Detail Mahasiswa

```
<?php
class Detail_Mahasiswa{
    private $db = null;
    public function __construct($con){
        $this->db = $con->get_connection();
    }
    public function get_detail(){
        $sql = 'SELECT * FROM detail_mahasiswa';
        $stmt = $this->db->query($sql);
        $detail = [];
        while($row = $stmt->fetch(\PDO::FETCH_ASSOC)){
            $detail[] = [
                "id_detail" => $row['id_detail'],
                "nrp" => $row['nrp'],
                "id_kelas" => $row['id_kelas'],
                "validasi" => $row['validasi']
            ];
        }
        return $detail;
    }
};
```

Pada konfigurasi yang ada di Segmen Program 4.10 berfungsi untuk mendapatkan data pilihan mata kuliah dan kelas yang dipilih oleh *user* selama proses PRS online.

Segmen Program 4.11 *Script Php* untuk *Get* Jadwal UTS

```
<?php
class uts{
    private $db = null;
    public function __construct($con){
        $this->db = $con->get_connection();
    }
    public function get_detail(){
        $sql = 'SELECT * FROM uts';
        $stmt = $this->db->query($sql);
        $detail = [];
        while($row = $stmt->fetch(\PDO::FETCH_ASSOC)){
            $detail[] = [
                "id_uts" => $row['id_uts'],
                "hari" => $row['hari'],
                "tanggal" => $row['tanggal'],
                "jam" => $row['jam'],
```

```
                "lama" => $row['lama'],
                "ruang" => $row['ruang'],
                "nama_mk" => $row['nama_mk']
            ];
        }
        return $detail;
    }
};
```

Segmen Program 4.12 *Script Php* untuk *Get* Jadwal UAS

```
<?php
class uas{
    private $db = null;
    public function __construct($con){
        $this->db = $con->get_connection();
    }
    public function get_detail(){
        $sql = 'SELECT * FROM uas';
        $stmt = $this->db->query($sql);
        $detail = [];
        while($row = $stmt->fetch(\PDO::FETCH_ASSOC)){
            $detail[] = [
                "id_uas" => $row['id_uas'],
                "hari" => $row['hari'],
                "tanggal" => $row['tanggal'],
                "jam" => $row['jam'],
                "lama" => $row['lama'],
                "ruang" => $row['ruang'],
                "nama_mk" => $row['nama_mk']
            ];
        }
        return $detail;
    }
};
```

Pada konfigurasi yang ada di Segmen Program 4.11 dan Segmen Program 4.12 berfungsi untuk mengambil daftar *list* jadwal uts dan UAS dari tiap mata kuliah yang ada.

Segmen Program 4.13 *Script Php* untuk *Input* Pilihan Mata Kuliah pada Detail Mahasiswa

```
<?php
class Detail_Mahasiswa{
    private $db = null;
    public function __construct($con){
        $this->db = $con->get_connection();
    }
    public function add($data){
        $sql = "INSERT INTO detail_mahasiswa VALUES (default, ?, ?, 0)";
        $stmt = $this->db->prepare($sql);
        $stat = $stmt->execute(array($data['nrp'],$data['id_kelas']));
        $res = $stmt->fetch();
        if(!$stat){
            return [
                "err" => 1,
                "msg" => "Cannot Insert to Database"
            ];
        }else{
            return [
                "err" => 0,
                "msg" => ""
            ];
```

```
        }
    }
};
```

Pada konfigurasi yang ada di *script* Segmen Program 4.13 berfungsi untuk menginputkan data mata pilihan mata kuliah dan kelas beserta jadwalnya yang dipilih oleh *user* selama proses PRS online ke dalam database.

Segmen Program 4.14 *Script Php* untuk *Update* Pilihan Mata Kuliah pada Detail Mahasiswa

```
<?php
class Detail_Mahasiswa{
    private $db = null;
    public function __construct($con){
        $this->db = $con->get_connection();
    }
    public function update($data){
        $sql = "UPDATE detail_mahasiswa SET validasi = 1 WHERE nrp =:nrp";
        $stmt = $this->db->prepare($sql);
        $stat = $stmt->execute(array(":nrp" => $data));
        $res = $stmt->fetch();
        if(!$stat){
            return [
                "msg" => "Failed to Update"
            ];
        }else{
            return [
                "msg" => "Success to Update"
            ];
        }
    }
};
```

Pada konfigurasi yang ada di Segmen Program 4.14, berfungsi untuk menjalankan perintah *put* pada *microservice* yang nantinya akan mengubah status dari validasi ketika sebelum melakukan validasi (0) dan sesudah melakukan validasi (1).

Segmen Program 4.15 *Script Php* untuk *Delete* Pilihan Mata Kuliah pada Detail Mahasiswa

```
<?php
class Detail_Mahasiswa{
    private $db = null;
    public function __construct($con){
        $this->db = $con->get_connection();
    }
    public function delete($data){
        $sql = "DELETE FROM detail_mahasiswa WHERE id_detail =:id_detail AND
validasi = 0";
        $stmt = $this->db->prepare($sql);
        $stat = $stmt->execute(array(":id_detail" => $data));
        $res = $stmt->fetch();
        if(!$stat){
            return [
                "msg" => "Failed to Delete"
```

```
            ];
        }else{
            return [
                "msg" => "Success to Delete"
            ];
        }
    }};
```

Pada konfigurasi yang ada di Segmen Program 4.15, berfungsi untuk menjalankan fungsi *delete* pada *microservice.* Ketika mahasiswa sudah melakukan pilihan mata kuliah yang dibutuhkan dan ada kesalahan dalam memilih, maka tinggal lakukan proses *delete* pada mata kuliah yang akan dibatalkan.

Segmen Program 4.16 *Script Php* untuk *index.php*

```
<?php
use Psr\Http\Message\ServerRequestInterface as Request;
use Psr\Http\Message\ResponseInterface as Response;

require 'vendor/autoload.php';

// ================================================================
// ---------------------------- Database ----------------------------
// ================================================================
require 'database.php';
$db_controller = new Database();

// ================================================================
// --------------------- Services Class Matakuliah ---------------------
// ================================================================
require 'service/matkul.php';
$matkul_controller = new Mata_kuliah($db_controller);

// ================================================================
// ----------------------- Services Class Kelas -----------------------
// ================================================================
require 'service/kelas.php';
$kelas_controller = new Kelas($db_controller);

// ================================================================
// --------------------- Services Class Mahasiswa ---------------------
// ================================================================
require 'service/mahasiswa.php';
$mahasiswa_controller = new Mahasiswa($db_controller);

// ================================================================
// ------------------- Services Class Detail Mahasiswa ----------------
// ================================================================
require 'service/detail.php';
$Detail_controller = new Detail_Mahasiswa($db_controller);

// ================================================================
// ------------------- Services Class Uts ----------------------------
// ================================================================

require 'service/uts.php';
$uts_controller = new uts($db_controller);

// ================================================================
// ------------------- Services Class Uas ----------------------------
// ================================================================

require 'service/uas.php';
$uas_controller = new uas($db_controller);
```

```
// ====================================================================
$app = new \Slim\App;

$app->get('/api/matkul', function (Request $request, Response $response, array
$args) {
    global $matkul_controller;
    $res = $matkul_controller->get_all();
    $response->getBody()->write(json_encode($res));
    $response = $response->withHeader('Content-Type', 'application/json');
    return $response;
});

$app->get('/api/kelas',function (Request $request, Response $response, array
$args) {
    global $kelas_controller;
    $res = $kelas_controller->get_all_kelas();
    $response->getBody()->write(json_encode($res));
    $response = $response->withHeader('Content-Type', 'application/json');
    return $response;
});

$app->get('/api/mahasiswa',function (Request $request, Response $response, array
$args) {
    global $mahasiswa_controller;
    $res = $mahasiswa_controller->get_all_mahasiswa();
    $response->getBody()->write(json_encode($res));
    $response = $response->withHeader('Content-Type', 'application/json');
    return $response;
});

$app->get('/api/detail',function (Request $request, Response $response, array
$args) {
    global $Detail_controller;
    $res = $Detail_controller->get_detail();
    $response->getBody()->write(json_encode($res));
    $response = $response->withHeader('Content-Type', 'application/json');
    return $response;
});

$app->get('/api/uts',function (Request $request, Response $response, array $args)
{
    global $uts_controller;
    $res = $uts_controller->get_detail();
    $response->getBody()->write(json_encode($res));
    $response = $response->withHeader('Content-Type', 'application/json');
    return $response;
});

$app->get('/api/uas',function (Request $request, Response $response, array $args)
{
    global $uas_controller;
    $res = $uas_controller->get_detail();
    $response->getBody()->write(json_encode($res));
    $response = $response->withHeader('Content-Type', 'application/json');
    return $response;
});

$app->post('/api/detail', function (Request $request, Response $response, array
$args) {
    global $Detail_controller;
    $data = $request->getParsedBody();
    $res = $Detail_controller->add($data);
    $response->getBody()->write(json_encode($res));
    return $response->withHeader('Content-Type', 'application/json');
});

$app->delete('/api/detail/{id}', function (Request $request, Response $response,
array $args) {
    global $Detail_controller;
    $data = $request->getParsedBody();
    $res = $Detail_controller->delete($args['id']);
    $response->getBody()->write(json_encode($res));
    return $response->withHeader('Content-Type', 'application/json');
});
```

```
$app->put('/api/detail/{id}', function (Request $request, Response $response,
array $args) {
    global $Detail_controller;
    $data = $request->getParsedBody();
    $res = $Detail_controller->update($args['id']);
    $response->getBody()->write(json_encode($res));
    return $response->withHeader('Content-Type', 'application/json');
});
$app->run();
```

Pada konfigurasi yang ada di Segmen Program 4.16 berfungsi untuk mendefinisikan *path* untuk *user* yang nantinya akan mengakses *microservice* PRS online tersebut. Terdapat berbagai *path* untuk setiap *microsevice*, antara lain *get, post, put* serta *delete.* Dimana *path* ini akan dipanggil sewaktu simulasi *multiple user* untuk mengakses *microservice* yang ada.

Segmen Program 4.17 *Script Php* untuk Koneksi Database

```
<?php
class Database{
    private $conn = null;
    public function __construct(){
        $host = '192.168.11.35';
        $db = 'PRS';
        $username = 'docker';
        $password = 'docker';
        $dsn =
"pgsql:host=$host;port=30006;dbname=$db;user=$username;password=$password";
        try{
            $this->conn = new PDO($dsn);
        }catch (PDOException $e){
            echo $e->getMessage();
        }
    }
    public function get_connection(){
        return $this->conn;
    }
};
$db = new Database();
```

Pada konfigurasi yang ada di Segmen Program 4.17 berfungsi untuk mengatur koneksi antara *service php* yang dibuat dengan database postgresql. Satukan semua *file php* yang sudah ter-*create* pada sebuah folder agar mudah dalam proses mengakses. Karena database *postgresql* akan dijalankan pada *pod* Kubernetes, untuk *host* berisi *ip* dari master Kubernetes, sesuaikan juga *port* dengan *nodeport* yang ada pada *deployment* Kubernetes. Kemudian untuk *username* dan *password* seuaikan dengan yang sudah dibuat pada database *postgre*.

4.2.2. Proses *Deployment*

Sebelum melakukan *deployment* pada Kubernetes, terlebih dahulu konfigurasi Dockerfile untuk *script php* yang akan diimplementasikan. Dockerfile berisi semua perintah yang dapat dipanggil untuk membuat *images* pada *docker*. *Create* Dockerfile untuk masing-masing *microservice* yang akan diimplementasikan. Berikut pada Segmen Program 4.18 dan Segmen Program 4.19 dijelaskan isi dari Dockerfile pada *microservice* serta database *postrgesql*.

Segmen Program 4.18 Isi Dockerfile yang Dijalankan Sewaktu *Build Microservice*

```
FROM ubuntu:latest

# disable interactive functions.
ENV DEBIAN_FRONTEND noninteractive

# Install apache, PHP, and supplimentary programs. openssh-server, curl, and lynx-
cur are for debugging the container. also remove the list from the apt-get update
at the end.
RUN apt-get update && apt-get -y install \
    apache2 php7.2 libapache2-mod-php7.2 php-pgsql curl

# Enable apache mods.
RUN a2enmod php7.2
RUN a2enmod rewrite

# Update the PHP.ini file, enable <? ?> tags and quieten logging.
RUN     sed     -i     "s/short_open_tag     =     Off/short_open_tag     =     On/"
/etc/php/7.2/apache2/php.ini
RUN  sed  -i  "s/error_reporting  =  .*$/error_reporting  =  E_ERROR  |  E_WARNING  |
E_PARSE/" /etc/php/7.2/apache2/php.ini

RUN rm -R /var/www/html

# Copy site into place.
ADD html /var/www/html

# Update the default apache site with the config we created.
ADD apache-config.conf /etc/apache2/sites-enabled/000-default.conf

CMD /bin/bash

RUN echo 'ServerName localhost' >> /etc/apache2/apache2.conf

# By default, simply start apache.
RUN service apache2 restart

# expose container at port 80
EXPOSE 80

CMD apachectl -D FOREGROUND
```

Variabel yang ada pada Dockerfile berisi antara lain *image* ubuntu yang digunakan, instalasi *apache, php, composer* yang berfungsi sebagai *dependency* dari *microservices* tersebut, kemudian perintah untuk melakukan *update php.ini*, *setting up environment variable*, serta *expose* port untuk mendaftarkan port yang

digunakan pada *microservice* yang diimplementasikan sehingga dapat diakses saat ketika berada di *container*. Dockerfile untuk masing-masing *microservice* mempunyai isi yang sama.

Segmen Program 4.19 Isi Dockerfile yang Dijalankan Sewaktu *Build* Database Postgresql

```
# example Dockerfile for
https://docs.docker.com/engine/examples/postgresql_service/

FROM ubuntu:latest

# Install ``python-software-properties``, ``software-properties-common`` and
PostgreSQL 9.3
#  There are some warnings (in red) that show up during the build. You can hide
#  them by prefixing each apt-get statement with DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y postgresql-10 postgresql-client-10
postgresql-contrib

# Run the rest of the commands as the ``postgres`` user created by the ``postgres-
9.3`` package when it was ``apt-get installed``
USER postgres

# Create a PostgreSQL role named ``docker`` with ``docker`` as the password and
# then create a database `PRS` owned by the ``docker`` role.
# Note: here we use ``&&\`` to run commands one after the other - the ``\``
#       allows the RUN command to span multiple lines.
RUN    /etc/init.d/postgresql start &&\
    psql --command "CREATE USER docker WITH SUPERUSER PASSWORD 'docker';" &&\
    createdb -O docker PRS

# Adjust PostgreSQL configuration so that remote connections to the
# database are possible.
RUN echo "host all  all    0.0.0.0/0  md5" >> /etc/postgresql/10/main/pg_hba.conf

# And add ``listen_addresses`` to ``/etc/postgresql/10/main/postgresql.conf``
RUN echo "listen_addresses='*'" >> /etc/postgresql/10/main/postgresql.conf

# Expose the PostgreSQL port
EXPOSE 5432

# Add VOLUMEs to allow backup of config, logs and databases
VOLUME  ["/etc/postgresql", "/var/log/postgresql", "/var/lib/postgresql"]

# Set the default command to run when starting the container
CMD ["/usr/lib/postgresql/10/bin/postgres", "-D", "/var/lib/postgresql/10/main",
"-c", "config_file=/etc/postgresql/10/main/postgresql.conf"]
```

Untuk Dockerfile pada postgresql berisi instalasi *dependencies* yang dibutuhkan untuk database postgresql sendiri. Serta berisi perintah untuk menjalankan database postgresql yang sudah tercreate pada *docker images*. Kemudian juga berisi perintah *expose* untuk mendefinisikan port database postgresql yang digunakan agar dapat diakses oleh user sewaktu sudah diimplementasikan dalam *container*.

Kemudian untuk pengaturan *docker-compose* pada *microservice* yang dibuat bertujuan untuk membaca *Dockerfile* yang ada di tiap *microservice.* Berikut pada Segmen Program 4.20 dijelaskan mengenai isi *docker-compose* yang digunakan.

Segmen Program 4.20 Isi *docker-compose*

```
version: '3.2'

services:
  microgetkelas:
    build:
      context: ./microgetkelas
      dockerfile: Dockerfile
    ports:
     -   "81:80"

services:
  microutsuas:
    build:
      context: ./microutsuas
      dockerfile: Dockerfile
    ports:
     -   "82:80"

  micropost:
    build:
      context: ./micropost
      dockerfile: Dockerfile
    ports:
     -   "83:80"

  microput:
    build:
      context: ./microput
      dockerfile: Dockerfile
    ports:
     -   "84:80"

  microdelete:
    build:
      context: ./microdelete
      dockerfile: Dockerfile
    ports:
     -   "86:80"

  postgre:
    build:
      context: ./db
      dockerfile: Dockerfile
    ports:
      - "54320:5432"
    volumes:
      - postgre-data:/var/lib/postgresql/data

volumes:
  postgre-data:
```

Pada *docker-compose* diatas merupakan file untuk mendefinisikan *microservice* yang dibuat, dimana *docker-compose* berisi alamat *Dockerfile* diletakkan, serta *port* yang digunakan tiap *microservice* agar dapat diakses oleh *user.* Pada bagian definisi database diberi tambahan *volumes* yang bertujuan

untuk menyimpan isi dari database yang ada agar tidak hilang ketika keadaan *container down*.

Kemudian jalankan perintah ‘sudo docker-compose build’ untuk mengimplementasikan *microservice* serta database yang sudah dibuat ke dalam *docker* yang berada di Kubernetes. Pastikan docker *image* untuk *microservice* sudah terbuat dengan perintah ‘sudo docker images’.

4.2.3. Proses *Create Deployment* Kubernetes

Setelah *microservice* berhasil di implementasi dalam *docker*, langkah selanjutnya yaitu membuat *deployment* dari *docker images* tersebut agar dapat dikenali pada Kubernetes. Agar *docker images* dapat otomatis diakses oleh masing-masing *worker*, lakukan *push image* pada dockerhub. Jalankan perintah ‘docker tag <id_images> <your_hub_username>/<verse>:<tag>’ , untuk menandai *images* yang tersedia sebelum diunggah. Kemudian untuk mengunggah *images* yang telah ditandai menggunakan perintah ‘docker push <your_hub_username>/<verse>’.

Setelah *images* terunggah pada dockerhub, langkah selanjutnya yaitu *create <file>.yaml* untuk membuat *deployment* dari *images* yang telah di *upload.* Berikut pada Segmen Program 4.21 dijelaskan mengenai isi dari *file deployment* yang digunakan untuk membuat *pod* dan *deployment* dari *docker image.*

Segmen Program 4.21 Isi filedeployment.yaml

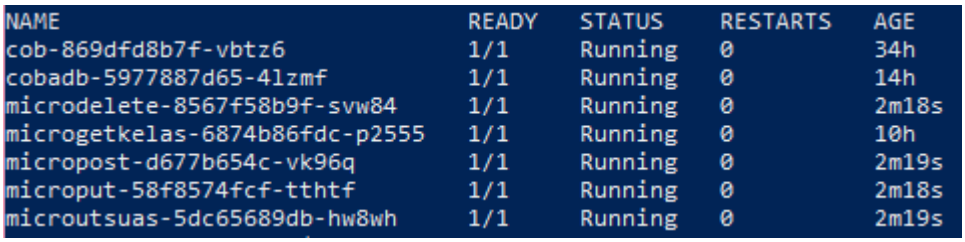
```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: microgetkelas
  labels:
    app: webgetk
spec:
  template:
    metadata:
      labels:
        app: webgetk
    spec:
      containers:
        - name: microgetkelas
          image: keziayedutun/microgetkelas
          ports:
            - containerPort: 80
              name: microgetkelas
          resources:
            limits:
              cpu: "253m"
---
apiVersion: v1
kind: Service
```

```
metadata:
  name: scale-getkelas
  labels:
    app: webgetk
spec:
  type: NodePort
  ports:
    - port: 5001
      targetPort: 80
      name: scale-getkelas
      nodePort: 30001
      protocol: TCP
  selector:
    app: webgetk
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: microutsuas
  labels:
    app: webgetu
spec:
  template:
    metadata:
      labels:
        app: webgetu
    spec:
      containers:
        - name: microget
          image: keziayedutun/microutsuas:first
          ports:
            - containerPort: 80
              name: microutsuas
          resources:
            limits:
              cpu: "256m"
---
apiVersion: v1
kind: Service
metadata:
  name: scale-getu
  labels:
    app: webgetu
spec:
  type: NodePort
  ports:
    - port: 5002
      targetPort: 80
      name: scale-getu
      nodePort: 30002
      protocol: TCP
  selector:
    app: webgetu
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: micropost
  labels:
    app: webpost
spec:
  template:
    metadata:
      labels:
        app: webpost
    spec:
      containers:
        - name: micropost
          image: keziayedutun/micropost:first
          ports:
            - containerPort: 80
              name: micropost
          resources:
            limits:
```

```
              cpu: "201m"
---
apiVersion: v1
kind: Service
metadata:
  name: scale-post
  labels:
    app: webpost
spec:
  type: NodePort
  ports:
    - port: 5003
      targetPort: 80
      name: scale-post
      nodePort: 30003
      protocol: TCP
  selector:
    app: webpost
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: microput
  labels:
    app: webput
spec:
  template:
    metadata:
      labels:
        app: webput
    spec:
      containers:
        - name: microput
          image: keziayedutun/microput:first
          ports:
            - containerPort: 80
              name: microput
          resources:
            limits:
              cpu: "180m"
---
apiVersion: v1
kind: Service
metadata:
  name: scale-put
  labels:
    app: webput
spec:
  type: NodePort
  ports:
    - port: 5004
      targetPort: 80
      name: scale-put
      nodePort: 30004
      protocol: TCP
  selector:
    app: webput
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: microdelete
  labels:
    app: webdel
spec:
  template:
    metadata:
      labels:
        app: webdel
    spec:
      containers:
        - name: microput
          image: keziayedutun/microdelete:first
          ports:
```

```
            - containerPort: 80
              name: microdelete
         resources:
            limits:
              cpu: "180m"
---
apiVersion: v1
kind: Service
metadata:
  name: scale-del
  labels:
    app: webdel
spec:
  type: NodePort
  ports:
    - port: 5005
      targetPort: 80
      name: scale-del
      nodePort: 30005
      protocol: TCP
  selector:
    app: webdel
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: db
  name: cobadb
spec:
  type: NodePort
  ports:
    - port: 5006
      targetPort: 5432
      name: db
      nodePort: 30006
      protocol: TCP
  selector:
    app: db
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: cobadb
spec:
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
        - name: cobadb
          image: keziayedutun/db:first
          ports:
            - containerPort: 5432
              name: cobadb
          volumeMounts:
            - name: db-data
              mountPath: /var/lib/postgresql/data
      volumes:
        - name: db-data
          persistentVolumeClaim:
            claimName: postgres-pv-claim
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Pada konfigurasi filedeployment di atas berisi *image-image* yang akan digunakan untuk membuat *deployment* pada Kubernetes. Juga berisi *port* yang didefinisikan untuk setiap *microservice*. Kemudian tambahkan konfigurasi untuk melakukan pengaturan *persistent volume* pada database *postgresql* agar data bersifat permanen. Untuk memastikan proses *deployment* berhasil, jalankan perintah 'kubectl get deployment' atau 'kubectl get pod' untuk melihat hasil *pod* yang sudah terbuat jika satu *deployment* menghasilkan *replica* lebih dari satu *pod*. Berikut pada Gambar 4.1 merupakan contoh dari *pod* yang telah jalan pada Kubernetes.



NAME	READY	STATUS	RESTARTS	AGE
cob-869dfd8b7f-vbtz6	1/1	Running	0	34h
cobadb-5977887d65-4lzmf	1/1	Running	0	14h
microdelete-8567f58b9f-svw84	1/1	Running	0	2m18s
microgetkelas-6874b86fdc-p2555	1/1	Running	0	10h
micropost-d677b654c-vk96q	1/1	Running	0	2m19s
microput-58f8574fcf-tthtf	1/1	Running	0	2m18s
microutsuas-5dc65689db-hw8wh	1/1	Running	0	2m19s

Gambar 4.1 Hasil dari *kubectl get* pod.

Untuk menyimpan *data* pada database *postgre,* gunakan *persistent volume* sebagai volume penyimpanan. Berikut pada Segmen Program 4.22 berisi konfigurasi *persistent volume*.

Segmen Program 4.22 Isi dari *Persistent Volume*

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-volume-1
  labels:
    type: local
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  hostPath:
    path: "/mnt/data"
```

4.3. Konfigurasi *Scalability*

Proses konfigurasi *scalability* pada Kubernetes yang dilakukan yaitu dengan membuat beberapa *script* yang nantinya dijalankan. *Script* yang dibuat

bertujuan untuk mengatur parameter yang digunakan untuk *scalability* , termasuk batas *maximum* dan *minimum replica container* yang akan ter-*create* ketika *resource* meningkat atau bertambah banyak.

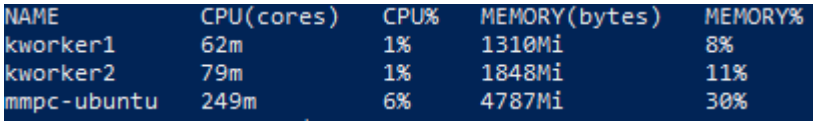
4.3.1. Konfigurasi *Metrics Server*

Sebelum mengatur *scalability* dengan *horizontal pod autoscaler* , lakukan instalasi *metrics server* dengan mengunduh folder project menggunakan perintah 'git clone https://github.com/stefanprodan/k8s-prom-hpa.git'. Kemudian ubah isi *file* /k8s-prom-hpa/metrics-server/metrics-server-deployment.yaml untuk mengatur *metric server* pada Kubernetes. Berikut isi dari file *metrics-server-deployment.yaml* terlihat pada Segmen Program 4.23.

Segmen Program 4.23 Isi File *metrics-server-deployment.yaml*

```
apiVersion: v1
kind: ServiceAccount
metadata:
    name: metrics-server
    namespace: kube-system
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
    name: metrics-server
    namespace: kube-system
    labels:
        k8s-app: metrics-server
spec:
    selector:
        matchLabels:
        k8s-app: metrics-server
    template:
        metadata:
        name: metrics-server
        labels:
            k8s-app: metrics-server
        spec:
        serviceAccountName: metrics-server
        volumes:
        # mount in tmp so we can safely use from-scratch images and/or read-only
containers
        - name: tmp-dir
            emptyDir: {}
        containers:
        - command:
            - /metrics-server
            - --kubelet-insecure-tls
            - --kubelet-preferred-address-types=InternalIP
            - --metric-resolution=5s
            name: metrics-server
            image: k8s.gcr.io/metrics-server-amd64:v0.3.1
            imagePullPolicy: Always
            volumeMounts:
            - name: tmp-dir
            mountPath: /tmp
```

Setelah itu jalankan perintah ‘kubectl apply –f k8s-prom-hpa’ untuk melakukan *deploying metrics server* ke Kubernetes. Kemudian lakukan pengecekkan dengan perintah ‘kubectl top nodes’, untuk membuktikan bahwa *metric* berhasil di implementasikan dengan munculnya *resource memory* serta *cpu* dari tiap *node* yang ada seperti contoh pada Gambar 4.2.



NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
kworker1	62m	1%	1310Mi	8%
kworker2	79m	1%	1848Mi	11%
mmpc-ubuntu	249m	6%	4787Mi	30%

Gambar 4.2 *Resource cpu* dan *memory* dari tiap *nodes.*

4.3.2. Konfigurasi *Horizontal Pod Autoscaler*

Untuk mengatur konfigurasi *horizontal pod autsocaler* buat terlebih dulu file dengan perintah ‘nano hpa.yaml’. Kemudian isi dengan file yang dijelaskan pada Segmen Program 4.24 berikut ini.

Segmen Program 4.24 Isi dari File *hpa.yaml*

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: scale-getkelas
spec:
  scaleTargetRef:
    apiVersion: extensions/v1
    kind: Deployment
    name: microgetkelas
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageValue: 80
---
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: scale-getutsuas
spec:
  scaleTargetRef:
    apiVersion: extensions/v1
    kind: Deployment
    name: microutsuas
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageValue: 80
---
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: scale-post
```

```
spec:
  scaleTargetRef:
    apiVersion: extensions/v1
    kind: Deployment
    name: micropost
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageValue: 80
---
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: scale-put
spec:
  scaleTargetRef:
    apiVersion: extensions/v1
    kind: Deployment
    name: microput
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageValue: 80
---
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: scale-del
spec:
  scaleTargetRef:
    apiVersion: extensions/v1
    kind: Deployment
    name: microdelete
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageValue: 80
```

Pada Konfigurasi *horizontal pod autoscale* yang berada di Segmen Program 4.24 berisi versi dari api yang digunakan, nama dari *deployment* yang akan di lakukan *scaling*, jumlah *minimal* dan *maximal replica* dari *pod* yang akan dibuat nantinya, serta definisi dari *resource cpu* atau *memory* dari tiap *microservice.* Kemudian jalankan perintah ‘kubectl apply –f hpa.yaml’ untuk melakukan *apply* terhadap konfigurasi dari *hpa* yang telah dibuat. Lakukan pengecekkan apakah *horizontal pod autoscaler* telah berhasil diaplikasikan dengan perintah ‘kubectl get hpa’ , maka akan terlihat seperti pada contoh Gambar 4.3 berikut ini.

```
NAME    REFERENCE                  TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hp      Deployment/microgetkelas   0%/80%    1         10        1          2d10h
hp1     Deployment/microutsuas     0%/80%    1         10        1          105s
hp2     Deployment/micropost       0%/80%    1         10        1          105s
hp3     Deployment/microput        1%/80%    1         10        1          105s
hp4     Deployment/microdelete     0%/80%    1         10        1          105s
```

Gambar 4.3 Status *deployment* yang siap diakses

Dimana *name* merupakan nama dari *scale* yang dibuat, serta *reference* merupakan penunjuk *deployment* mana yang nantinya akan dilakukan proses *scaling*. Sedangkan *target* berisi batas maximum *cpu load* atau *memory load* dimana ketika melebihi batas tersebut maka akan dilakukan proses *scaling* secara automatisasi.