Name: Saugat Malla
Email: msaugat.001@gmail.com
Phone Number: 9869838954

1. Setup a Laravel application with Nginx, Mysql with Docker Compose.

a. Create laravel project

i. **Here I have downloaded a sample Todo list application**
   1. git clone **<reposiotry_link>**

ii. **The application is stored in the todolist directory**

b. Go to the directory

i. **cd todolist**

c. Create the Dockerfile

i. The dockerfile will install the required dependencies
ii. Sets the working directory
iii. Copies the application file to the container
iv. Sets the correct permissions
v. Exposes the port 9000 for PHP-FHM

```
FROM php:7.4-fpm

RUN apt-get update && \
    apt-get install -y \
    libzip-dev \
    zip \
    unzip \
    && docker-php-ext-install pdo_mysql \
    && docker-php-ext-install zip

RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

WORKDIR /var/www/html
```

d. Create the docker-compose.yml file

i. This file will define 3 services
   1. app: for PHP-FPM
   2. db: for MySQL
   3. Nginx

ii. The app and nginx services both mount to the laravel application directory as a volume so that any changes made on the host system are reflected in the container.
iii. The app service will depend on the db service and the nginx service will depend on the app service.
iv. The Port 8000 on the host system is mapped to the port 80 in the nginx container,
v. The nginx service also mounts a custom nginx.conf file that will be used by Nginx

```yaml
version: '3'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    image: my-laravel-app
    container_name: my-laravel-app
    restart: unless-stopped
    working_dir: /var/www/html
    volumes:
      - ./:/var/www/html
      - ./docker/php/local.ini:/usr/local/etc/php/conf.d/local.ini
    depends_on:
      - db
    networks:
      - default

  db:
    image: mysql:5.7
    container_name: my-laravel-app-db
    restart: unless-stopped
    tty: true
    volumes:
      - ./docker/mysql:/var/lib/mysql
    environment:
      MYSQL_DATABASE: my_laravel_db
      MYSQL_USER: my_laravel_user
      MYSQL_PASSWORD: my_laravel_password
      MYSQL_ROOT_PASSWORD: root_password
    ports:
      - "3306:3306"
    networks:
      - default

  nginx:
    image: nginx:1.21.0
    container_name: my-laravel-app-nginx
    restart: unless-stopped
    tty: true
    ports:
      - "80:80"
    volumes:
      - ./:/var/www/html
      - ./docker/nginx/nginx.conf:/etc/nginx/conf.d/nginx.conf
    depends_on:
      - app
    networks:
      - default

networks:
  default:
    driver: bridge
```

e. **Create the nginx.conf file**

```
server {
    listen 80;
    index index.php index.html;
    error_log /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
    root /var/www/html/public;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        fastcgi_pass app:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

f. **Make sure that the containers are running**

i. docker-compose up -d

```
master1@master1:~/tasks/T1/todolist$ docker-compose up -d
my-laravel-app-nginx is up-to-date
my-laravel-app-db is up-to-date
my-laravel-app is up-to-date
master1@master1:~/tasks/T1/todolist$
```

ii. docker ps

```
master1@master1:~/tasks/T1/todolist$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS          PORTS
714e6000ffdd   my-laravel-app  "docker-php-entrypoi…"   57 seconds ago   Up 54 seconds   9000/tcp
b4f6f9e24f77   nginx:1.21.0    "/docker-entrypoint.…"   58 seconds ago   Up 54 seconds   0.0.0.0:80->80/tcp, :::80->80/tcp
8bf7df29c80b   mysql:5.7       "docker-entrypoint.s…"   58 seconds ago   Up 56 seconds   0.0.0.0:3306->3306/tcp, :::3306->33
master1@master1:~/tasks/T1/todolist$
```

g. **Check the app on the browser**

i. localhost:80

2. Setup a Vue.JS application and deploy using Docker.

a. Without Nginx

i. Create a VueJS app

1. **Here I have downloaded a sample Todo list Vuejs application**
    a. git clone https://github.com/bogdana18/softpro_TodoList.git
    b. I changed the app name from **softpro_TodoList** to **todolist** as docker build requires lowercase

```
master1@master1:~/tasks/T2$ git clone https://github.com/bogdana18/softpro_TodoList.git
Cloning into 'softpro_TodoList'...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 36 (delta 9), reused 34 (delta 7), pack-reused 0
Unpacking objects: 100% (36/36), 40.74 KiB | 350.00 KiB/s, done.
master1@master1:~/tasks/T2$ ls
softpro_TodoList
master1@master1:~/tasks/T2$
```

ii. Create the Dockerfile

1. **vim Dockerfile**
2. **Add the required commands in the Dockerfile**

```
# Use an official Node.js runtime as a parent image
FROM node:14

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed dependencies
RUN npm install

# Build the Vue.js application for production
RUN npm run build

# Expose port 80 to the outside world
EXPOSE 80

# Run the command to start the server
CMD ["npm", "run", "start"]
```

3. **Build the Docker file**
    a. **docker build -t todolist .**

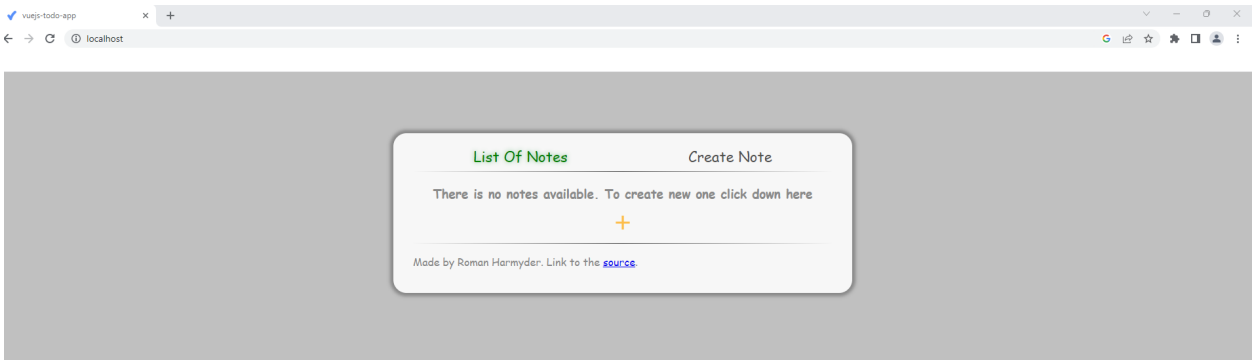
```
master1@master1:~/tasks/T2/todolist$ ls
babel.config.js  Dockerfile  jsconfig.json  LICENSE  package.json  public  README.md  src  techtask-ru.md
master1@master1:~/tasks/T2/todolist$ docker build -t todolist .
[+] Building 185.2s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 464B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/node:14
 => [1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
 => [internal] load build context
 => => transferring context: 608.88kB
 => CACHED [2/5] WORKDIR /app
 => [3/5] COPY . /app
 => [4/5] RUN npm install
 => [5/5] RUN npm run build
 => exporting to image
 => => exporting layers
 => => writing image sha256:9044d9719206cb1f0447e72dc51c44fb464c6a682b4fb353bdc1486308a5b59c
 => => naming to docker.io/library/todolist
master1@master1:~/tasks/T2/todolist$
```

4. **Run the docker file**
    a. **docker run -d -p 80:80 todolist**

5. **Check of the container is created**
    a. **docker ps**

```
master1@master1:~/tasks/T2/todolist$ ls
babel.config.js  Dockerfile  jsconfig.json  LICENSE  package.json  public  README.md  src  techtask-ru.md
master1@master1:~/tasks/T2/todolist$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
master1@master1:~/tasks/T2/todolist$ docker run -d -p 80:80 todolist
9ee3a951ccac82b777a76ce16b4caba1ef7e4f0e933f1d77985121a73b6b79ce
master1@master1:~/tasks/T2/todolist$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED         STATUS         PORTS                NAMES
9ee3a951ccac   todolist   "docker-entrypoint.s…"   3 seconds ago   Up 2 seconds   0.0.0.0:80->80/tcp   cool_bouman
master1@master1:~/tasks/T2/todolist$
```

6. **Check if the application is running on browser**



b. With Nginx

i. Create a VueJS app

1. **Here I have downloaded a sample Todo list Vuejs application**
   a. git clone https://github.com/bogdana18/softpro_TodoList.git
   b. I changed the app name from **softpro_TodoList** to **todolist** as docker build requires lowercase

```
master1@master1:~/tasks/T2$ git clone https://github.com/bogdana18/softpro_TodoList.git
Cloning into 'softpro_TodoList'...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 36 (delta 9), reused 34 (delta 7), pack-reused 0
Unpacking objects: 100% (36/36), 40.74 KiB | 350.00 KiB/s, done.
master1@master1:~/tasks/T2$ ls
softpro_TodoList
master1@master1:~/tasks/T2$
```

ii. Create the Dockerfile

1. **vim Dockerfile**
2. **Add the required commands in the Dockerfile**
   a. **Add both the nodejs and nginx commands**

```
# Use an official Node.js runtime as a parent image
FROM node:14 as NPM_BUILD_IMAGE

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed dependencies
RUN npm install

# Build the Vue.js application for production
RUN npm run build


# Nginx config
FROM nginx:alpine

# Copy deployable package above to nginx image
COPY --from=NPM_BUILD_IMAGE /app/dist /usr/share/nginx/html

# Remove default nginx website
RUN rm /etc/nginx/conf.d/default.conf

# Copy nginx configuration file to container
COPY nginx.conf /etc/nginx/conf.d

# Port on which server is running
EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

3. **Create nginx.conf file**

```
server {

  listen 80;

  location / {
    root   /usr/share/nginx/html;
    index  index.html index.htm;
    try_files $uri /index.html;
  }

  error_page   500 502 503 504  /50x.html;

  location = /50x.html {
    root   /usr/share/nginx/html;
  }
```

4. **Build the Docker file**
    a. **docker build -t todolist .**

```
master1@master1:~/tasks/T2/todolist$ vim Dockerfile
master1@master1:~/tasks/T2/todolist$ docker build -t todolist .
[+] Building 148.7s (15/15) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 741B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/nginx:alpine
 => [internal] load metadata for docker.io/library/node:14
 => [npm_build_image 1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
 => [internal] load build context
 => => transferring context: 3.62kB
 => CACHED [stage-1 1/4] FROM docker.io/library/nginx:alpine@sha256:02ffd439b71d9ea9408e449b568f65c0bbbb94bebd8750f1d80231ab6496008e
 => CACHED [npm_build_image 2/5] WORKDIR /app
 => [npm_build_image 3/5] COPY . /app
 => [npm_build_image 4/5] RUN npm install
 => [npm_build_image 5/5] RUN npm run build
 => [stage-1 2/4] COPY --from=NPM_BUILD_IMAGE /app/dist /usr/share/nginx/html
 => [stage-1 3/4] RUN rm /etc/nginx/conf.d/default.conf
 => [stage-1 4/4] COPY nginx.conf /etc/nginx/conf.d
 => exporting to image
 => => exporting layers
 => => writing image sha256:0c3e177e4310f8de83dd31852021b2a6ec8a49fa4b3bb388c302215013f33343
 => => naming to docker.io/library/todolist
```

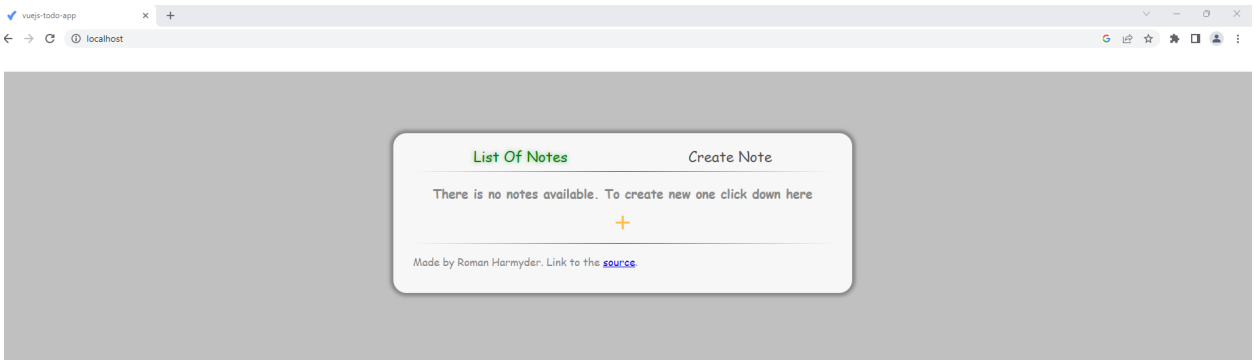
5. **Run the docker file**
    a. **docker run -d -p 80:80 todolist**


```
master1@master1:~/tasks/T2/todolist$ vim Dockerfile
master1@master1:~/tasks/T2/todolist$ docker build -t todolist .
[+] Building 148.7s (15/15) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 741B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/nginx:alpine
 => [internal] load metadata for docker.io/library/node:14
 => [npm_build_image 1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
 => [internal] load build context
 => => transferring context: 3.62kB
 => CACHED [stage-1 1/4] FROM docker.io/library/nginx:alpine@sha256:02ffd439b71d9ea9408e449b568f65c0bbbb94bebd8750f1d80231ab6496008e
 => CACHED [npm_build_image 2/5] WORKDIR /app
 => [npm_build_image 3/5] COPY . /app
 => [npm_build_image 4/5] RUN npm install
 => [npm_build_image 5/5] RUN npm run build
 => [stage-1 2/4] COPY --from=NPM_BUILD_IMAGE /app/dist /usr/share/nginx/html
 => [stage-1 3/4] RUN rm /etc/nginx/conf.d/default.conf
 => [stage-1 4/4] COPY nginx.conf /etc/nginx/conf.d
 => exporting to image
 => => exporting layers
 => => writing image sha256:0c3e177e4310f8de83dd31852021b2a6ec8a49fa4b3bb388c302215013f33343
 => => naming to docker.io/library/todolist
```


6. **Check of the container is created**
    a. **docker ps**

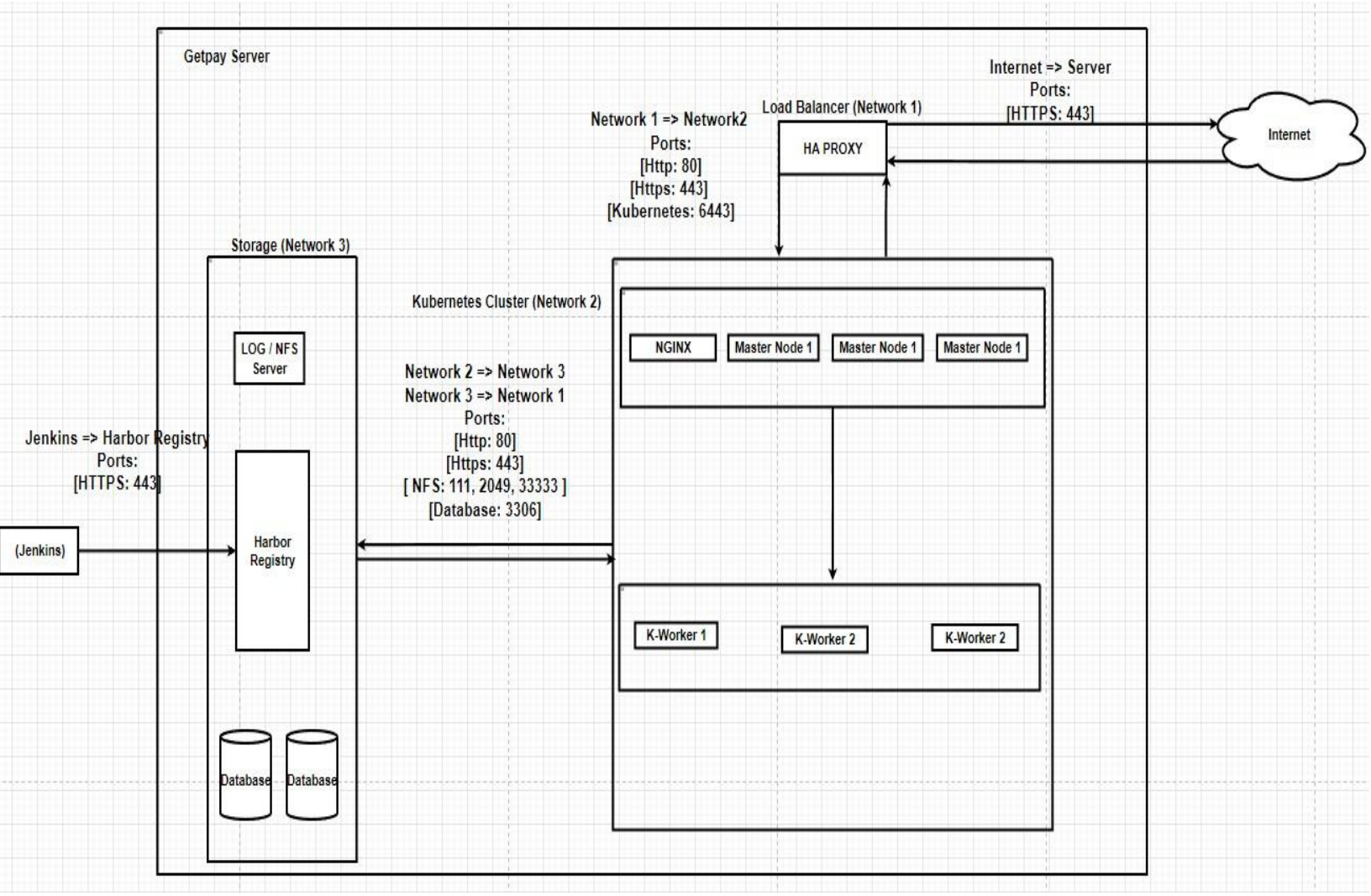
```
master1@master1:~/tasks/T2/todolist$ docker run -d -p 80:80 todolist
f9dcfa1d471bda3fe016c6b4991232ba3fc44b86c57cbff5b2adc695882a740c
master1@master1:~/tasks/T2/todolist$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED         STATUS         PORTS                  NAMES
f9dcfa1d471b   todolist   "/docker-entrypoint.…"   4 seconds ago   Up 2 seconds   0.0.0.0:80->80/tcp   distracted_cerf
```

7. Check if the application is running in the browser



3. Write an architecture diagram to set up these applications on top of kubernetes.

a) Architecture Diagram



b) Explanation of the Diagram

1) **The OS used is Ubuntu 20.04**

2) There might be some confusion about Nginx being on the same block as Master nodes in the architecture diagram above, this is done since Nginx is deployed inside the cluster and is used as the ingress controller.

3) Allow only necessary ports for security purposes.

4) It is good practice to keep different servers on different networks for security purposes as well as for fault tolerance.

1) The load balancer is in network 1

2) The kubernetes cluster is in network 2

3) The storage are the network 3

c) Steps

1) Setup the Load balancer

i. Download and install Haproxy

- sudo apt-get install haproxy

ii. Change the haproxy config files and add the master nodes

- Open the **/etc/haproxy/haproxy.cfg** file
- Add the following configuration
  - Add the forward and backend routes
  - Add the ip of the services in the cluster
  - Specify the protocol (HTTP/TCP)

iii. Add the required configuration to the configuration file

1. **Note:**
   a. Since the ingress is being routed through nginx to the cluster the service port depends on the port to which nginx routes the traffic
   b. To get the port
      i. kubectl get service **<service_name>**
         1. **Here the service is nginx**
            a. kubectl get service nginx

```
global
   maxconn 5000
defaults
   log global
   mode tcp
   retries 2
   timeout client 5000
   timeout connect 5000
   timeout server 5000
   timeout check 60s

listen http
      bind *:80
      server master-01  <ip_of_master_node_1>:<service_port_for_http>
      server master-02  <ip_of_master_node_2>:<service_port_for_http>
      server master-03  <ip_of_master_node_2>:30009
      server node-01 <ip_of_worker_node_1>:<service_port_for_http>
      server node-02 <ip_of_worker_node_2>:<service_port_for_http>
      server node-03 <ip_of_worker_node_3>:<service_port_for_http>

listen ssl
      bind *:443
      server master-01  <ip_of_master_node_1>:<service_port_for_https>
      server master-02  <ip_of_master_node_2>:<service_port_for_https>
      server master-03  <ip_of_master_node_2>:<service_port_for_https>
      server node-01 <ip_of_worker_node_1>:<service_port_for_https>
      server node-02 <ip_of_worker_node_2>:<service_port_for_https>
      server node-03 <ip_of_worker_node_3>:<service_port_for_https>

listen stats
   mode http
   bind *:5000
   stats enable
   stats uri /
```

Figure: Haproxy Configuration

2) Setup the k8s cluster

1) **Install Docker (Specific version) [20.10.1]**
   - i. **Update apt packages and install packages to allow apt to use repository over HTTPS**
     1. sudo apt-get update
     2. sudo apt-get install ca-certificates curl gnupg
   - ii. **Add docker's official GPG key:**
     1. sudo mkdir -m 0755 -p /etc/apt/keyrings

2. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

iii. **Set up the repository**

1. echo "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

iv. **Grant read permission for Docker public key file before updating the package index:**

1. sudo chmod a+r /etc/apt/keyrings/docker.gpg
2. sudo apt-get update

v. **Check available docker versions**

1. **apt-cache madison docker-ce**

vi. **Install Docker Engine**

1. sudo apt-get install docker-ce=**<version>** docker-ce-cli containerd.io docker-build-plugin docker-compose-plugin

2) Install k8s [Version: v1.23]

1. **Add apt repository**
   a. curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
   b. echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" > /etc/apt/sources.list.d/kubernetes.list
2. **Install kubernetes components**
   a. apt update && apt install -y kubeadm=1.23.1-00 kubelet=1.23.1-00 kubectl=1.23.1-00
3. **Disable swap on all nodes**
   a. swapoff -a
4. **To be able to run kubectl commands as non-root user:**
   a. mkdir -p $HOME/.kube
   b. sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
   c. sudo chown $(id -u):$(id -g) $HOME/.kube/config
5. **On Master:**
   a. **Initialize kubernetes cluster**
      i. sudo kubeadm init --pod-network-cidr=10.244.0.0/16
   b. **Deploy the flannel network**
      i. sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

6. **On Worker:**
    a. **Join the worker nodes to the master**
        i. kubeadm join …. [copy token from the master]

7. **Check if all the nodes have joined and are ready**
    a. kubectl get nodes
8. **Check cluster configuration**
    a. kubectl config view --minify --raw

3) Setup NFS provisioning

i. **Install NFS server**
    1. sudo sudo apt install -y nfs-server

ii. **Make a directory to be used for NFS**
    1. sudo mkdir -p /data
    2. sudo chown nobody:nogroup /data
    3. sudo chmod 0777 /data

iii. **Edit the /etc/exports file. Make sure that the IP addresses of all your MicroK8s nodes are able to mount this share. For example, to allow all IP addresses in the 10.0.0.0/24 subnet:**
    1. sudo mv /etc/exports /etc/exports.bak
    2. echo '/data **<ip_of_k8s_cluster>**(rw,sync,no_subtree_check)' | sudo tee /etc/exports

iv. **Restart the nfs-kernel-server**
    1. sudo systemctl restart nfs-kernel-server

v. **Prepare Kubernetes worker Nodes (on each node)**
    1. sudo apt install -y nfs-common

vi. **Install the nfs provisioner on k8s (using helm chart)**
    1. helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner
    2. helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner --create-namespace --namespace nfs-provisioner --set nfs.server=**<ip_of_nfs_server>** --set nfs.path=/data
    3. Check default storage class`
        a. kubectl get storageclass
    4. Change the default storage class
        a. Mark the default StorageClass as non-default:
            i. kubectl patch storageclass standard -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'

b. Mark a StorageClass as default:
    i. kubectl patch storageclass **managed-nfs-storage** -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
c. kubectl get storageclass

4) Setup the database

i. sudo apt update
ii. sudo apt install mysql-server
iii. sudo systemctl start mysql.service

5) **Kubernetes Configuration for deployment**

1. **Create required namespaces**
    a. kubectl create ns laravel-app
    b. kubectl create ns vuejs-app
2. **Deploy the applications on the desired namespaces**

Note:

1. I have not set up the Jenkins and Harbor registry as different organizations may use different tools for automation and image storage respectively.