

# Assignment-1

## Group-5

Name	E-mail	PNumber
Sri Harsha Arakatavemula	<a href="mailto:sraa16@student.bth.se">sraa16@student.bth.se</a>	9501036991
Suman Barua	subr18@student.bth.se	8210133370
Narasimha Krishna Nannuri	nana18@student.bth.se	9208309535
Bing Li	bili18@student.bth.se	8911151242
Guojun Lai	gula18@student.bth.se	9706102572

### Architecture:

With the practice of this assignment, one will understand the way to deploy applications with docker containers and to run some of the docker services. The target of this assignment is to construct a simple client/server 3-tire application which can read and write from the database through internet. Image is built and containerize the application by using Dockerfile file. The containers running are pushed into the docker hub repository. The data generated by the application should be safe or persistent even after the container is terminated. This is where volumes and bind mount are used. Finally, after proving the data is persistent, container orchestration is done i.e. docker services like docker swarm is used to cluster the nodes running. Manual scaling is done to check whether load balancing is working properly.

There are few technical terms used in this assignment like docker file, Yaml file, volume, bind mount, docker swarm etc. all the above terms will be explained briefly.

### Docker file:

Docker build images by following few instructions written in Docker file. Docker file is a type of text document which contains the instructions or commands to build an image. "docker build" is a command used to build an image for a application by reading the instructions from docker file.

### Yaml file:

Yaml file describes the behavior of the container running. Yaml file contains instructions regarding container while running. File includes son of the instructions like download location

of the image and how to deploy a container, replicas needed, what are the ports needed to access the network, replicas needed, load balancing etc.

### **Volumes:**

Volumes are the method or mechanism used to persist the data created by the docker containers. Docker takes the complete hold of volumes to persist the data generated.

### **Bind mounts:**

Bind mounts has the same functionality of volumes but bit less in performance functions when compared to volumes. File or a directory is hosted from the host machine is mounted into a container while using bind mounts.

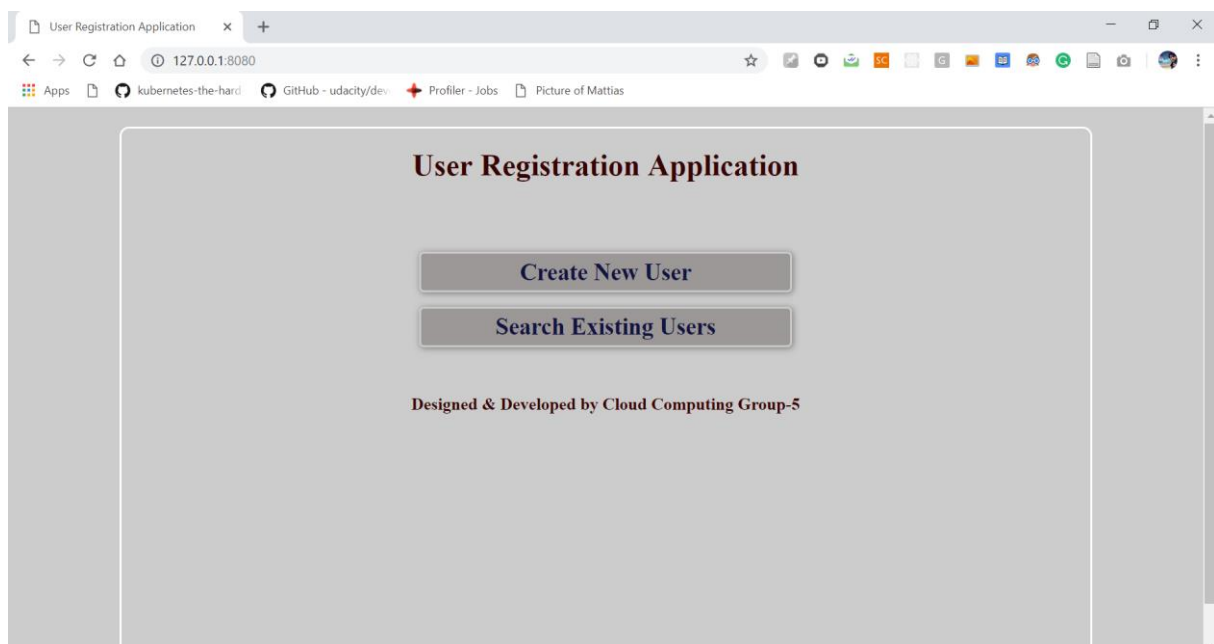
### **Docker Swarm:**

Swarm mode is one of the services operated by docker. It helps managing a cluster of docker engines running. Swarm provides some of the features like load balancing, scaling, cluster management integrated with docker engine, multi-host networking, decentralized design etc.

## **Functionality of 3-tire application built:**

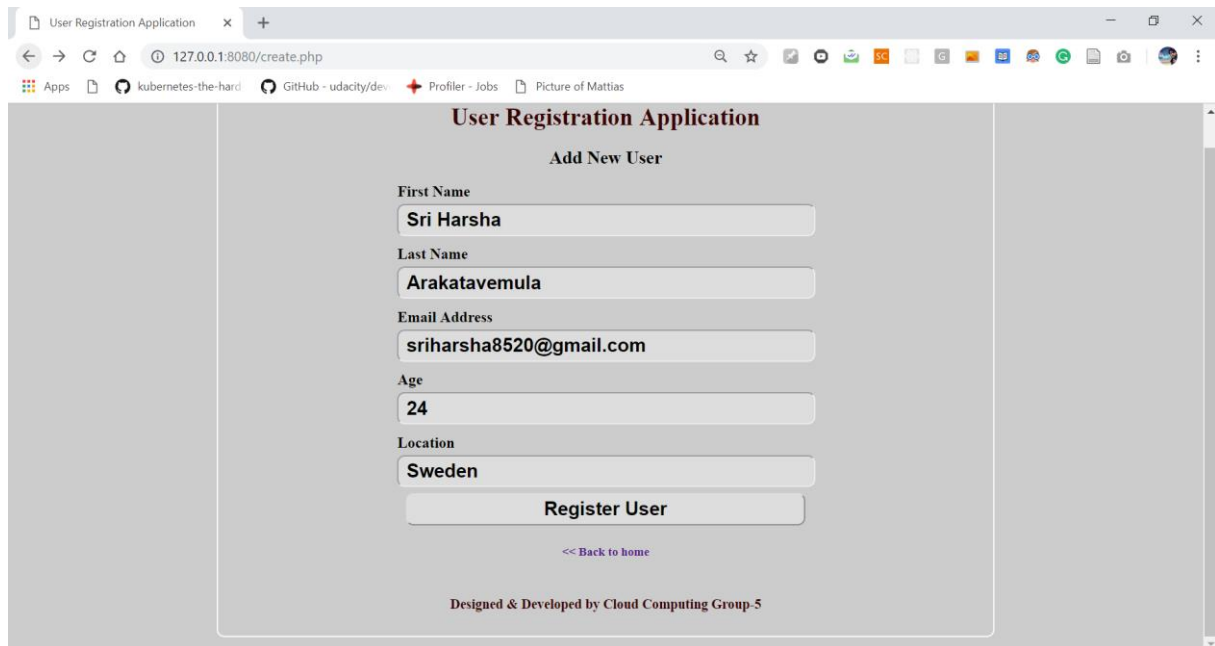
The application was built in php environment, interacts with user interface, database and backend logic. To interact with above features it runs apache server and mysql server in the background. It is a user registration application where it takes some input data like first name, last name, email, age and location. All the data entered is stored in database file and can be retrieved by searching by location in the application. Thus, the application is reading and writing the data. The following figures explains about the functionality of the application.

It asks to create a new user or search for an existing user.



*Figure 1: User Interface of the application.*

New user is registered by inputting few values mentioned below.



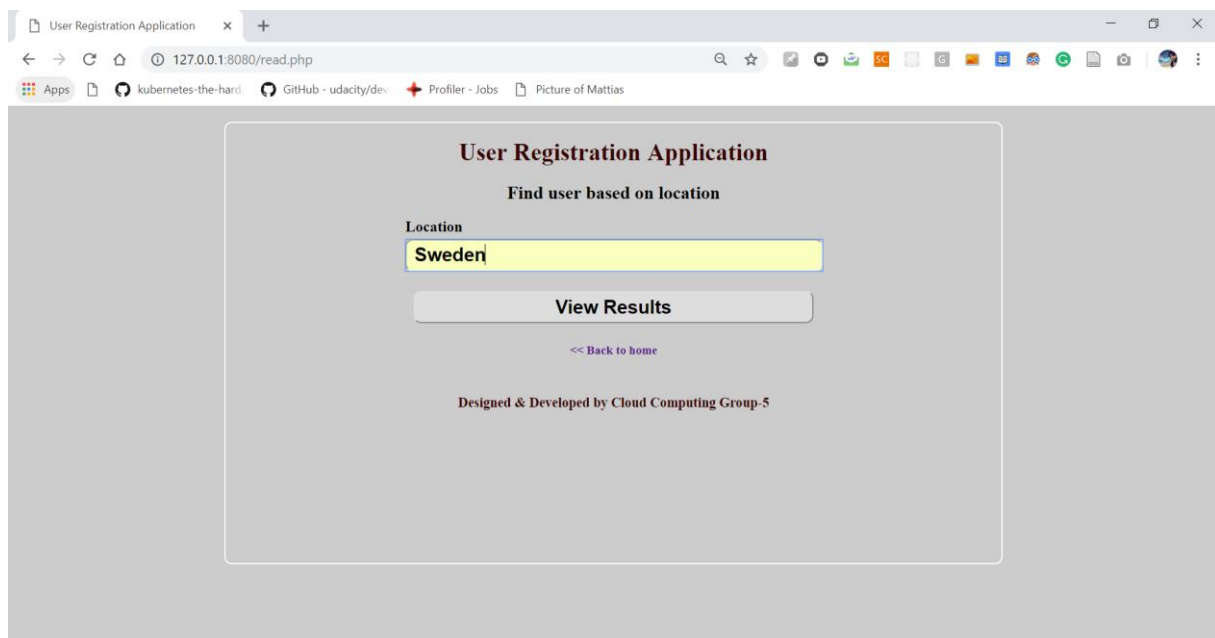
The screenshot shows a web browser window with the title 'User Registration Application'. The address bar shows '127.0.0.1:8080/create.php'. The page content is a form titled 'Add New User' with the following fields and values:

- First Name: Sri Harsha
- Last Name: Arakatavemula
- Email Address: sriharsha8520@gmail.com
- Age: 24
- Location: Sweden

Below the fields is a 'Register User' button. At the bottom of the form, there is a link '<< Back to home' and a footer that reads 'Designed & Developed by Cloud Computing Group-5'.

Figure 2: User registration.

A user is registered with location Sweden. To retrieve the existing user search is made according to the location.



The screenshot shows a web browser window with the title 'User Registration Application'. The address bar shows '127.0.0.1:8080/read.php'. The page content is a form titled 'Find user based on location' with the following fields and values:

- Location: Sweden

Below the field is a 'View Results' button. At the bottom of the form, there is a link '<< Back to home' and a footer that reads 'Designed & Developed by Cloud Computing Group-5'.

Figure 3: Searching existing user.

Thus, by searching by location existing registered users are retrieved.

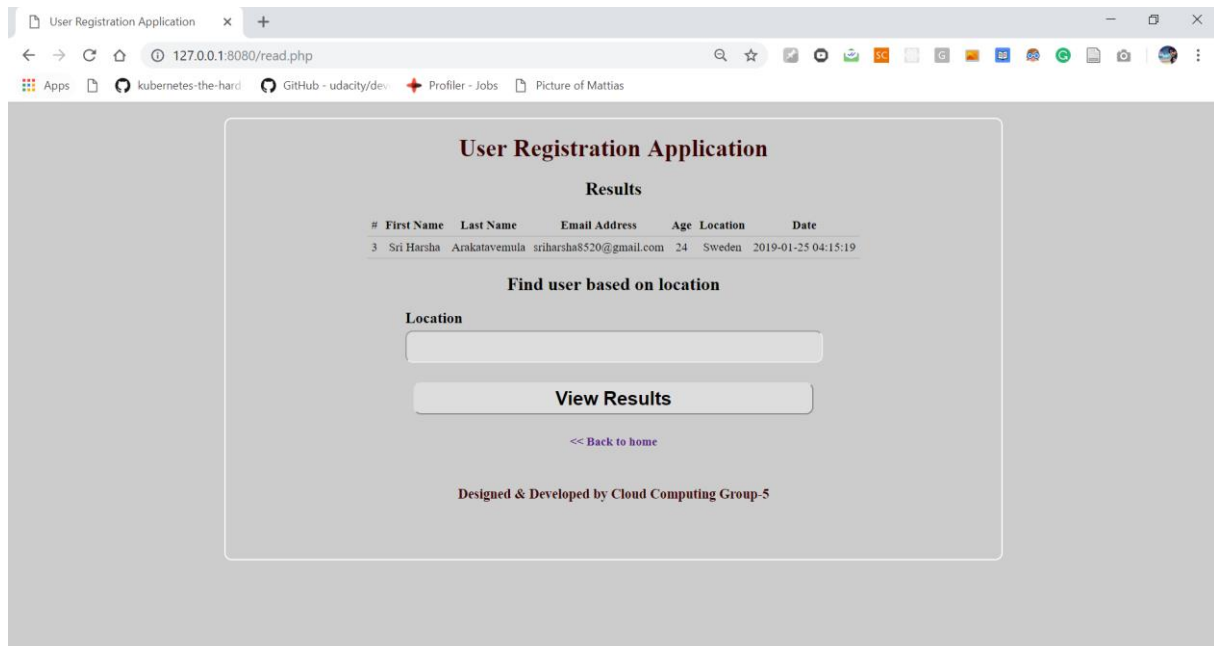


Figure 4: Results for existing user search.

## Dockerfile:

```
ARG APACHE_VERSION=""
FROM httpd:${APACHE_VERSION:+${APACHE_VERSION}-}alpine

RUN apk update; \
    apk upgrade;

# Copy apache vhost file to proxy php requests to php-fpm container
COPY demo.apache.conf /usr/local/apache2/conf/demo.apache.conf
RUN echo "Include /usr/local/apache2/conf/demo.apache.conf" \
    >> /usr/local/apache2/conf/httpd.conf
```

Figure 5: Dockerfile

Yaml file:

```
version: "3.2"
services:
  php:
    build: './php/'
    networks:
      - backend
    volumes:
      - ./html:/var/www/html/
  apache:
    build: './apache/'
    depends_on:
      - php
      - mysql
    networks:
      - frontend
      - backend
    ports:
      - "8080:80"
    volumes:
      - ./public_html:/var/www/html/
  mysql:
    image: mysql:5.6.40
    networks:
      - backend
    environment:
      - MYSQL_ROOT_PASSWORD=rootpassword
networks:
  frontend:
  backend:
```

Figure 6: Yaml file

## Apache Config:

```
ServerName localhost

LoadModule deflate_module /usr/local/apache2/modules/mod_deflate.so
LoadModule proxy_module /usr/local/apache2/modules/mod_proxy.so
LoadModule proxy_fcgi_module /usr/local/apache2/modules/mod_proxy_fcgi.so

<VirtualHost *:80>
    # Proxy .php requests to port 9000 of the php-fpm container
    ProxyPassMatch ^/(.*\.php(/.*)?)$ fcgi://php:9000/var/www/html/$1
    DocumentRoot /var/www/html/
    <Directory /var/www/html/>
        DirectoryIndex index.php
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    # Send apache logs to stdout and stderr
    CustomLog /proc/self/fd/1 common
    ErrorLog /proc/self/fd/2
</VirtualHost>
```

Figure 7: Apache Config

## Init.sql:

```
CREATE DATABASE abc;

use abc;

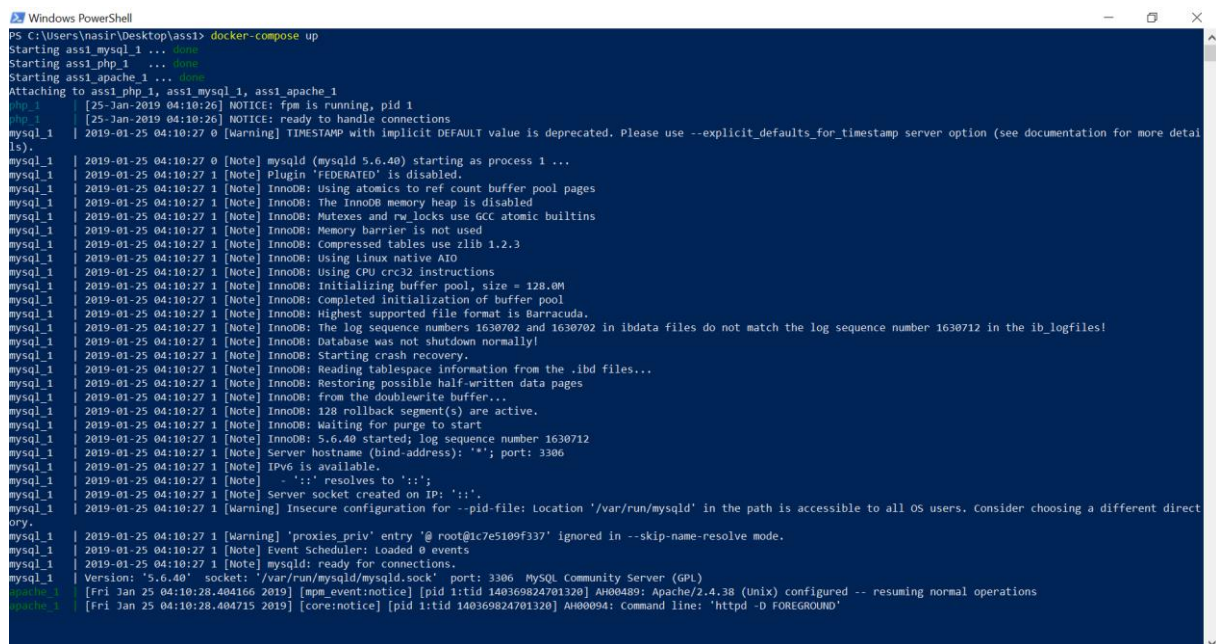
CREATE TABLE users (
    id INT(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(30) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    email VARCHAR(50) NOT NULL,
    age INT(3),
    location VARCHAR(50),
    date TIMESTAMP
);
```

Figure 8: Init.sql

Initially, image is built and containerize the application with a single command.

➤ `docker-compose up`

The path is set to the docker compose.yml file in the terminal and the above command is executed. The following figure explains the execution of above command.

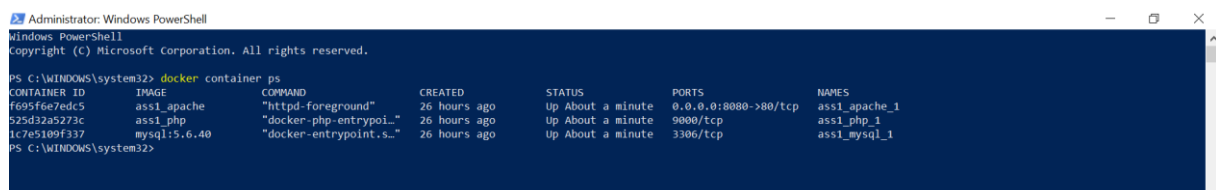


```
PS C:\Users\nasir\Desktop\ass1> docker-compose up
Starting ass1_mysql_1 ... done
Starting ass1_php_1 ... done
Starting ass1_apache_1 ... done
Attaching to ass1_php_1, ass1_mysql_1, ass1_apache_1
ass1_php_1 | [25-Jan-2019 04:10:26] NOTICE: fpm is running, pid 1
ass1_php_1 | [25-Jan-2019 04:10:26] NOTICE: ready to handle connections
ass1_mysql_1 | 2019-01-25 04:10:27 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp server option (see documentation for more details).
ass1_mysql_1 | 2019-01-25 04:10:27 0 [Note] mysqld (mysqld 5.6.40) starting as process 1 ...
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] Plugin 'FEDERATED' is disabled.
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Using atomics to ref count buffer pool pages
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: The InnoDB memory heap is disabled
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Mutexes and rw_locks use GCC atomic builtins
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Memory barrier is not used
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Compressed tables use zlib 1.2.3
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Using linux native AIO
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Using CPU crc32 instructions
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Initializing buffer pool, size = 128.0M
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Completed initialization of buffer pool
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Highest supported file format is Barracuda.
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: The log sequence numbers 1630702 and 1630702 in ibdata files do not match the log sequence number 1630712 in the ib_logfiles!
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Database was not shutdown normally!
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Starting crash recovery.
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Reading tablespace information from the .ibd files...
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Restoring possible half-written data pages
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: from the doublewrite buffer...
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: 128 rollback segment(s) are active.
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: Waiting for purge to start
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] InnoDB: 5.6.40 started; log sequence number 1630712
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] Server hostname (bind-address): '*'; port: 3306
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] IPv6 is available.
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] - '::' resolves to '::';
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] Server socket created on IP: '::'.
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Warning] Insecure configuration for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Warning] 'proxies_priv' entry '@root@1c7e5109f337' ignored in --skip-name-resolve mode.
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] Event Scheduler: Loaded 0 events
ass1_mysql_1 | 2019-01-25 04:10:27 1 [Note] mysqld: ready for connections.
ass1_mysql_1 | Version: '5.6.40' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server (GPL)
ass1_php_1 | [Fri Jan 25 04:10:28.404166 2019] [mpm_event:notice] [pid 1:tid 140369824701320] AH00489: Apache/2.4.38 (Unix) configured -- resuming normal operations
ass1_apache_1 | [Fri Jan 25 04:10:28.404715 2019] [core:notice] [pid 1:tid 140369824701320] AH00094: Command line: 'httpd -D FOREGROUND'
```

Figure 9: Execution of `docker-compose up`

Now, the image is built and run resulting in three containers running. The below figure describes the containers running. By using the command

➤ `docker container ps`



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\WINDOWS\system32> docker container ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
f695f6e7edc5       ass1_apache        "httpd-foreground"  26 hours ago       Up About a minute   0.0.0.0:8080->80/tcp ass1_apache_1
525d32a5273c       ass1_php           "docker-php-entrypoi"  26 hours ago       Up About a minute   9000/tcp           ass1_php_1
1c7e5109f337       mysql:5.6.40      "docker-entrypoint.s"  26 hours ago       Up About a minute   3306/tcp           ass1_mysql_1
```

Figure 10: Containers running

The application is running on the 8080 port. The application can be accessed in localhost i.e. 127.0.0.1:8080 port. The below explains the application running on 127.0.0.1:8080 port

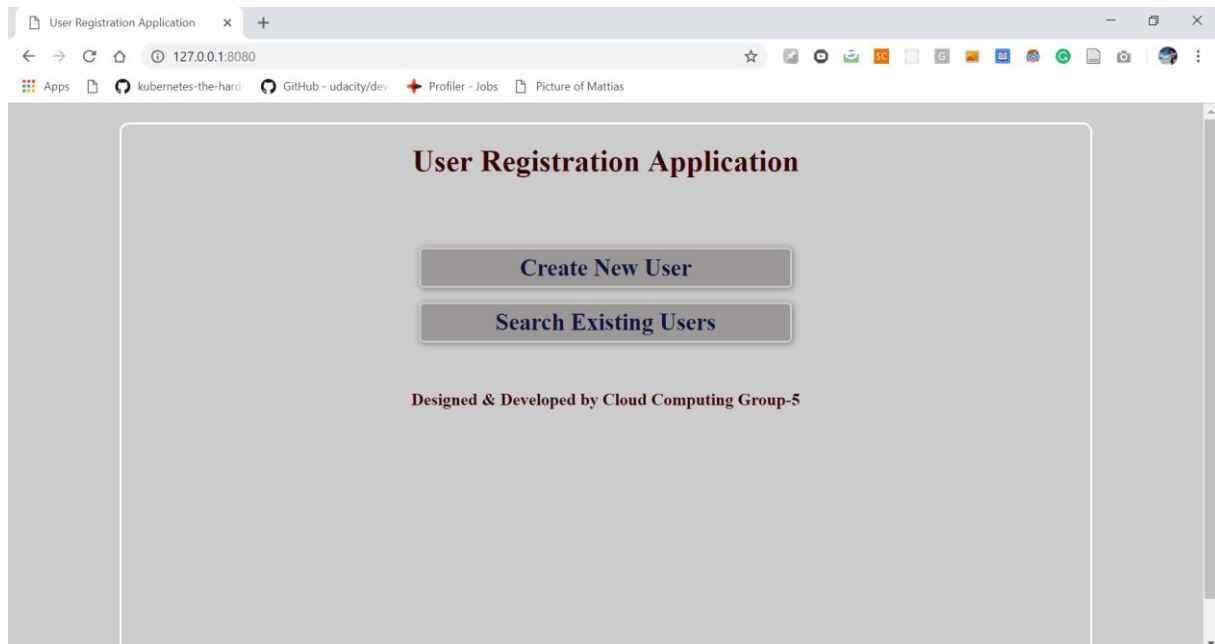


Figure 11: Application running on 127.0.0.1:8080 port

As explained above in the architecture, a user is registered, and containers are terminated. But the data is still persistent because of volumes. The below figures explain the container termination, volume created and running the containers by using volumes. All the commands must be executed as a root user.

```
PS C:\WINDOWS\system32> docker container stop f695f6e7edc5
f695f6e7edc5
PS C:\WINDOWS\system32> docker container stop 525d32a5273c
525d32a5273c
PS C:\WINDOWS\system32> docker container stop 1c7e5109f337
1c7e5109f337
PS C:\WINDOWS\system32> docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
PS C:\WINDOWS\system32>
```

Figure 12: containers stopped

In the above figure, it is known that all the containers are terminated. The volumes can be seen by using below command.

➤ `docker volume ls`



```
PS C:\WINDOWS\system32> docker volume ls
DRIVER          VOLUME NAME
local           5a081b7507c712c0aef5dcdaadb8454e8dc745bacb504b2706d9e39914bc11ee
local           harsha
PS C:\WINDOWS\system32>
```

Figure 12: Volumes created.

The containers can be restarted with the command.

- docker container start container name

```
PS C:\WINDOWS\system32> docker container start ass1_apache_1
ass1_apache_1
PS C:\WINDOWS\system32> docker container start ass1_php_1
ass1_php_1
PS C:\WINDOWS\system32> docker container start ass1_mysql_1
ass1_mysql_1
PS C:\WINDOWS\system32> docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
f695f6e7edc5   ass1_apache   "httpd-foreground"      27 hours ago   Up 37 seconds  0.0.0.0:8080->80/tcp     ass1_apache_1
525d32a5273c   ass1_php      "docker-php-entrypoi..." 27 hours ago   Up 23 seconds  9000/tcp                 ass1_php_1
1c7e5109f337   mysql:5.6.40  "docker-entrypoint.s..." 27 hours ago   Up 11 seconds  3306/tcp                 ass1_mysql_1
PS C:\WINDOWS\system32>
```

Figure 13: Restarting docker containers.

By using volumes, the data is persistent. The below figure explains the persistent data ran by the container.

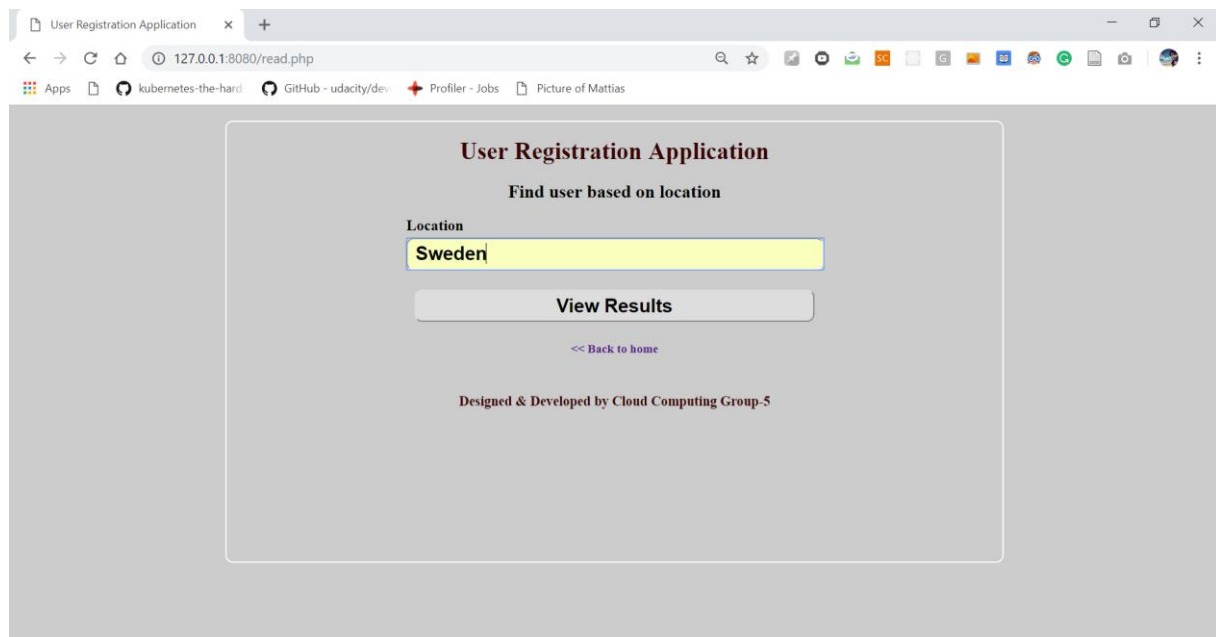


Figure 14: Search by location

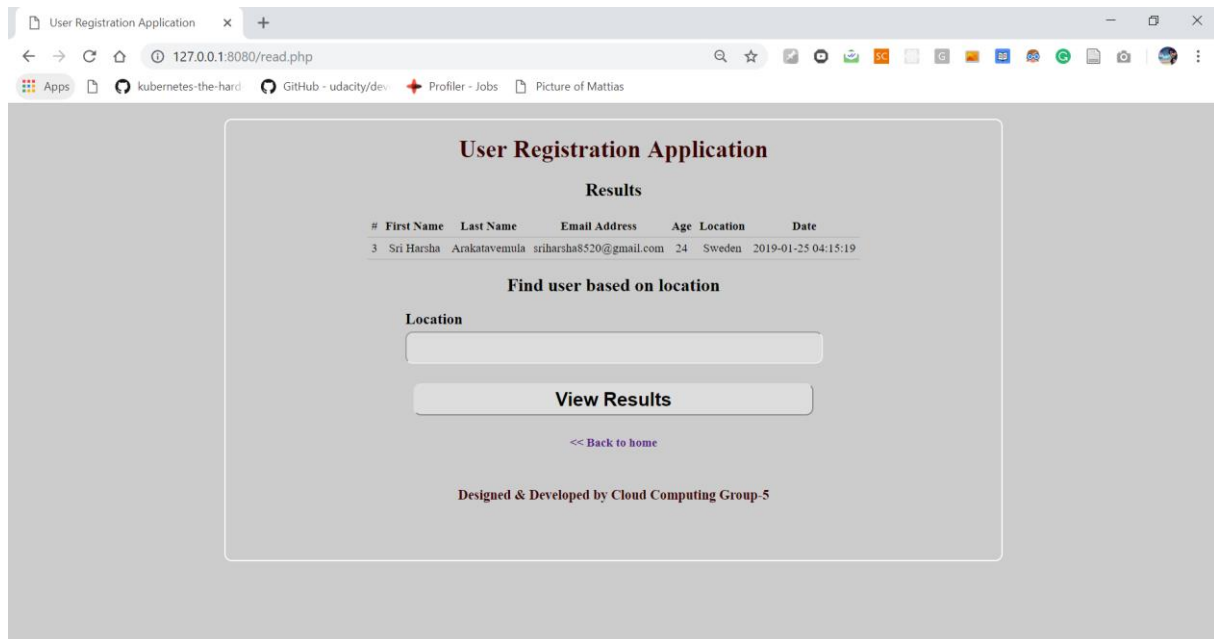


Figure 15: Persistent data by volumes.

Thus, the volumes are used to persistent data ran by a container. By using bind mount file or a directory is hosted from the host machine is mounted into a container. Two folders have been mounted on host machine to two different containers.

.idea	12/18/2018 10:42	File folder	
db	12/20/2018 12:01	File folder	
test	12/20/2018 12:09	File folder	
index.php	12/18/2018 9:55 PM	PHP File	2 KB
server.php	12/20/2018 12:02	PHP File	1 KB
style.css	7/13/2017 12:00 PM	Cascading Style Sh...	2 KB

Figure 16: Folder with name test is on host.

```

Windows PowerShell
PS D:\Program\XAMPP\xampp\htdocs\phpweb> docker exec -it htdocs_web_1 /bin/bash
root@phpweb:/var/www/html# ls
db index.php server.php style.css test
root@phpweb:/var/www/html#

```

Figure 17: Folder test on the container.

```
Windows PowerShell
PS D:\Program\XAMPP\xampp\htdocs\phpweb> docker exec -it htdocs_db_1 /bin/bash
root@f5e59e5be4fb:/# ls
bin boot dev docker-entrypoint-initdb.d entrypoint.sh etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
root@f5e59e5be4fb:/# cd docker-entrypoint-initdb.d/
root@f5e59e5be4fb:/docker-entrypoint-initdb.d# ls
db.sql db1.sql
root@f5e59e5be4fb:/docker-entrypoint-initdb.d#
```

Figure 18: Init.sql file on container.

Now, a docker service called swarm is used to create a cluster. This cluster contains three docker machines, one manager named as manager1 and two workers named as worker1 and worker2. Thus, there are three docker machines are running. To create a docker machine the following command is used. The same command is used to create three docker machines. The following figure explains the docker machines created.

- docker-machine create -d hyperv --hyperv-virtual-switch "Primary Virtual Switch" manager1
- docker-machine create -d hyperv --hyperv-virtual-switch "Primary Virtual Switch" worker1
- docker-machine create -d hyperv --hyperv-virtual-switch "Primary Virtual Switch" worker2

```
PS C:\WINDOWS\system32> docker-machine create -d hyperv --hyperv-virtual-switch "Primary Virtual Switch" manager1
Running pre-create checks...
Creating machine...
(manager1) Copying C:\Users\nasir\.docker\machine\cache\boot2docker.iso to C:\Users\nasir\.docker\machine\machines\manager1\boot2docker.iso...
(manager1) Creating SSH key...
(manager1) Creating VM...
(manager1) Using switch "Primary Virtual Switch"
(manager1) Creating VHD...
(manager1) Starting VM...
(manager1) Waiting for host to start...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: C:\Program Files\Docker\Docker\Resources\bin\docker-machine.exe env manager1
```

Figure 19: Creating manager1

By using the below command, the docker machines which are created can be listed.

- docker-machine ls

The below figure shows the docker machines created.

```
PS C:\WINDOWS\system32> docker-machine ls
NAME      ACTIVE  DRIVER  STATE   URL                     SWARM   DOCKER  ERRORS
manager1  -       hyperv  Running tcp://192.168.1.110:2376  -       v18.09.1
worker1   -       hyperv  Running tcp://192.168.1.111:2376  -       v18.09.1
worker2   -       hyperv  Running tcp://192.168.1.112:2376  -       v18.09.1
PS C:\WINDOWS\system32>
```

Figure 20: List of docker machines created.

There are totally three docker machines running one manager1 and two worker1 and worker2. Now, cluster is created by making manager1 as manager and worker1, worker2 as workers. By executing the below command, manager1 is made as manager.

- `docker-machine manager1 "docker swarm init --advertise-addr 192.168.1.107"`

```
PS C:\WINDOWS\system32> docker-machine ssh manager1 "docker swarm init --advertise-addr 192.168.1.107"
Swarm initialized: current node (fwphhs5y5b2jfjvkmm75rrea9) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-22j469ac6nueq5w0j8or8u8eyn6u8qyd4y55pu51w62xlcu7b5-cbmbk1i895nkt6cnxeachgchf 192.168.1.107:2377

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

PS C:\WINDOWS\system32>
```

*Figure 21: making manager1 as manager.*

In the above figure, it is shown that manager1 is manager. Now manager can join workers in his cluster by running the above-mentioned command in the figure in both worker1 and worker2. To enter into worker1 and worker2 the following command is used.

- `docker-machine ssh worker1`

```
PS C:\WINDOWS\system32> docker-machine ssh worker1
( '>')
/) TC (\   Core is distributed with ABSOLUTELY NO WARRANTY.
(/-__-_-\)   www.tinycorelinux.net

docker@worker1:~$
```

*Figure 22: Entering into worker1*

- `docker-machine ssh worker2`

```
PS C:\WINDOWS\system32> docker-machine ssh worker2
( '>')
/) TC (\   Core is distributed with ABSOLUTELY NO WARRANTY.
(/-__-_-\)   www.tinycorelinux.net

docker@worker2: $
```

*Figure 23: Entering into worker2*

- `docker-machine ssh manager1`

```
PS C:\WINDOWS\system32> docker-machine ssh manager1
( '>')
/) TC (\   Core is distributed with ABSOLUTELY NO WARRANTY.
(/-__-_-\)   www.tinycorelinux.net

docker@manager1: $
```

*Figure 24: Entering into manager1*

```
docker@worker1:~$ docker swarm join --token SWMTKN-1-2rejvo0h2uwbe7m5xbwyul17819f03fpx52btpd2h6ekzgkpfh-6543z1rnizu4kc21k7ygvcl6l 192.168.1.113:2377
This node joined a swarm as a worker.
docker@worker1:~$
```

*Figure 25: Worker1 joined swarm*

```
docker@worker2:~$ docker swarm join --token SWMTKN-1-2rejvo0h2uwbe7m5xbwyul17819f03fpx52btpd2h6ekzgkpfh-6543z1rnizu4kc21k7ygvcl6l 192.168.1.113:2377
This node joined a swarm as a worker.
docker@worker2:~$
```

*Figure 26: Worker2 joined swarm*

All the commands can be running in manager1 but not in worker1 and worker2 because only manager has the permissions. By executing the following command in manager1, information about the swarm cluster created can be known.

➤ docker swarm info

```
docker@manager1:~$ docker info
Containers: 0
Running: 0
Paused: 0
Stopped: 0
Images: 0
Server Version: 18.09.1
Storage Driver: overlay2
Backing Filesystem: extfs
Supports d_type: true
Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
Volume: local
Network: bridge host macvlan null overlay
Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: active
NodeID: 4advvvygyvmmldhoxjqxdxy
Is Manager: true
ClusterID: s5055ccdefpcevfvucgfrmjzj
Managers: 1
Nodes: 3
Default Address Pool: 10.0.0.0/8
SubnetSize: 24
Orchestration:
Task History Retention Limit: 5
Raft:
Snapshot Interval: 10000
Number of Old Snapshots to Retain: 0
Heartbeat Tick: 1
Election Tick: 10
Dispatcher:
Heartbeat Period: 5 seconds
CA Configuration:
Expiry Duration: 3 months
Force Rotate: 0
```

*Figure 27: Information about swarm*

By executing the following command in manager1, the status of the nodes in swarm can be known.

- docker node ls

```
docker@manager1: $ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
4advvvygyvmdlhonxjqxdsy *	manager1	Ready	Active	Leader	18.09.1
y1g12gv63jrqu49p903gorn5	worker1	Ready	Active		18.09.1
ggx3124bowg9ioghiz9t5n3ig	worker2	Ready	Active		18.09.1

```
docker@manager1: $
```

Figure 28: Status of the nodes.

The containers of the application running are pushed into the docker hub repository. The following command is used to tag the container and push into the repository.

- docker tag ass1\_apache harsha2117/grp5:part1
- docker push harsha2117/grp5:part1

```
PS C:\WINDOWS\system32> docker tag ass1_apache harsha2117/grp5:part1
PS C:\WINDOWS\system32> docker push harsha2117/grp5:part1
The push refers to repository [docker.io/harsha2117/grp5]
f1a9ce38ed2b: Mounted from harsha2117/fem
25ed00d5f165: Mounted from harsha2117/fem
53890e88216a: Mounted from harsha2117/fem
3769ae1289cb: Mounted from harsha2117/fem
b5af98ddf2d6: Mounted from harsha2117/fem
713501a5f7af: Mounted from harsha2117/fem
51e5127a08e3: Mounted from harsha2117/fem
7bff100f35cb: Mounted from harsha2117/fem
part1: digest: sha256:1efd89df1f3e41bb2e40bbcf11613c69c13773f9be96e66a7eb390650f2dcc568 size: 1988
PS C:\WINDOWS\system32>
```

Figure 29: Pushing the container into repository

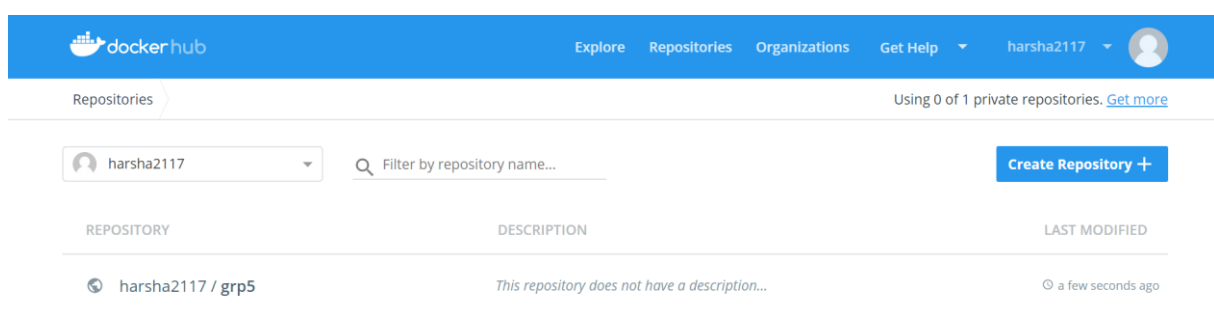


Figure 30: Repository created with name grp5

By using the below command, the contained from the repository is run on the swarm.

- docker service create --replicas 3 -p 8080:80 --aserver harsha2117/grp5:part1

By using the below command, the number of replicas running can be seen.

- docker service ls

```

docker@manager1: $ docker service ls
ID            NAME      MODE      REPLICAS  IMAGE      PORTS
15emotqcyqd0 aserver   replicated 3/3        harsha2117/grp5:part1 *:80->80/tcp
docker@manager1: $

```

Figure 31: Number of replicas created

By using the below command, the services running on the swarm are shown.

- docker service ps aserver

```

docker@manager1: $ docker service ps aserver
ID            NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE      ERROR      PORTS
6hajfgnvsu1  aserver.1 harsha2117/grp5:part1 worker2      Running         Running 3 minutes ago
wk4letjfhgzy aserver.2 harsha2117/grp5:part1 manager1     Running         Running 3 minutes ago
yqn1qolh18n6 aserver.3 harsha2117/grp5:part1 worker1      Running         Running 3 minutes ago
docker@manager1: $

```

Figure 32: Services running

Now, scale service up and scale service down is performed. Scale service up is done by increasing the replicas from 3 to 5 by using the below command.

- docker service scale aserver=5

```

docker@manager1: $ docker service scale aserver=5
aserver scaled to 5
overall progress: 5 out of 5 tasks
1/5: running [=====>]
2/5: running [=====>]
3/5: running [=====>]
4/5: running [=====>]
5/5: running [=====>]
verify: Service converged
docker@manager1: $

```

Figure 33: scaling up

The services running can be seen by using the below commands.

- docker service ls

```

docker@manager1: $ docker service ls
ID            NAME      MODE      REPLICAS  IMAGE      PORTS
15emotqcyqd0 aserver   replicated 5/5        harsha2117/grp5:part1 *:80->80/tcp
docker@manager1: $

```

Figure 34: Five replicas are running

- docker service ps aserver

```

docker@manager1: $ docker service ps aserver
ID            NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE      ERROR      PORTS
6hajfgnvsu1  aserver.1 harsha2117/grp5:part1 worker2      Running         Running 8 minutes ago
wk4letjfhgzy aserver.2 harsha2117/grp5:part1 manager1     Running         Running 8 minutes ago
yqn1qolh18n6 aserver.3 harsha2117/grp5:part1 worker1      Running         Running 8 minutes ago
aaknb4q8i3k  aserver.4 harsha2117/grp5:part1 worker1      Running         Running 2 minutes ago
532xdfvughc6 aserver.5 harsha2117/grp5:part1 manager1     Running         Running 2 minutes ago
docker@manager1: $

```

Figure 35: Services running

Scale service down is done by decreasing the replicas from 5 to 2 by using the below command.

- docker service scale aserver=2

```

docker@manager1:~$ docker service scale aserver=2
aserver scaled to 2
overall progress: 2 out of 2 tasks
1/2: running  [=====>]
2/2: running  [=====>]
verify: Service converged
docker@manager1:~$

```

*Figure 36: Scaling down*

The service running can be seen by using the below commands.

➤ docker service ls

```

docker@manager1:~$ docker service ls
ID                NAME      MODE                REPLICAS        IMAGE              PORTS
l5emotqcyqdo     aserver   replicated          2/2             harsha2117/grp5:part1  *:80->80/tcp
docker@manager1:~$

```

*Figure 37: Two replicas are running.*

➤ docker service ps aserver

```

docker@manager1:~$ docker service ps aserver
ID                NAME      IMAGE              NODE             DESIRED STATE       CURRENT STATE      ERROR             PORTS
6hajfgnvnsl      aserver.1  harsha2117/grp5:part1  worker2          Running              Running 11 minutes ago
wk4letjfhgzy     aserver.2  harsha2117/grp5:part1  manager1         Running              Running 11 minutes ago
docker@manager1:~$

```

*Figure 38: Services running*

Thus, scale service up and scale service down is done.