DEPARTMENT OF INFORMATION TECHNOLOGY

Semester	S.E. Semester IV – Information Technology Engineering
Subject	Computer Networks and Network Design Lab
Subject Professor In-charge	Unnati Gohil
Assisting Teachers	-
Laboratory	MS Teams

Student Name	Sanika Kate	
Roll Number	22101A2005	
Grade and Subject Teacher's Signature		

Experiment Number	3	
Experiment Title	Installing Docker	
Resources / Apparatus Required	Hardware: Basic Desktop with Windows or Linux.	Software: Java/ Python/Wireshark/Docker
Objectives (Skill Set / Knowledge Tested / Imparted)		

Theory:

What is Docker?

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host. You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

What is Container?

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers – images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

What is a Docker image?

A Docker image is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker Container, like a template. Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments.

Docker images have multiple layers, each one originates from the previous layer but is different from it. The layers speed up the Docker while increasing reusability and decreasing disk use. Image layers are also read-only files. Once a container is created, a writable layer is added on top of the unchangeable images, allowing a user to make changes.

COMMANDS

1. **docker version**

The version command prints the current version number for all independently versioned Docker components.

2. **Docker pull ubuntu**

This command will instruct Docker to download the latest version of the Ubuntu image from the Docker Hub repository

3. **docker images**

This command will display a list of Docker images along with their repository, tag, image ID, creation date, and size. The list will include both official Docker images and any custom images you may have built or pulled from Docker Hub.

4. **docker run -it -d ubuntu**

The command you've provided, docker run -it -d ubuntu, will run a Docker container based on the official Ubuntu image with an interactive terminal (-it) and in detached mode (-d).

5. **docker ps**

The docker ps command is used to list the running Docker containers on your system. When you run this command in your terminal, it will display a list of containers along with their relevant information such as container ID, image, status, ports, names, and more.

6. **docker exec -it <container_name_or_id> d**

This will execute the "d" command inside the specified container, provided that the "d" command is available within the container's filesystem.

7. **ls**

This command will list all the directories within the docker container.

8. **cd var**

This command will create directory in the docker container

9. **apt-get install nano**

This command will install Nano Text Editor.

10. **docker commit <container_name_or_id> exmple58/ubuntu**

The docker commit command allows you to create a new Docker image from an existing container, effectively capturing the current state of that container as an image.

11. **docker run -it -d -p 82:80 example58/ubuntu**

This command will create a new container from the "example58/ubuntu" Docker image, and it will run a web server on port 80 inside the container. You'll be able to access this web server by visiting http://localhost:82 in your web browser because you've mapped port 82 on your host to port 80 in the container.

12. **docker exec -it <container_name_or_id> bash**

This command will open an interactive shell session inside the specified container, allowing you to execute commands and interact with the container's file system and environment.

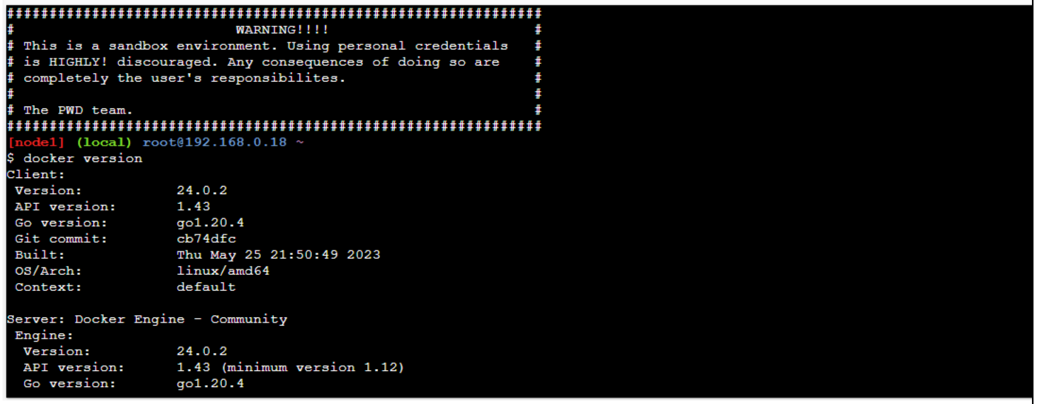
13. docker login

To log in to Docker Hub or another Docker registry, you can use the docker login command.

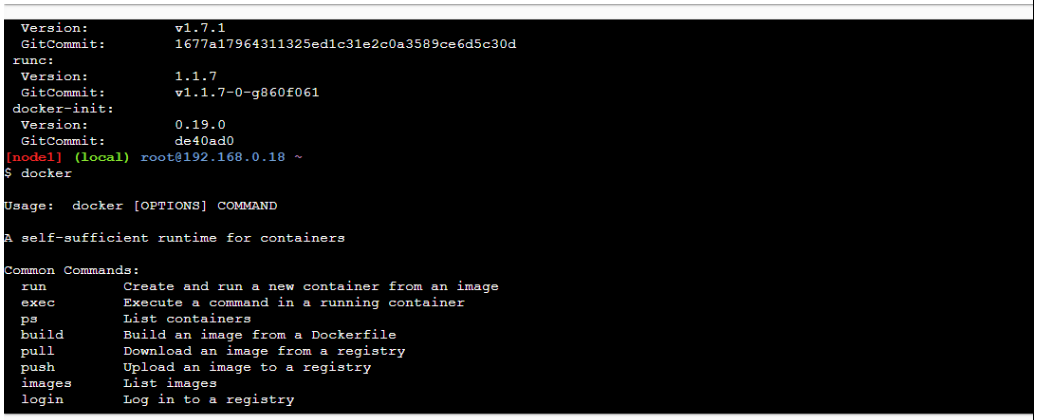
14. docker rmi -f exmple58/ubuntu

To force the removal of a Docker image with the name "exmple58/ubuntu," you can use the docker rmi command with the -f (force) option.

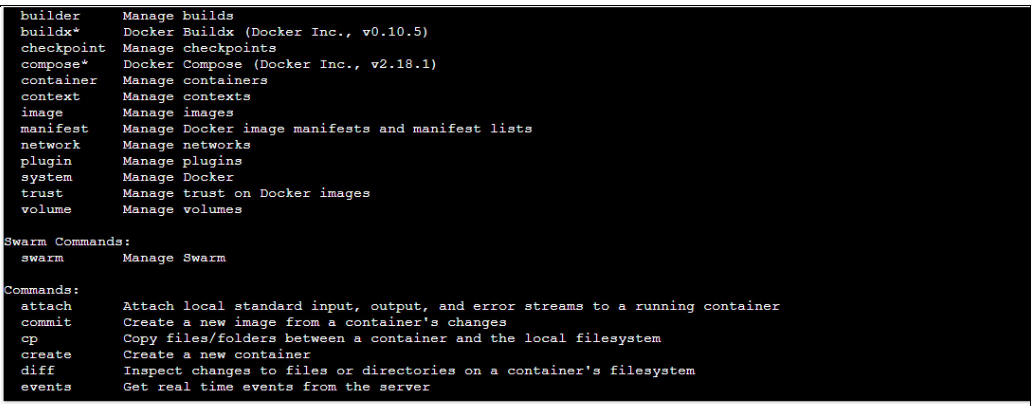
Output



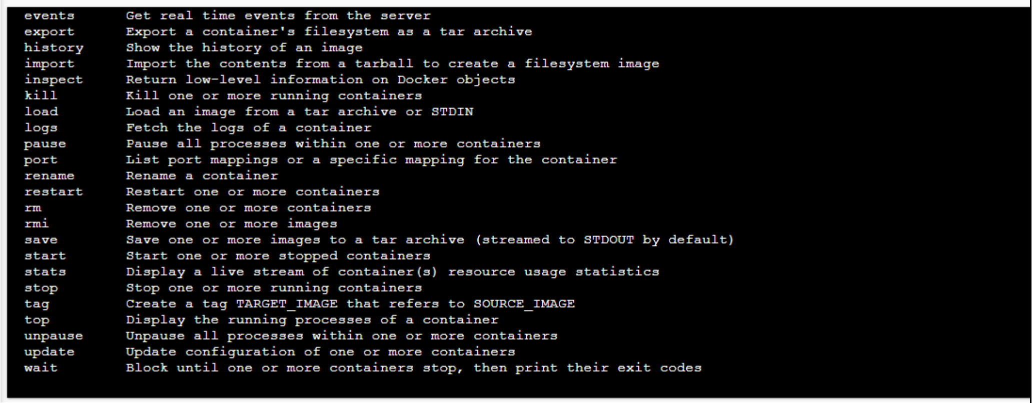
```
###################################################################
#                          WARNING!!!!                            #
# This is a sandbox environment. Using personal credentials       #
# is HIGHLY! discouraged. Any consequences of doing so are        #
# completely the user's responsibilites.                          #
#                                                                 #
# The PWD team.                                                   #
###################################################################
[node1] (local) root@192.168.0.18 ~
$ docker version
Client:
 Version:           24.0.2
 API version:       1.43
 Go version:        go1.20.4
 Git commit:        cb74dfc
 Built:             Thu May 25 21:50:49 2023
 OS/Arch:           linux/amd64
 Context:           default

Server: Docker Engine - Community
 Engine:
  Version:          24.0.2
  API version:      1.43 (minimum version 1.12)
  Go version:       go1.20.4
```



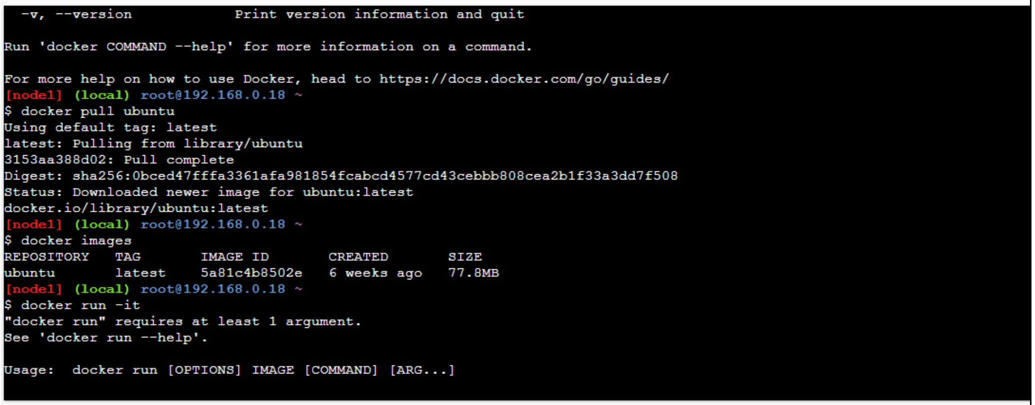
```
  Version:          v1.7.1
  GitCommit:        1677a17964311325ed1c31e2c0a3589ce6d5c30d
 runc:
  Version:          1.1.7
  GitCommit:        v1.1.7-0-g860f061
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
[node1] (local) root@192.168.0.18 ~
$ docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Log in to a registry
```



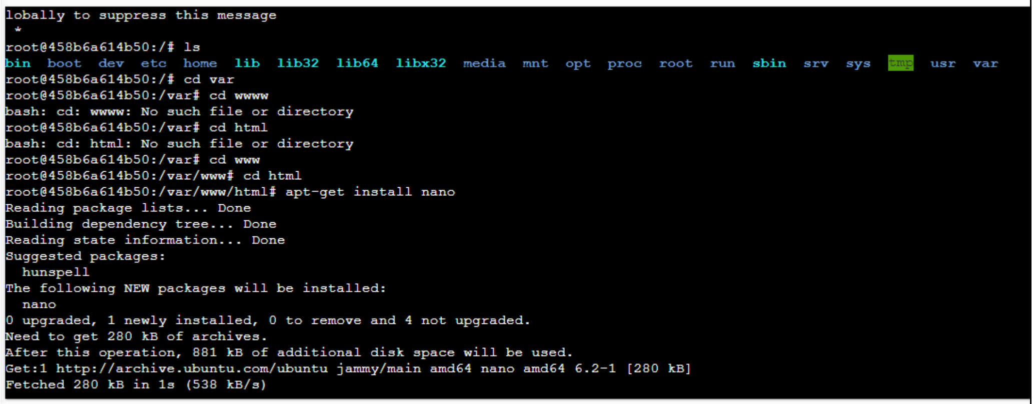
```
  builder     Manage builds
  buildx*     Docker Buildx (Docker Inc., v0.10.5)
  checkpoint  Manage checkpoints
  compose*    Docker Compose (Docker Inc., v2.18.1)
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  plugin      Manage plugins
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Swarm Commands:
  swarm       Manage Swarm

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
```



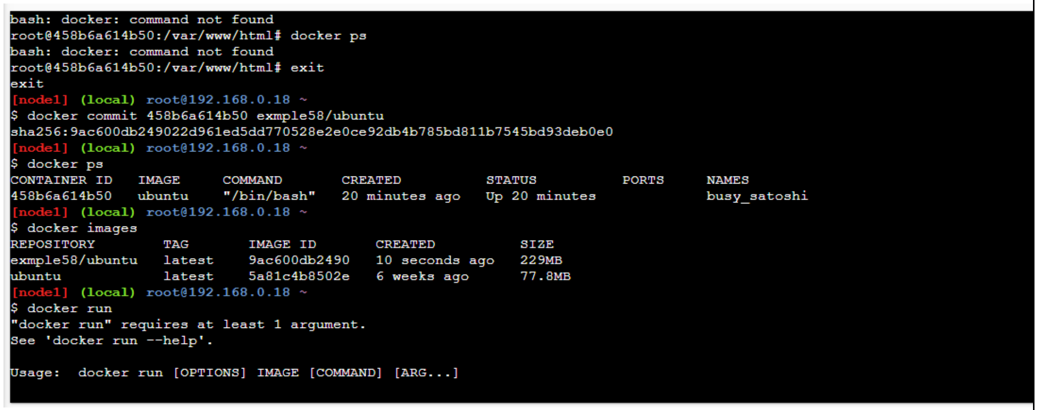
```
  events      Get real time events from the server
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes
```



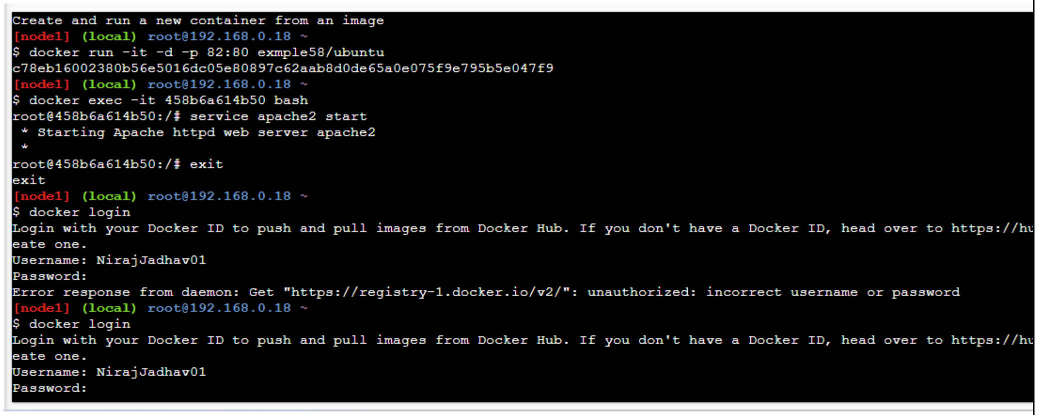
```
  -v, --version            Print version information and quit

Run 'docker COMMAND --help' for more information on a command.

For more help on how to use Docker, head to https://docs.docker.com/go/guides/
[node1] (local) root@192.168.0.18 ~
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
3153aa388d02: Pull complete
Digest: sha256:0bced47fffa3361afa981854fcabcd4577cd43cebbb808cea2b1f33a3dd7f508
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
[node1] (local) root@192.168.0.18 ~
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       latest    5a81c4b8502e   6 weeks ago    77.8MB
[node1] (local) root@192.168.0.18 ~
$ docker run -it
"docker run" requires at least 1 argument.
See 'docker run --help'.

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```



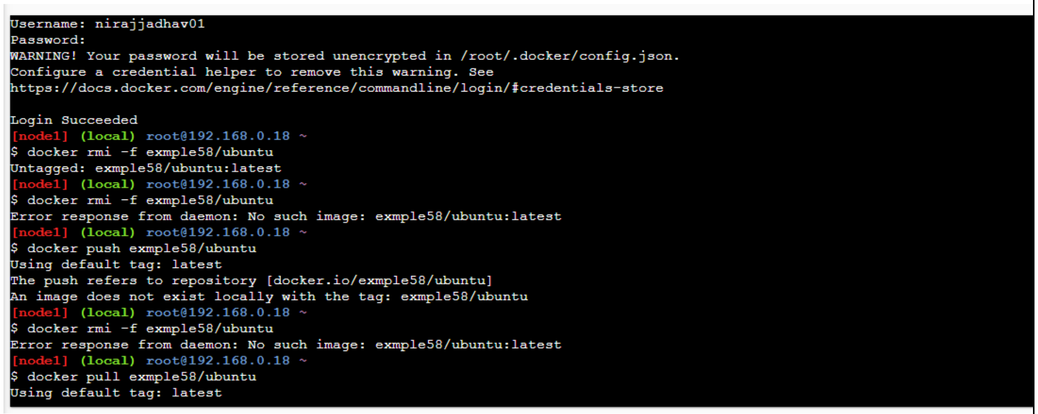
```
lobally to suppress this message
 *
root@458b6a614b50:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@458b6a614b50:/# cd var
root@458b6a614b50:/var# cd wwww
bash: cd: wwww: No such file or directory
root@458b6a614b50:/var# cd html
bash: cd: html: No such file or directory
root@458b6a614b50:/var# cd www
root@458b6a614b50:/var/www# cd html
root@458b6a614b50:/var/www/html# apt-get install nano
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Suggested packages:
  hunspell
The following NEW packages will be installed:
  nano
0 upgraded, 1 newly installed, 0 to remove and 4 not upgraded.
Need to get 280 kB of archives.
After this operation, 881 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/main amd64 nano amd64 6.2-1 [280 kB]
Fetched 280 kB in 1s (538 kB/s)
```



```
bash: docker: command not found
root@458b6a614b50:/var/www/html# docker ps
bash: docker: command not found
root@458b6a614b50:/var/www/html# exit
exit
[node1] (local) root@192.168.0.18 ~
$ docker commit 458b6a614b50 exmple58/ubuntu
sha256:9ac600db249022d961ed5dd770528e2e0ce92db4b785bd811b7545bd93deb0e0
[node1] (local) root@192.168.0.18 ~
$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
458b6a614b50   ubuntu    "/bin/bash"   20 minutes ago   Up 20 minutes             busy_satoshi
[node1] (local) root@192.168.0.18 ~
$ docker images
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
exmple58/ubuntu   latest    9ac600db2490   10 seconds ago   229MB
ubuntu            latest    5a81c4b8502e   6 weeks ago      77.8MB
[node1] (local) root@192.168.0.18 ~
$ docker run
"docker run" requires at least 1 argument.
See 'docker run --help'.

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```



```
Create and run a new container from an image
[node1] (local) root@192.168.0.18 ~
$ docker run -it -d -p 82:80 exmple58/ubuntu
c78eb16002380b56e5016dc05e80897c62aab8d0de65a0e075f9e795b5e047f9
[node1] (local) root@192.168.0.18 ~
$ docker exec -it 458b6a614b50 bash
root@458b6a614b50:/# service apache2 start
 * Starting Apache httpd web server apache2
 *
root@458b6a614b50:/# exit
exit
[node1] (local) root@192.168.0.18 ~
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hu
eate one.
Username: NirajJadhav01
Password:
Error response from daemon: Get "https://registry-1.docker.io/v2/": unauthorized: incorrect username or password
[node1] (local) root@192.168.0.18 ~
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hu
eate one.
Username: NirajJadhav01
Password:
```



```
Username: nirajjadhav01
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[node1] (local) root@192.168.0.18 ~
$ docker rmi -f exmple58/ubuntu
Untagged: exmple58/ubuntu:latest
[node1] (local) root@192.168.0.18 ~
$ docker rmi -f exmple58/ubuntu
Error response from daemon: No such image: exmple58/ubuntu:latest
[node1] (local) root@192.168.0.18 ~
$ docker push exmple58/ubuntu
Using default tag: latest
The push refers to repository [docker.io/exmple58/ubuntu]
An image does not exist locally with the tag: exmple58/ubuntu
[node1] (local) root@192.168.0.18 ~
$ docker rmi -f exmple58/ubuntu
Error response from daemon: No such image: exmple58/ubuntu:latest
[node1] (local) root@192.168.0.18 ~
$ docker pull exmple58/ubuntu
Using default tag: latest
```

Conclusion	In this way we installed docker as well as created a docker image.