CHAPTER 1

INTRODUCTION

Containers are a type of virtualization technology that enables the reuse of certain system resources that are not required to be duplicated. This can add a few different benefits depending on your needs. The most significant difference is that you are going to being able to spin up a higher quantity of containers on the same hardware or be able to assign more resources to your containers versus traditional virtual machines.

There are a number of components that are important for an OS to function properly. The parts we are going to focus on are the ones that are handled differently between traditional Virtual Machine (VM) configurations and container implementations. When power is run to a computer system (typically when the power button is turned on, or a piece of equipment is plugged in) the very first thing that happens is power runs to the NVRAM (Non-Volatile RAM), which is a particular kind of chip that stores a dedicated section of code called the BIOS (Basic Input/Output System). The BIOS is a set of code that loads a basic set of drivers that don’t need to be changed often. The BIOS is the reason you are able to use your keyboard and anything else that interfaces with the computer before the actual operating system is loaded. The BIOS is configured to look for an MBR (Master Boot Record) in the first section of a formatted drive. Despite not technically being part of the operating system, we can think of the bootloader as the first part of the OS. The bootloader sits on a special section of the boot disk called the MBR (Master Boot Record) and contains executable code that kicks off the booting process of the OS code. Some popular bootloaders that you may have seen are LILO (LInux LOader), LOADLIN (LOAD LINux), GRUB (GRand Unified Bootloader) and of course, the Windows bootloader. Bootloaders can point to multiple operating systems, or can even be chained together in some cases. This is most commonly done when dual-booting an OS where modifying the bootloader can cause problems. For example, running GRUB to run Linux, but passing to the Windows bootloader if the Windows OS is selected from the GRUB menu). The function of the bootloader is to load the OS Kernel. The Kernel is the core set of code that handles interfacing with the lower levels of the system and allows all of the abstraction that modern operating systems offer, such as graphical windowing systems (commonly referred to as GUI’s, Desktop Environments, or Window Managers, depending on which part is being referenced). The kernel of an operating system contains the core group of libraries and binaries required to run the OS. This set of code includes all of the underlying tools used by every other application running under the OS. This combination of code and resources is commonly referred to as Bins/Libs, for binaries and libraries.

In order to execute the bootloader and operating system in a traditional virtual machine environment, a Virtualization layer sits on top of the Host OS (the Operating system that you have installed onto the physical device). This layer then emulates hardware, presenting a virtualized version of a set of hardware to the Guest OS (the OS being installed inside of the Virtualization Layer). The virtualization layer essentially presents a fake version of physical hardware, copying how the hardware would usually function as closely as possible. This means you are presenting an emulated storage device for the boot loader, kernel, the OS bins/libs, and any other default applications installed by the Guest OS. Below is a simple diagram, but there will be differences depending on the type of virtualization. Notably Para virtualization, which is a specialized kernel that is configured to send commands to a hypervisor rather than directly to hardware. The nuanced differences are difficult to explain, but the core differences come down to the following. If the guest is aware that it is running in a virtualized environment, it purposefully routes requests to a hypervisor. If instead, the Host OS makes special allocations to run a higher execution ring, it allows the guest to execute in ring 0. This is really where containers shine, When we load a container layer, such as Docker Engine or Google Container Engine, the engine does not provide a typical virtualization. Instead, it provides the guest OS a wrapper to access the Host OS's existing kernel, scheduler, and memory manager.

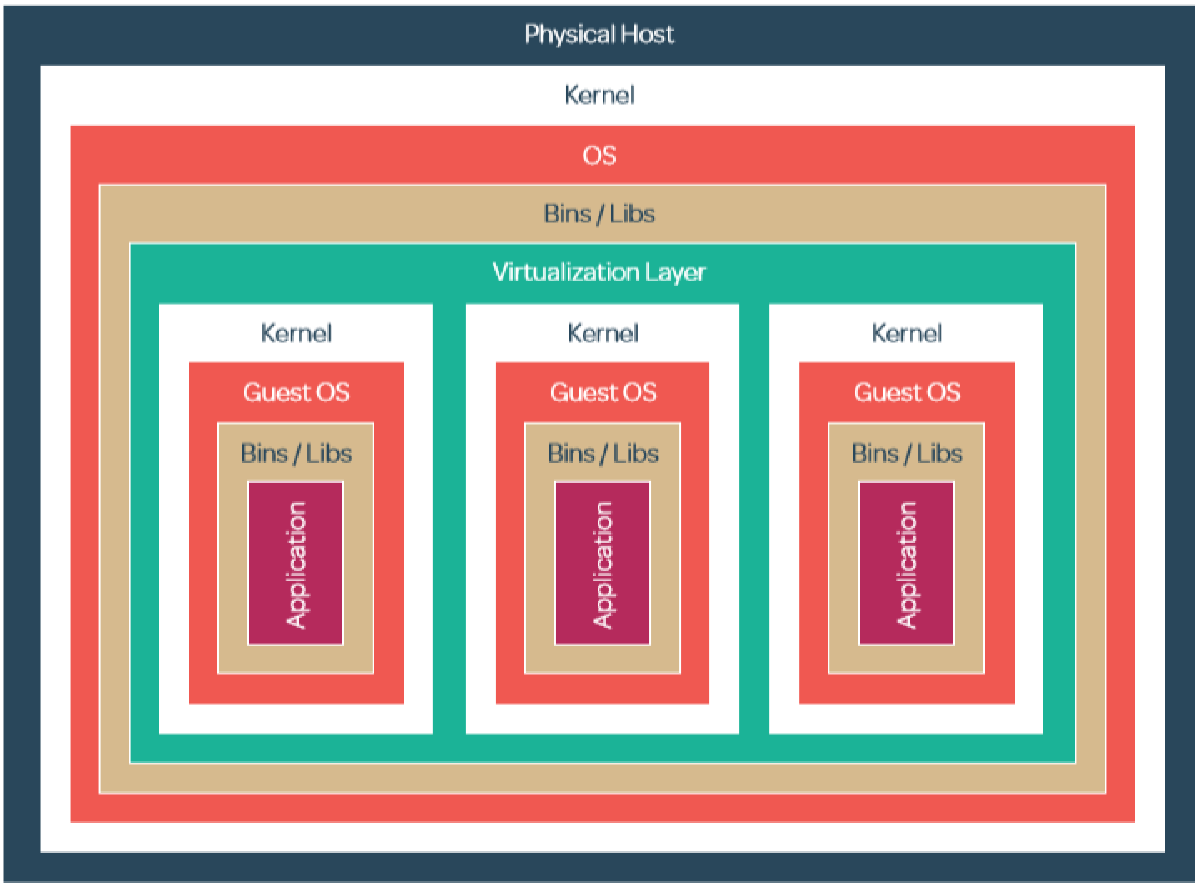


Figure 1.1: Traditional Virtualization

In reality, containers are not running a virtualized OS at all. Instead, they are allowed limited access to the existing kernel, binaries, and libraries that exist on the host operating system. The engine loads any additional bins/libs that are required by the application and groups the running processes of the container together, very similarly to the way that chroot jails function. The reason containers can run in this manner (as opposed to traditional styles of virtualization) is in large part due to cgroups and namespaces.

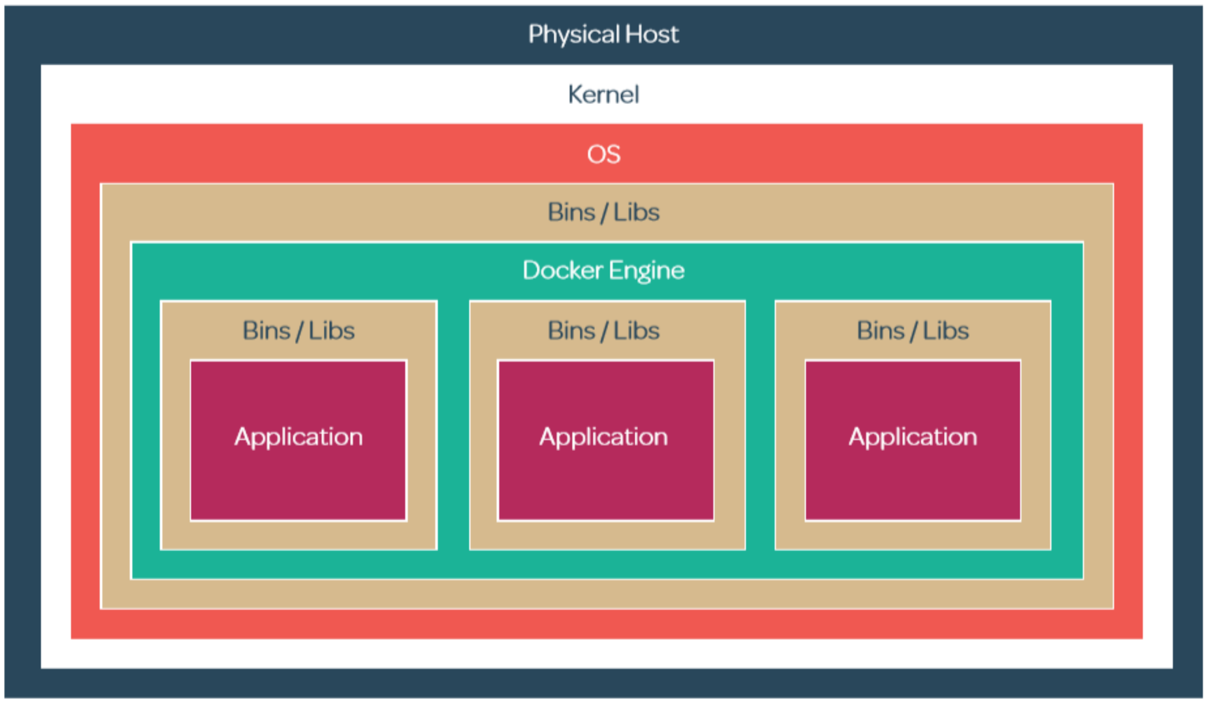


Figure 1.2: Container Architecture

Docker is an open-source project based on Linux containers. It uses Linux Kernel features like namespaces and control groups to create containers on top of an operating system.

Containers are far from new; Google has been using their own container technology for years. Others Linux container technologies include Solaris Zones, BSD jails, and LXC, which have been around for many years.

Docker is used for many reasons

1. **Ease of use:** Docker has made it much easier for anyone developers, systems admins, architects and others to take advantage of containers in order to quickly build and test portable applications. It allows anyone to package an application on their laptop, which in turn can run unmodified on any public cloud, private cloud, or even bare metal. The mantra is: “build once, run anywhere.”

2. **Speed:** Docker containers are very lightweight and fast. Since containers are just sandboxed environments running on the kernel, they take up fewer resources. You can create and run a Docker container in seconds, compared to VMs which might take longer because they have to boot up a full virtual operating system every time.

3. **Docker Hub:** Docker users also benefit from the increasingly rich ecosystem of Docker Hub, which you can think of as an “app store for Docker images.” Docker Hub has tens of thousands of public images created by the community that are readily available for use. It’s incredibly easy to search for images that meet your needs, ready to pull down and use with little-to-no modification.

4. **Modularity and Scalability:** Docker makes it easy to break out your application’s functionality into individual containers. For example, you might have your Postgres database running in one container and your Redis server in another while your Node.js app is in another. With Docker, it’s become easier to link these containers together to create your application, making it easy to scale or update components independently in the future.

1.1. Fundamentals of Docker

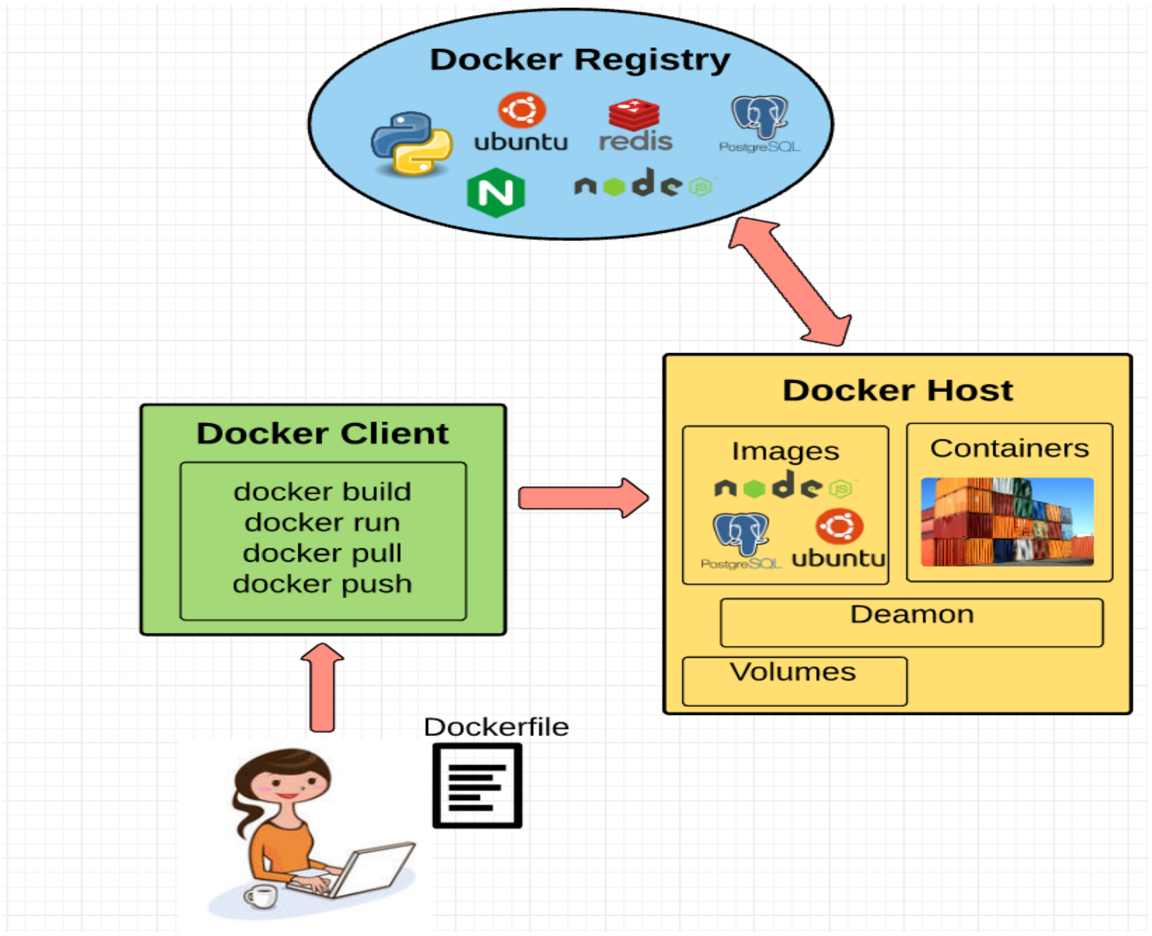


Figure.1.1.1: Fundamentals of Docker

Docker engine is the layer on which Docker runs. It’s a lightweight runtime and tooling that manages containers, images, builds, and more. It runs natively on Linux systems and is made up of:

1. A Docker Daemon that runs in the host computer.
2. A Docker Client that then communicates with the Docker Daemon to execute commands.
3. A REST API for interacting with the Docker Daemon remotely.

1.1.1. Docker Client

The Docker Client is what you, as the end-user of Docker, communicate with. Think of it as the UI for Docker. For example, when you do docker build iampeekay/someImage   you are communicating to the Docker Client, which then communicates your instructions to the Docker Daemon. Docker can be explained as a client and server based application, as depicted in Figure. The docker server gets the request from the docker client and then processes it accordingly. The complete RESTful (Representational state transfer) API and a command line client binary are shipped by docker. Docker daemon/server and docker client can be run on the same machine or a local docker client can be connected with a remote server or daemon, which is running on another machine

1.1.2. Docker Daemon

The Docker daemon is what actually executes commands sent to the Docker Client—like building, running, and distributing your containers. The Docker Daemon runs on the host machine, but as a user, you never communicate directly with the Daemon. The Docker Client can run on the host machine as well, but it’s not required to. It can run on a different machine and communicate with the Docker Daemon that’s running on the host machine.

1.1.3. Docker Images

There are two methods to build an image. The first one is to build an image by using a read-only template. The foundation of every image is a base image. Operating system images are basically the base images, such as Ubuntu 14.04 LTS, or Fedora 20. The images of operating system create a container with an ability of complete running OS. Base image can also be created from the scratch. Required applications can be added to the base image by modifying it, but it is necessary to build a new image. The process of building a new image is called “committing a change”. The second method is to create a docker file. The docker file contains a list of instructions when “Docker build” command is run from the bash terminal it follows all the instructions given in the docker file and builds an image. This is an automated way of building an image.

1.1.4. Dockerfile

A Dockerfile is where you write the instructions to build a Docker image. These instructions can be:

- **RUN apt-get y install some-package**: to install a software package
- **EXPOSE 8000:** to expose a port
- **ENV ANT_HOME /usr/local/apache-ant** to pass an environment variable

and so forth. Once you’ve got your Dockerfile set up, you can use the **docker build** command to build an image from it.

1.1.5. Docker Registries

Docker images are placed in docker registries. It works correspondingly to source code repositories where images can be pushed or pulled from a single source. There are two types of registries, public and private. Docker Hub is called a public registry where everyone can pull available images and push their own images without creating an image from the scratch. Images can be distributed to a particular area (public or private) by using docker hub feature.

1.1.6. Docker Containers

Docker image creates a docker container. Containers hold the whole kit required for an application, so the application can be run in an isolated way. For example, suppose there is an image of Ubuntu OS with SQL SERVER, when this image is run with docker run command, then a container will be created and SQL SERVER will be running on Ubuntu OS.

CHAPTER 2

COMPARATIVE STUDY ON CONTAINERS AND RELATED TECHNOLOGIES

Containerisation, an Operating system (OS)-level virtualization technology, simplifies the deployment and management of applications by providing a flexible, low-overhead virtualization solution. In recent years, containers have been deployed as a replacement for Virtual Machine (VM) based virtualization. Containers offer better performance and flexibility at the expense of isolation and security when compared to traditional hypervisor virtualization. Their low overhead allows cloud infrastructure providers to deploy more micro services on a single host. Due to the recent surge in interest and deployment, containers have matured and overcome many of their initial limitations. The containers are compared on their runtime with the brief history of their origin.

FreeBSD Jails The concept of Jails was first introduced as part of FreeBSD in 2000 [1]. This concept enables the definition of separate virtual compartments on a single host to which a subset of super user privileges can be delegated. Jails were made possible by hardening the chroot [2] system call in the FreeBSD kernel.

FreeBSD issued as a standard container to restrict access outside the sandbox environment (e.g. files, processes and predefined user accounts). Later improvements to the Jails management facility introduced support for a hierarchical structure where one container relies on the functionality of another[1].Structuring containers in a hierarchy, however, is rarely used[3].Currently, FreeBSD does not have a large user base in the corporate landscape [4].

VServer As a reaction to the release of FreeBSD Jails, Linux released the Linux-V Server project in 2003 [5]. Linux-V Server uses the term Virtual Private Server (VPS) to refer to its containers. The biggest downside of Linux-V Server is the need to recompile the Linux kernel. This recompilation is required since V Server updates are not integrated into the mainline Linux kernel development branch. State-of-container reports such as the one from Cloud HQ show that Linux V Server is no longer widely used since its client-base has moved towards other Linux container solutions such as LXC .

Solaris Zones The Solaris Zones project was the first attempt to design containers able to support commercial solutions in 2004 [6]. Sun engineers built zone containers because the already existent Linux-VServer and FreeBSD Jails were not mature enough and offered limited OS integration. An important feature of Solaris Zones is the ability to delegate zone setup and configuration to administrators [7]. This delegation is done through a set of zone management tools. The zones themselves, used to restrict the abilities of a process, are formed by assigning a zone identifier to each process. Industry trends show that the usage of Solaris Zones is declining [8]

OpenVZ Another Open Source container solution for Linux, released in 2005,is Open Virtuozzo (OpenVZ)[9].The Virtual Environments(VEs)[24]created with OpenVZ are commercially offered as part of the Parallels Cloud Server solution [10] under the name Virtuozzo Containers [11]. Older versions of OpenVZ use relatively old upstream Linux kernels, but recent investments from Virtuozzo resulted in better support for more recent Linux kernels.

OpenVZ introduced the implementation of Checkpoint and restart (CR). CR allows processes to move between different physical or virtual environments, which is useful for load-balancing in high-availability deployments .

LXC aims towards using containers as a replacement for other operating system virtualization solutions using hypervisors. To this end, the LXC project provides an interface for the Linux Kernel containment functionality including tools for container and image management [12][13]. This philosophy has security implications such as overly permissive root-capability set as default. LXC has been considered production ready since the 1.0 release in February 2014 [14]. The standard Linux kernel maintains the LXC project. As a result, the project supports the use of the latest kernel version. This property is a major advantage over OpenVZ, which requires kernel patches for every new version. Because of this patching, upgrade to more recent kernel takes more time.

LXD is an extension on the LXC project driven by Canonical. It provides an additional Command Line Interface (CLI) management interface and a REST API for network based management. For those using the Open Stack cloud environment, there is also an Open Stack Nova plugin available to enable the use of containers on a cloud [15]. Furthermore, the LXD project has commercial support available.

Docker The most widely used container technology that is currently on the market is Docker Engine (referred to as Docker for the remainder of this section). Although Docker provides Linux-based Application Containers, it runs on Linux, MacOS and Windows. In MacOS and Windows, a virtualized Linux environment is employed. Docker can also run Windows-based containers on Windows 10 and Windows Server 2016 [16]. The technology relies on several features available in the kernel to provide isolated container.As an example, Docker employs five kernel namespaces: pid, net, ipc, mnt and uts. These namespaces are used to isolate processes in each container. Together with c groups, namespaces form the foundation of a container. Docker used to be built upon LXC but added more functionality. Currently, it uses lib container as the underlying container library. The most important functionality provided by Docker is the portability of the images and an ecosystem that allows developers to use other images efficiently on any system. Another important feature is the component-based approach which allows containers to be derived from other ontainers by using the same image as a basis. Although Docker is designed to run one process per container, a process can use fork() to instantiate multiple processes.

CHAPTER 3

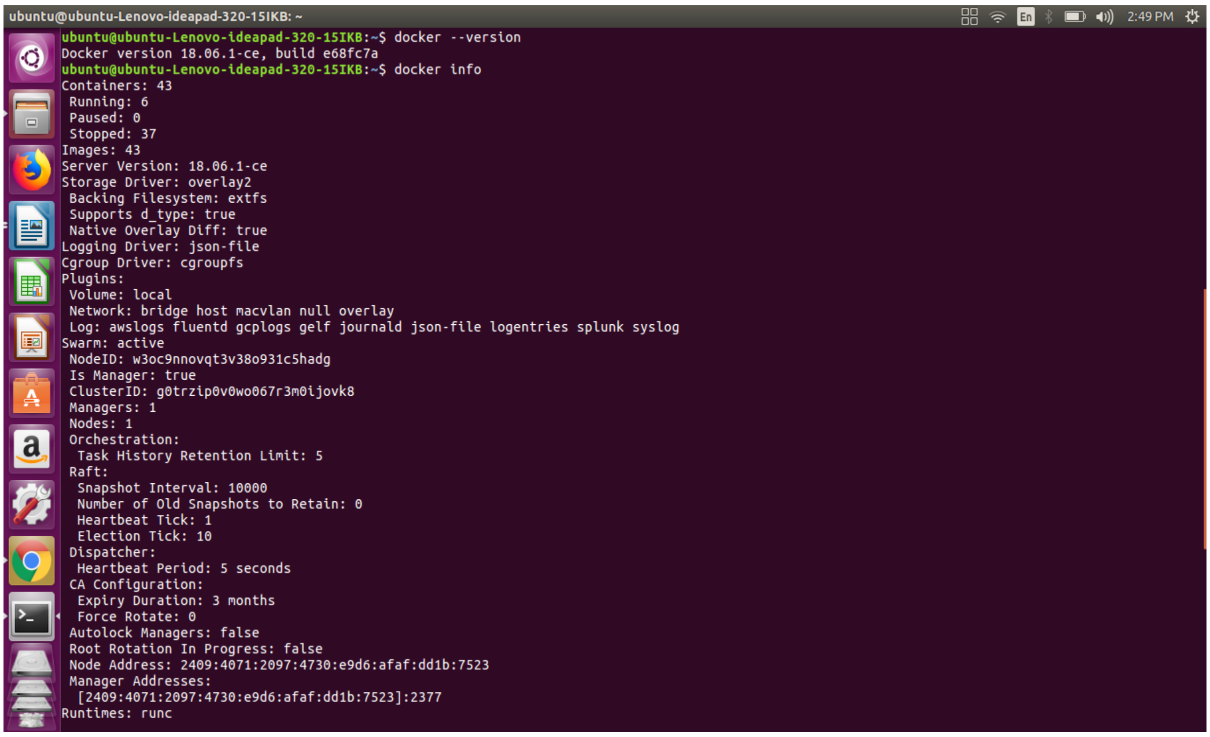
GETTING STARTED WITH DOCKER

To start working with docker the docker community version has been installed. The platform we are working is Ubuntu 16.04 version. After the installation we perform the following steps to ensure that our docker has been installed correctly.

3.1. Test docker version

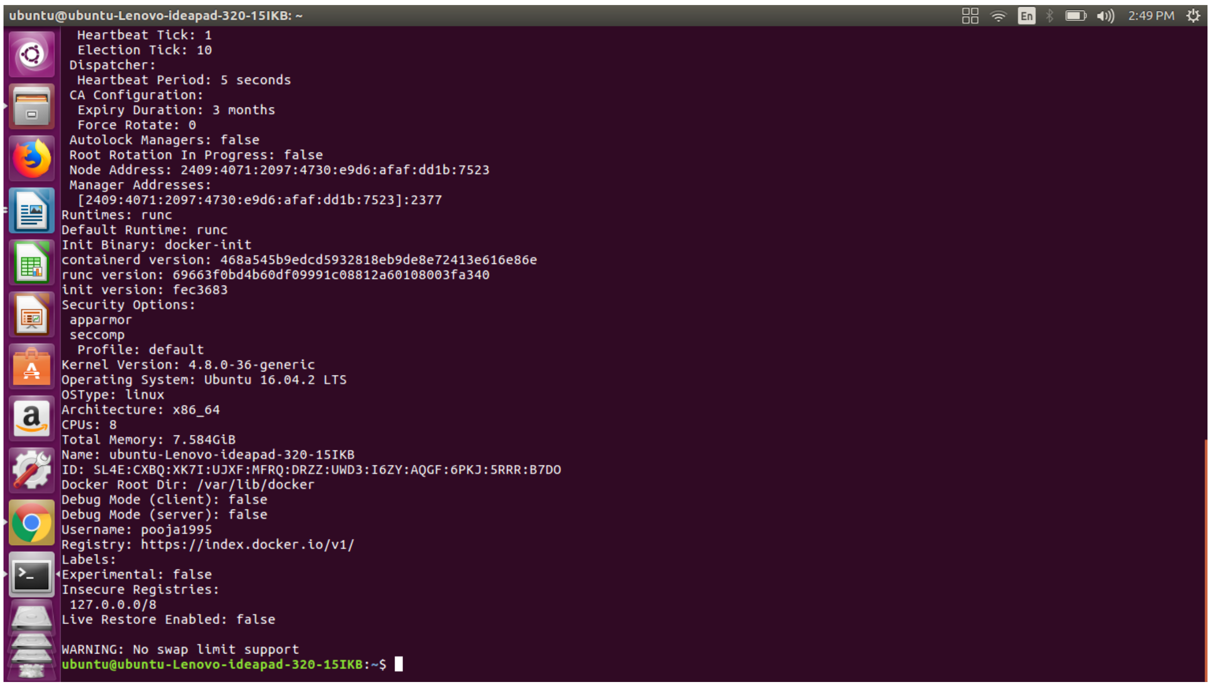
- We run *docker –version* and ensure that we have a supported version of Docker:

- Run *docker info* to view even more details about your docker installation



```
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB: ~
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ docker --version
Docker version 18.06.1-ce, build e68fc7a
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ docker info
Containers: 43
 Running: 6
 Paused: 0
 Stopped: 37
Images: 43
Server Version: 18.06.1-ce
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 logentries splunk syslog
Swarm: active
 NodeID: w3oc9nnovqt3v38o931c5hadg
 Is Manager: true
 ClusterID: g0trzip0v0wo067r3m0ijovk8
 Managers: 1
 Nodes: 1
 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
 Autolock Managers: false
 Root Rotation In Progress: false
 Node Address: 2409:4071:2097:4730:e9d6:afaf:dd1b:7523
 Manager Addresses:
  [2409:4071:2097:4730:e9d6:afaf:dd1b:7523]:2377
Runtimes: runc
```

Figure 3.1.1: Checking for docker version.

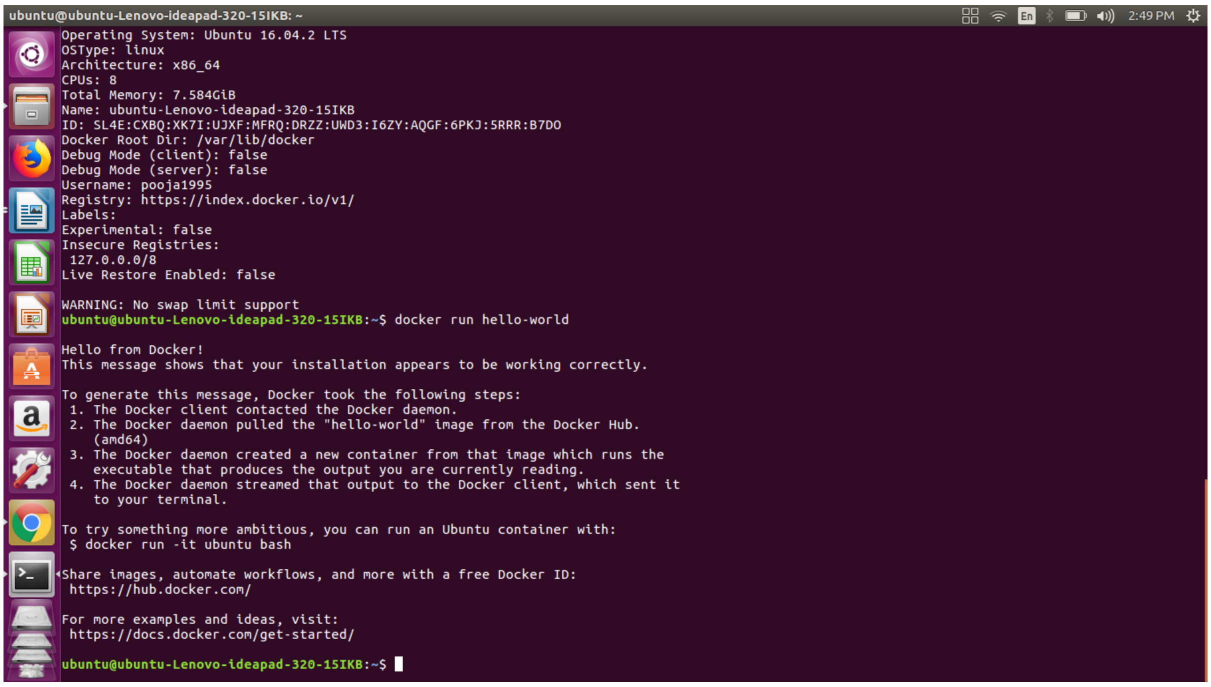


```
   Heartbeat Tick: 1
   Election Tick: 10
  Dispatcher:
   Heartbeat Period: 5 seconds
  CA Configuration:
   Expiry Duration: 3 months
   Force Rotate: 0
  Autolock Managers: false
  Root Rotation In Progress: false
  Node Address: 2409:4071:2097:4730:e9d6:afaf:dd1b:7523
  Manager Addresses:
   [2409:4071:2097:4730:e9d6:afaf:dd1b:7523]:2377
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 468a545b9edcd5932818eb9de8e72413e616e86e
runc version: 69663f0bd4b60df09991c08812a60108003fa340
init version: fec3683
Security Options:
 apparmor
 seccomp
  Profile: default
Kernel Version: 4.8.0-36-generic
Operating System: Ubuntu 16.04.2 LTS
OSType: linux
Architecture: x86_64
CPUs: 8
Total Memory: 7.584GiB
Name: ubuntu-Lenovo-ideapad-320-15IKB
ID: SL4E:CXBQ:XK7I:UJXF:MFRQ:DRZZ:UWD3:I6ZY:AQGF:6PKJ:5RRR:B7DO
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Username: pooja1995
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false

WARNING: No swap limit support
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$
```

Figure 3.1.2: Docker info

3.2. Test Docker installation

Test that your installation works by running the simple Docker image, hello-world

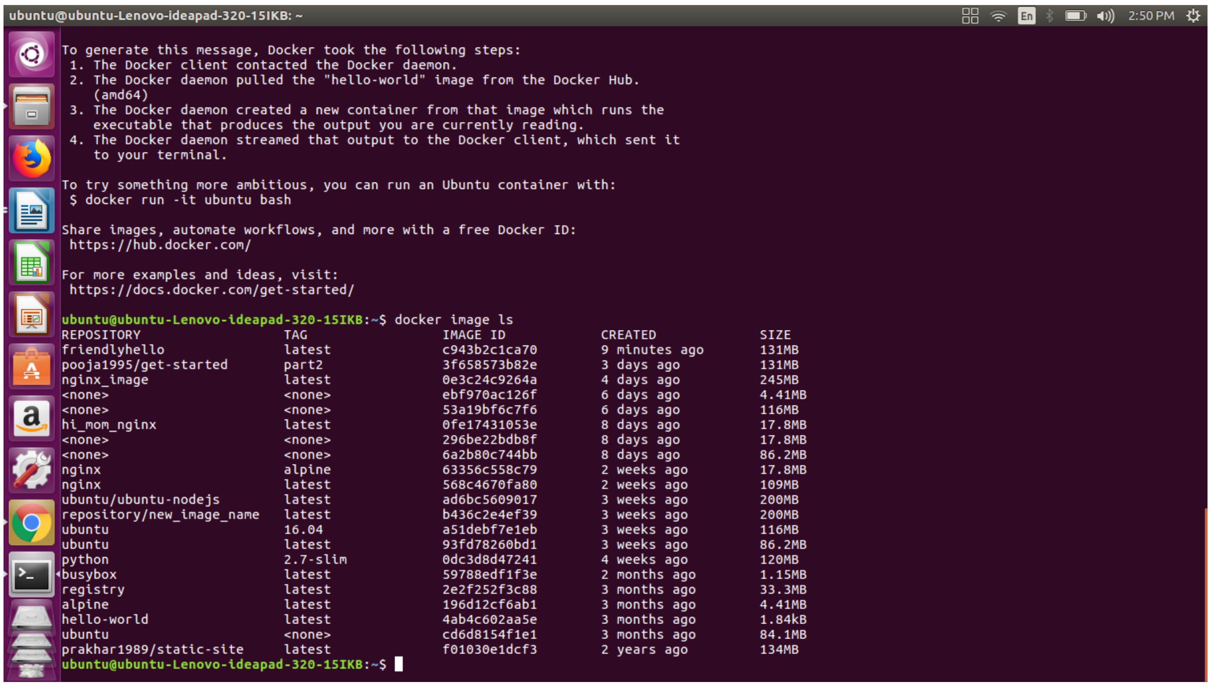


```
Operating System: Ubuntu 16.04.2 LTS
OSType: linux
Architecture: x86_64
CPUs: 8
Total Memory: 7.584GiB
Name: ubuntu-Lenovo-ideapad-320-15IKB
ID: SL4E:CXBQ:XK7I:UJXF:MFRQ:DRZZ:UWD3:I6ZY:AQGF:6PKJ:5RRR:B7DO
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Username: pooja1995
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false

WARNING: No swap limit support
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$
```

Figure 3.2.1: checking for successful installation

Listing the hello-world image that was downloaded to the machine

```
docker image ls
```

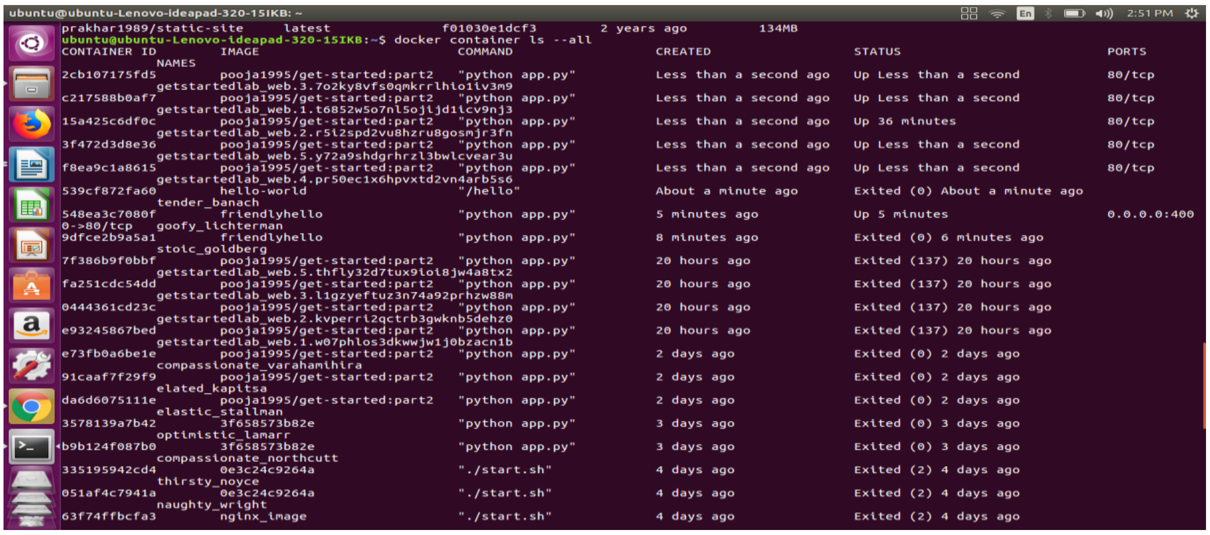


```
To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ docker image ls
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
friendlyhello               latest              c943b2c1ca70        9 minutes ago       131MB
pooja1995/get-started       part2               3f658573b82e        3 days ago          131MB
nginx_image                 latest              0e3c24c9264a        4 days ago          245MB
<none>                      <none>              ebf970ac126f        6 days ago          4.41MB
<none>                      <none>              53a19bf6c7f6        6 days ago          116MB
hi_mom_nginx                latest              0fe17431053e        8 days ago          17.8MB
<none>                      <none>              296be22bdb8f        8 days ago          17.8MB
<none>                      <none>              6a2b80c744bb        8 days ago          86.2MB
nginx                       alpine              63356c558c79        2 weeks ago         17.8MB
nginx                       latest              568c4670fa80        2 weeks ago         109MB
ubuntu/ubuntu-nodejs        latest              ad6bc5609017        3 weeks ago         200MB
repository/new_image_name   latest              b436c2e4ef39        3 weeks ago         200MB
ubuntu                      16.04               a51debf7e1eb        3 weeks ago         116MB
ubuntu                      latest              93fd78260bd1        3 weeks ago         86.2MB
python                      2.7-slim            0dc3d8d47241        4 weeks ago         120MB
busybox                     latest              59788edf1f3e        2 months ago        1.15MB
registry                    latest              2e2f252f3c88        3 months ago        33.3MB
alpine                      latest              196d12cf6ab1        3 months ago        4.41MB
hello-world                 latest              4ab4c602aa5e        3 months ago        1.84kB
ubuntu                      <none>              cd6d8154f1e1        3 months ago        84.1MB
prakhar1989/static-site     latest              f01030e1dcf3        2 years ago         134MB
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$
```

Figure 3.2.2: listing the downloaded images.

Listing the hello-world container (spawned by the image) which exits after displaying its message. If it were still running, we would not need the –all option

```
docker container ls –all
```



```
prakhar1989/static-site     latest              f01030e1dcf3        2 years ago         134MB
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ docker container ls --all
CONTAINER ID        IMAGE                          COMMAND                  CREATED                  STATUS                            PORTS
            NAMES
2cb107175fd5        pooja1995/get-started:part2    "python app.py"          Less than a second ago   Up Less than a second             80/tcp
            getstartedlab_web.3.7o2ky8vfs0qmkrrlhio1iv3m9
c217588b0af7        pooja1995/get-started:part2    "python app.py"          Less than a second ago   Up Less than a second             80/tcp
            getstartedlab_web.1.t6852w5o7nl5ojijd1icv9nj3
15a425c6df0c        pooja1995/get-started:part2    "python app.py"          Less than a second ago   Up 36 minutes                     80/tcp
            getstartedlab_web.2.r5i2spd2vu8hzru8gosmjr3fn
3f472d3d8e36        pooja1995/get-started:part2    "python app.py"          Less than a second ago   Up Less than a second             80/tcp
            getstartedlab_web.5.y72a9shdgrhrzl3bwlcvear3u
f8ea9c1a8615        pooja1995/get-started:part2    "python app.py"          Less than a second ago   Up Less than a second             80/tcp
            getstartedlab_web.4.pr50ec1x6hpvxtd2vn4arb5s6
539cf872fa60        hello-world                    "/hello"                 About a minute ago       Exited (0) About a minute ago
            tender_banach
548ea3c7080f        friendlyhello                  "python app.py"          5 minutes ago            Up 5 minutes                      0.0.0.0:400
0->80/tcp   goofy_lichterman
9dfce2b9a5a1        friendlyhello                  "python app.py"          8 minutes ago            Exited (0) 6 minutes ago
            stoic_goldberg
7f386b9f0bbf        pooja1995/get-started:part2    "python app.py"          20 hours ago             Exited (137) 20 hours ago
            getstartedlab_web.5.thfly32d7tux9ioi8jw4a8tx2
fa251cdc54dd        pooja1995/get-started:part2    "python app.py"          20 hours ago             Exited (137) 20 hours ago
            getstartedlab_web.3.l1gzyeftuz3n74a92prhzw88m
0444361cd23c        pooja1995/get-started:part2    "python app.py"          20 hours ago             Exited (137) 20 hours ago
            getstartedlab_web.2.kvperri2qctrb3gwknb5dehz0
e93245867bed        pooja1995/get-started:part2    "python app.py"          20 hours ago             Exited (137) 20 hours ago
            getstartedlab_web.1.w07phlos3dkwwjw1j0bzacn1b
e73fb0a6be1e        pooja1995/get-started:part2    "python app.py"          2 days ago               Exited (0) 2 days ago
            compassionate_varahamihira
91caaf7f29f9        pooja1995/get-started:part2    "python app.py"          2 days ago               Exited (0) 2 days ago
            elated_kapitsa
da6d6075111e        pooja1995/get-started:part2    "python app.py"          2 days ago               Exited (0) 2 days ago
            elastic_stallman
3578139a7b42        3f658573b82e                   "python app.py"          3 days ago               Exited (0) 3 days ago
            optimistic_lamarr
b9b124f087b0        3f658573b82e                   "python app.py"          3 days ago               Exited (0) 3 days ago
            compassionate_northcutt
335195942cd4        0e3c24c9264a                   "./start.sh"             4 days ago               Exited (2) 4 days ago
            thirsty_noyce
051af4c7941a        0e3c24c9264a                   "./start.sh"             4 days ago               Exited (2) 4 days ago
            naughty_wright
63f74ffbcfa3        nginx_image                    "./start.sh"             4 days ago               Exited (2) 4 days ago
```

Figure 3.2.3: listing containers

3.3. Dockerfile

In the past, if we were to start writing a Python app, our first order of business was to install a Python runtime onto our machine. But, that creates a situation where the environment on our machine needs to be perfect for our app to run as expected, and also needs to match our production environment.

With Docker, we can just grab a portable Python runtime as an image, no installation necessary. Then, our build can include the base Python image right alongside our app code, ensuring that our app, its dependencies, and the runtime, all travel together. These portable images are defined by something called a Dockerfile.

Dockerfile defines what goes on in the environment inside our container. Access to resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of our system, so we need to map ports to the outside world, and be specific about what files we want to “copy in” to that environment. However, after doing that, we can expect that the build of our app defined in this Dockerfile behaves exactly the same wherever it runs.

Creating a Dockerfile

- We create an empty directory.
- Change directories (cd) into the new directory
- Create a file called Dockerfile
- Copy-and-paste the following content into that file
- Save it.

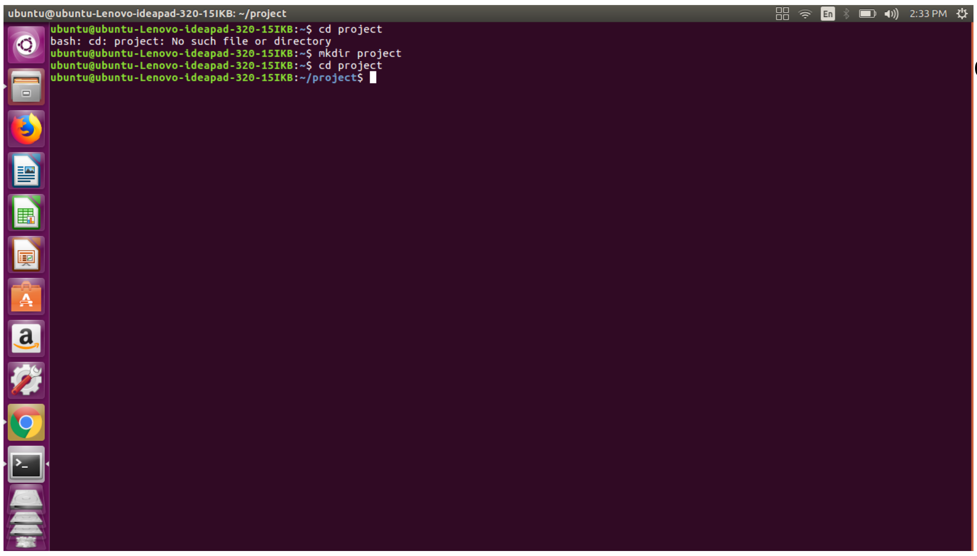

```
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB: ~/project
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ cd project
bash: cd: project: No such file or directory
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ mkdir project
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ cd project
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ 
```

Figure 3.3.1: creating a directory

```
•   # Use an official Python runtime as a parent image
•   FROM python:2.7-slim
•
•   # Set the working directory to /app
•   WORKDIR /app
•
•   # Copy the current directory contents into the container at /app
•   COPY . /app
•
•   # Install any needed packages specified in requirements.txt
•   RUN pip install –trusted-host pypi.python.org –r requirements.txt
•
•   # Make port 80 available to the world outside this container
•   EXPOSE 80
•
•   # Define environment variable
•   ENV NAME World
•
•   # Run app.py when the container launches
•   CMD [“python”, “app.py”]
```

This Dockerfile refers to a couple of files we haven’t created yet, namely app.py and requirements.txt. we will Create two more files, requirements.txt and app.py, and put them in the same folder with the Dockerfile. This completes our app. When the above Dockerfile is built into an image, app.py and requirements.txt is present because of that Dockerfile’s COPY command, and the output from app.py is accessible over HTTP thanks to the EXPOSE command.

Requirements.txt

```
Flask

Redis
```

app.py

```
from flask import Flask

from redis import Redis, RedisError

import os

import socket


# Connect to Redis

redis = Redis(host=”redis”, db=0, socket_connect_timeout=2, socket_timeout=2)


app = Flask(__name__)
```

```
@app.route(“/”)

def hello():

    try:

        visits = redis.incr(“counter”)

    except RedisError:

        visits = “<i>cannot connect to Redis, counter disabled</i>”


    html = “<h3>Hello {name}!</h3>” \

           “<b>Hostname:</b> {hostname}<br/>” \

           “<b>Visits:</b> {visits}”

    return html.format(name=os.getenv(“NAME”, “world”),
hostname=socket.gethostname(), visits=visits)


if __name__ == “__main__”:

    app.run(host=’0.0.0.0’, port=80)
```

Build the app

We are ready to build the app. Here’s what ls should show:

```
$ ls

Dockerfile               app.py                   requirements.txt
```

Now we will run the build command. This creates a Docker image, which we’re going to tag using –t so it has a friendly name.

```
docker build –t friendlyhello .
```

The execution of the build command is shown below.

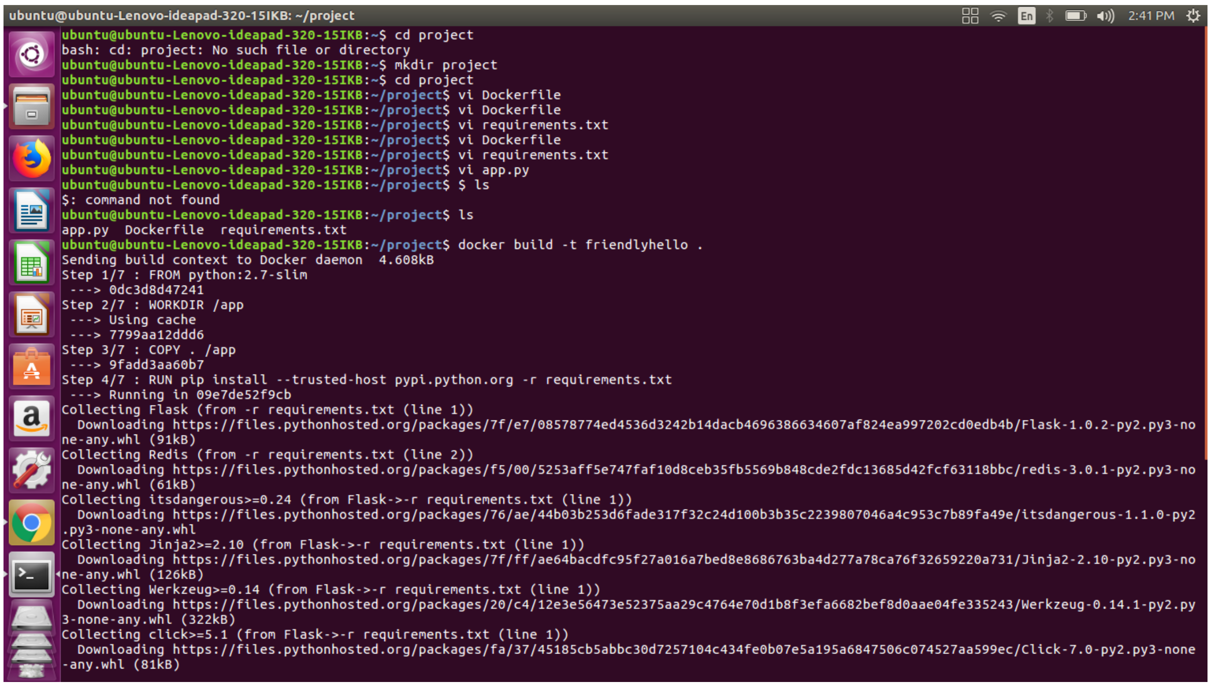

```
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB: ~/project
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ cd project
bash: cd: project: No such file or directory
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ mkdir project
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~$ cd project
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ vi Dockerfile
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ vi Dockerfile
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ vi requirements.txt
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ vi Dockerfile
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ vi requirements.txt
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ vi app.py
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ $ ls
$: command not found
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ ls
app.py  Dockerfile  requirements.txt
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ docker build -t friendlyhello .
Sending build context to Docker daemon  4.608kB
Step 1/7 : FROM python:2.7-slim
 ---> 0dc3d8d47241
Step 2/7 : WORKDIR /app
 ---> Using cache
 ---> 7799aa12ddd6
Step 3/7 : COPY . /app
 ---> 9fadd3aa60b7
Step 4/7 : RUN pip install --trusted-host pypi.python.org -r requirements.txt
 ---> Running in 09e7de52f9cb
Collecting Flask (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7f/e7/08578774ed4536d3242b14dacb4696386634607af824ea997202cd0edb4b/Flask-1.0.2-py2.py3-no
ne-any.whl (91kB)
Collecting Redis (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/f5/00/5253aff5e747faf10d8ceb35fb5569b848cde2fdc13685d42fcf63118bbc/redis-3.0.1-py2.py3-no
ne-any.whl (61kB)
Collecting itsdangerous>=0.24 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2
.py3-none-any.whl
Collecting Jinja2>=2.10 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7f/ff/ae64bacdfc95f27a016a7bed8e8686763ba4d277a78ca76f32659220a731/Jinja2-2.10-py2.py3-no
ne-any.whl (126kB)
Collecting Werkzeug>=0.14 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/20/c4/12e3e56473e52375aa29c4764e70d1b8f3efa6682bef8d0aae04fe335243/Werkzeug-0.14.1-py2.py
3-none-any.whl (322kB)
Collecting click>=5.1 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/fa/37/45185cb5abbc30d7257104c434fe0b07e5a195a6847506c074527aa599ec/Click-7.0-py2.py3-none
-any.whl (81kB)
```

Figure 3.3.2: creating docker image

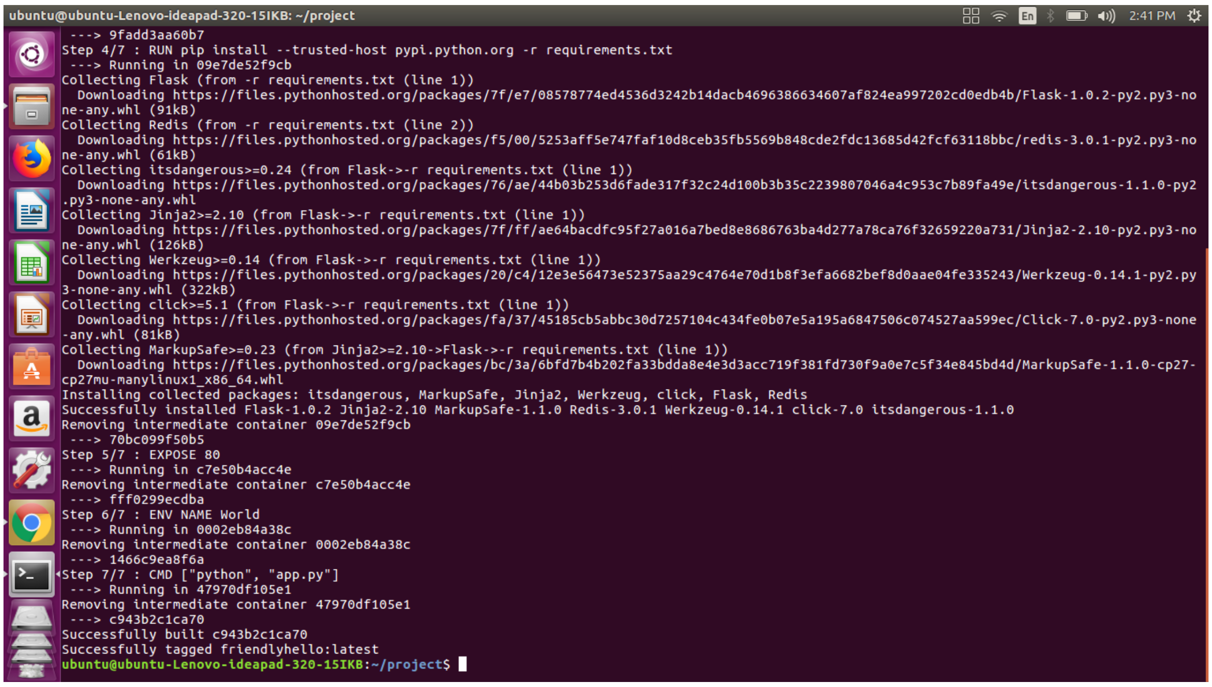
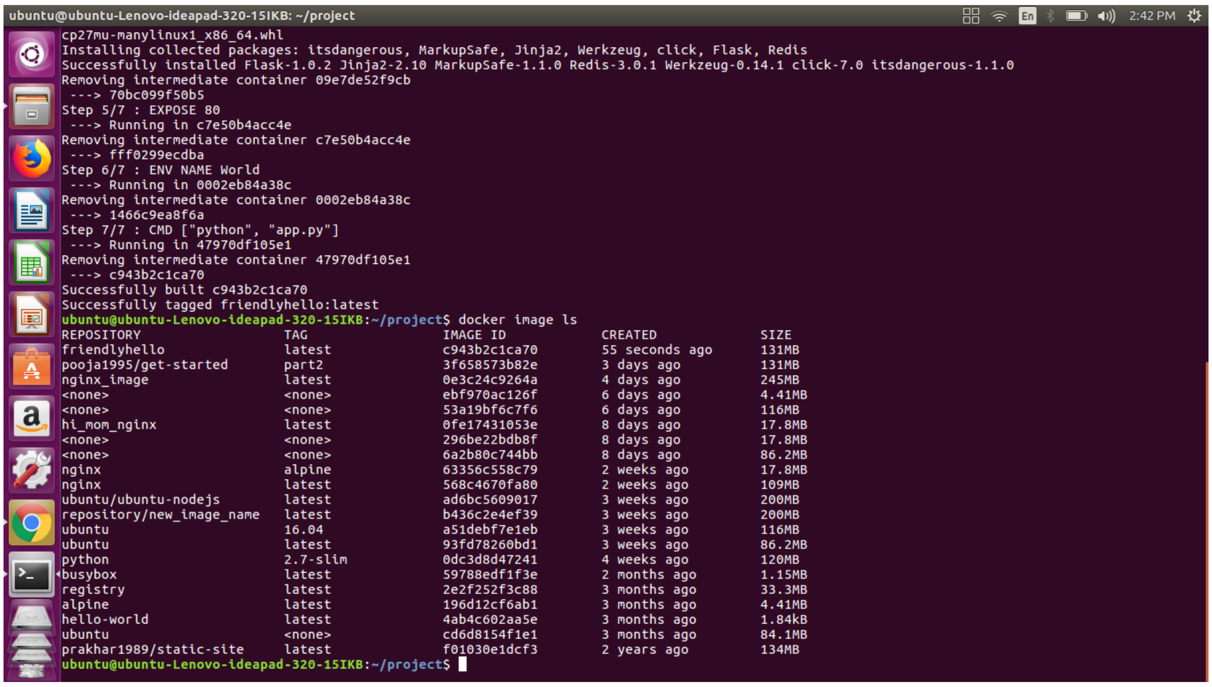

```
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB: ~/project
 ---> 9fadd3aa60b7
Step 4/7 : RUN pip install --trusted-host pypi.python.org -r requirements.txt
 ---> Running in 09e7de52f9cb
Collecting Flask (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7f/e7/08578774ed4536d3242b14dacb4696386634607af824ea997202cd0edb4b/Flask-1.0.2-py2.py3-no
ne-any.whl (91kB)
Collecting Redis (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/f5/00/5253aff5e747faf10d8ceb35fb5569b848cde2fdc13685d42fcf63118bbc/redis-3.0.1-py2.py3-no
ne-any.whl (61kB)
Collecting itsdangerous>=0.24 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/76/ae/44b03b253d6fade317f32c24d100b3b35c2239807046a4c953c7b89fa49e/itsdangerous-1.1.0-py2
.py3-none-any.whl
Collecting Jinja2>=2.10 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7f/ff/ae64bacdfc95f27a016a7bed8e8686763ba4d277a78ca76f32659220a731/Jinja2-2.10-py2.py3-no
ne-any.whl (126kB)
Collecting Werkzeug>=0.14 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/20/c4/12e3e56473e52375aa29c4764e70d1b8f3efa6682bef8d0aae04fe335243/Werkzeug-0.14.1-py2.py
3-none-any.whl (322kB)
Collecting click>=5.1 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/fa/37/45185cb5abbc30d7257104c434fe0b07e5a195a6847506c074527aa599ec/Click-7.0-py2.py3-none
-any.whl (81kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.10->Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/bc/3a/6bfd7b4b202fa33bdda8e4e3d3acc719f381fd730f9a0e7c5f34e845bd4d/MarkupSafe-1.1.0-cp27-
cp27mu-manylinux1_x86_64.whl
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, Werkzeug, click, Flask, Redis
Successfully installed Flask-1.0.2 Jinja2-2.10 MarkupSafe-1.1.0 Redis-3.0.1 Werkzeug-0.14.1 click-7.0 itsdangerous-1.1.0
Removing intermediate container 09e7de52f9cb
 ---> 70bc099f50b5
Step 5/7 : EXPOSE 80
 ---> Running in c7e50b4acc4e
Removing intermediate container c7e50b4acc4e
 ---> fff0299ecdba
Step 6/7 : ENV NAME World
 ---> Running in 0002eb84a38c
Removing intermediate container 0002eb84a38c
 ---> 1466c9ea8f6a
Step 7/7 : CMD ["python", "app.py"]
 ---> Running in 47970df105e1
Removing intermediate container 47970df105e1
 ---> c943b2c1ca70
Successfully built c943b2c1ca70
Successfully tagged friendlyhello:latest
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ 
```

Figure 3.3.3: creating docker image

Where is our built image? It’s in your machine’s local Docker image registry:Run the app

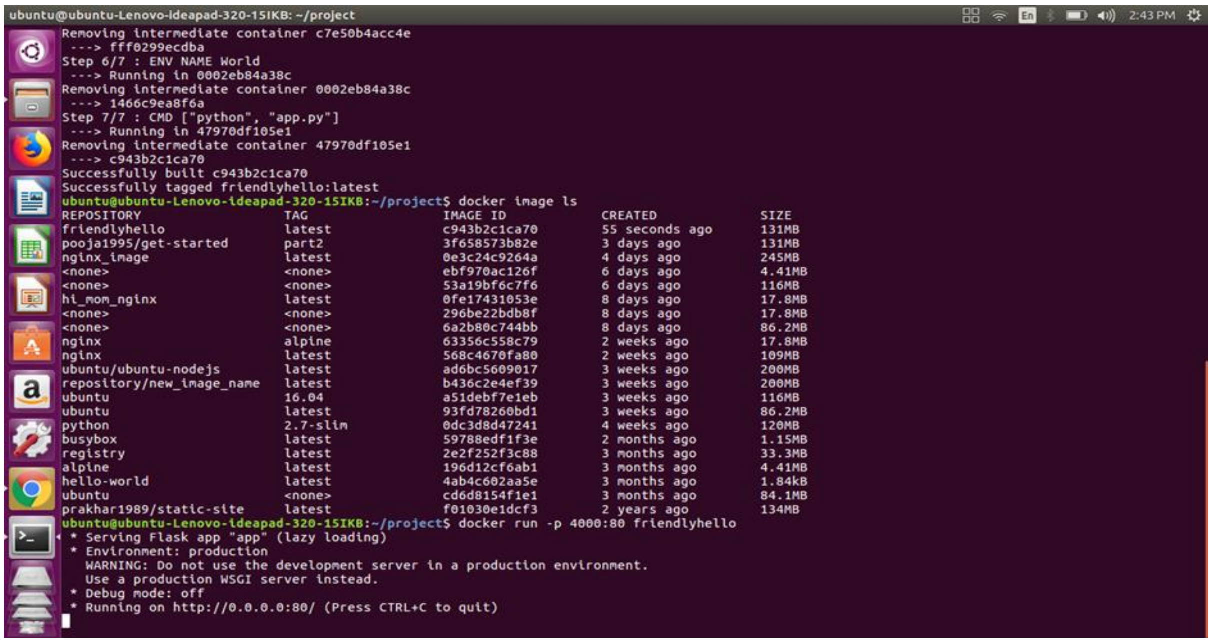


```
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB: ~/project
cp27mu-manylinux1_x86_64.whl
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, Werkzeug, click, Flask, Redis
Successfully installed Flask-1.0.2 Jinja2-2.10 MarkupSafe-1.1.0 Redis-3.0.1 Werkzeug-0.14.1 click-7.0 itsdangerous-1.1.0
Removing intermediate container 09e7de52f9cb
 ---> 70bc099f50b5
Step 5/7 : EXPOSE 80
 ---> Running in c7e50b4acc4e
Removing intermediate container c7e50b4acc4e
 ---> fff0299ecdba
Step 6/7 : ENV NAME World
 ---> Running in 0002eb84a38c
Removing intermediate container 0002eb84a38c
 ---> 1466c9ea8f6a
Step 7/7 : CMD ["python", "app.py"]
 ---> Running in 47970df105e1
Removing intermediate container 47970df105e1
 ---> c943b2c1ca70
Successfully built c943b2c1ca70
Successfully tagged friendlyhello:latest
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ docker image ls
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
friendlyhello               latest              c943b2c1ca70        55 seconds ago      131MB
pooja1995/get-started       part2               3f658573b82e        3 days ago          131MB
nginx_image                 latest              0e3c24c9264a        4 days ago          245MB
<none>                      <none>              ebf970ac126f        6 days ago          4.41MB
<none>                      <none>              53a19bf6c7f6        6 days ago          116MB
hi_mom_nginx                latest              0fe17431053e        8 days ago          17.8MB
<none>                      <none>              296be22bdb8f        8 days ago          17.8MB
<none>                      <none>              6a2b80c744bb        8 days ago          86.2MB
nginx                       alpine              63356c558c79        2 weeks ago         17.8MB
nginx                       latest              568c4670fa80        2 weeks ago         109MB
ubuntu/ubuntu-nodejs        latest              ad6bc5609017        3 weeks ago         200MB
repository/new_image_name   latest              b436c2e4ef39        3 weeks ago         200MB
ubuntu                      16.04               a51debf7e1eb        3 weeks ago         116MB
ubuntu                      latest              93fd78260bd1        3 weeks ago         86.2MB
python                      2.7-slim            0dc3d8d47241        4 weeks ago         120MB
busybox                     latest              59788edf1f3e        2 months ago        1.15MB
registry                    latest              2e2f252f3c88        3 months ago        33.3MB
alpine                      latest              196d12cf6ab1        3 months ago        4.41MB
hello-world                 latest              4ab4c602aa5e        3 months ago        1.84kB
ubuntu                      <none>              cd6d8154f1e1        3 months ago        84.1MB
prakhar1989/static-site     latest              f01030e1dcf3        2 years ago         134MB
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ 
```

Figure 3.3.4: list of docker images

Running the app, mapping our machine’s port 4000 to the container’s published port 80 using –p:

```
docker run -p 4000:80 friendlyhello
```

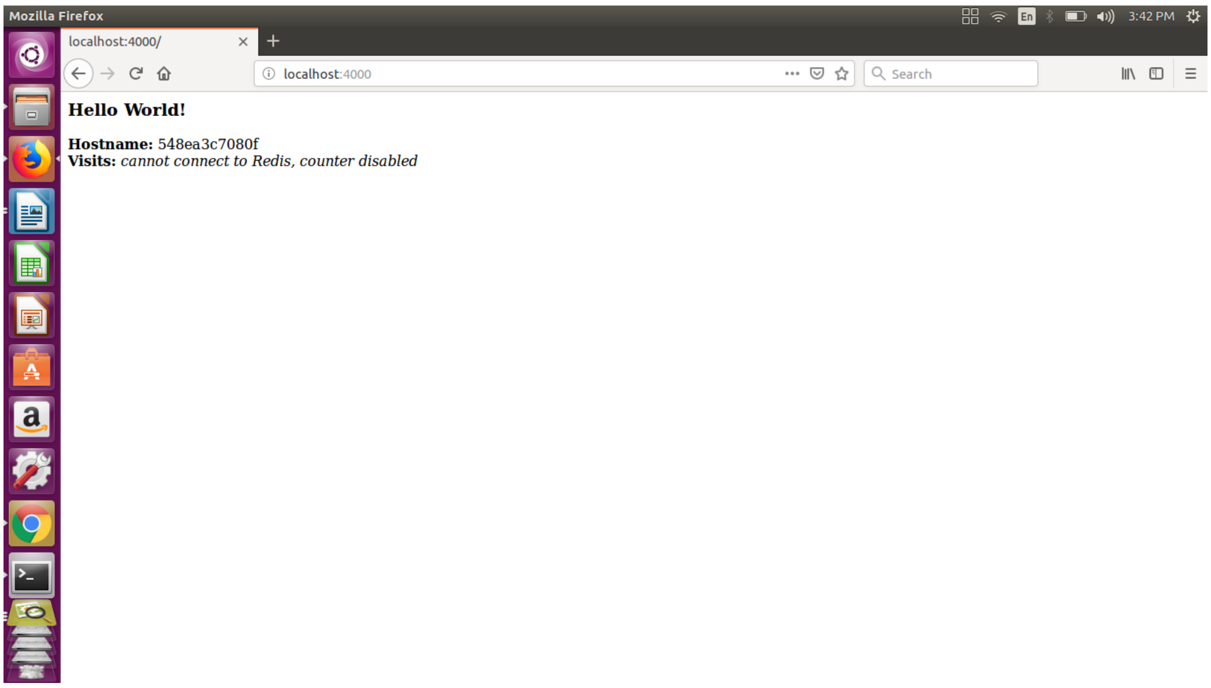


```
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB: ~/project
Removing intermediate container c7e50b4acc4e
 ---> fff0299ecdba
Step 6/7 : ENV NAME World
 ---> Running in 0002eb84a38c
Removing intermediate container 0002eb84a38c
 ---> 1466c9ea8f6a
Step 7/7 : CMD ["python", "app.py"]
 ---> Running in 47970df105e1
Removing intermediate container 47970df105e1
 ---> c943b2c1ca70
Successfully built c943b2c1ca70
Successfully tagged friendlyhello:latest
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ docker image ls
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
friendlyhello               latest              c943b2c1ca70        55 seconds ago      131MB
pooja1995/get-started       part2               3f658573b82e        3 days ago          131MB
nginx_image                 latest              0e3c24c9264a        4 days ago          245MB
<none>                      <none>              ebf970ac126f        6 days ago          4.41MB
<none>                      <none>              53a19bf6c7f6        6 days ago          116MB
hi_mom_nginx                latest              0fe17431053e        8 days ago          17.8MB
<none>                      <none>              296be22bdb8f        8 days ago          17.8MB
<none>                      <none>              6a2b80c744bb        8 days ago          86.2MB
nginx                       alpine              63356c558c79        2 weeks ago         17.8MB
nginx                       latest              568c4670fa80        2 weeks ago         109MB
ubuntu/ubuntu-nodejs        latest              ad6bc5609017        3 weeks ago         200MB
repository/new_image_name   latest              b436c2e4ef39        3 weeks ago         200MB
ubuntu                      16.04               a51debf7e1eb        3 weeks ago         116MB
ubuntu                      latest              93fd78260bd1        3 weeks ago         86.2MB
python                      2.7-slim            0dc3d8d47241        4 weeks ago         120MB
busybox                     latest              59788edf1f3e        2 months ago        1.15MB
registry                    latest              2e2f252f3c88        3 months ago        33.3MB
alpine                      latest              196d12cf6ab1        3 months ago        4.41MB
hello-world                 latest              4ab4c602aa5e        3 months ago        1.84kB
ubuntu                      <none>              cd6d8154f1e1        3 months ago        84.1MB
prakhar1989/static-site     latest              f01030e1dcf3        2 years ago         134MB
ubuntu@ubuntu-Lenovo-ideapad-320-15IKB:~/project$ docker run -p 4000:80 friendlyhello
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
```

Figure 3.3.5: running the image

We will get to see a message that Python is serving our app at http://0.0.0.0:80. But that message is coming from inside the container, which doesn't know we have mapped port 80 of that container to 4000, making the correct URL http://localhost:4000.

We will Go to that URL in a web browser to see the display content served up on a web page.


Mozilla Firefox

localhost:4000/

localhost:4000

Hello World!

Hostname: 548ea3c7080f
Visits: *cannot connect to Redis, counter disabled*

Figure 3.3.6: Display content served up on a web page.

We can also use the curl command in a shell to view the same content.

```
$ curl http://localhost:4000


<h3>Hello World!</h3><b>Hostname:</b> 8fc990912a14<br/><b>Visits:</b> <i>cannot
connect to Redis, counter disabled</i>
```

Now we use docker container stop to end the process, using the CONTAINER ID, like so:

```
docker container stop 1fa4ab2cf395
```

3.3.1. Sharing the image

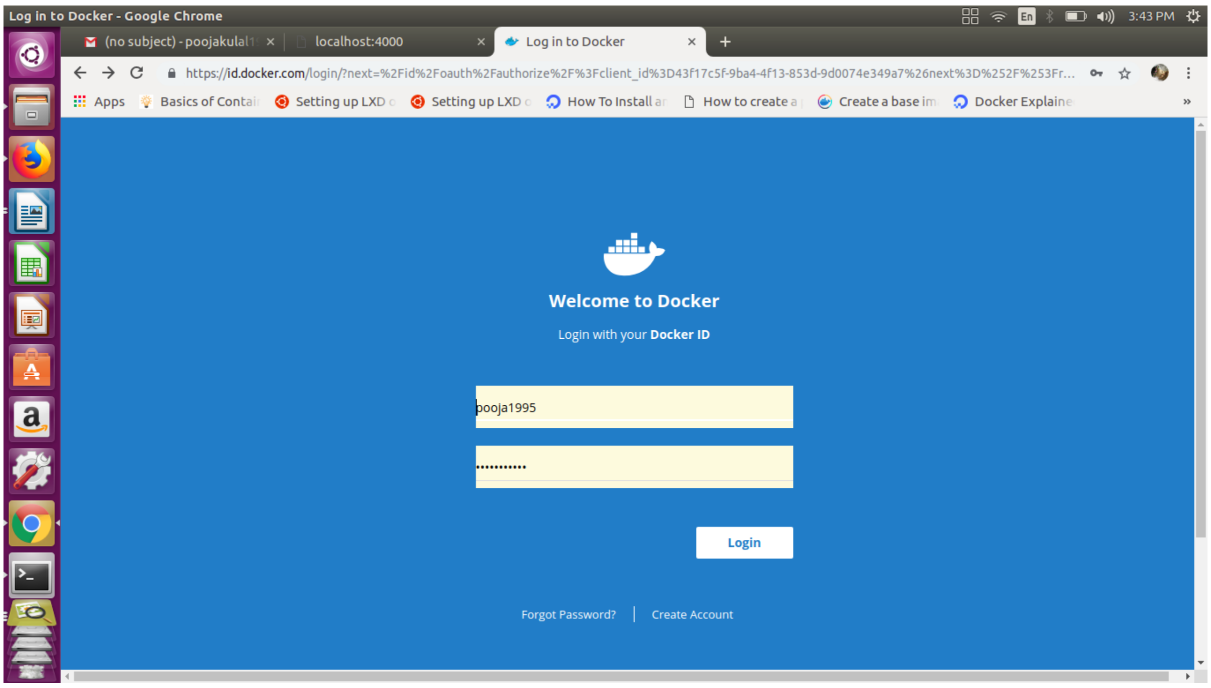
To demonstrate the portability of what we just created, let’s upload our built image and run it somewhere else. After all, we need to know how to push to registries when we want to deploy containers to production.

A registry is a collection of repositories, and a repository is a collection of images sort of like a GitHub repository, except the code is already built. An account on a registry can create many repositories. The docker CLI uses Docker’s public registry by default.

We will log in with our Docker ID

We will log in to the Docker public registry on our local machine.

```
$ docker login
```



https://id.docker.com/login/?next=%2Fid%2Foauth%2Fauthorize%2F%3Fclient_id%3D43f17c5f-9ba4-4f13-853d-9d0074e349a7%26next%3D%252F%253Fr...

Welcome to Docker

Login with your **Docker ID**

pooja1995

••••••••••

Figure 3.3.1.1: docker registry login page

3.3.2. Tag the image

The notation for associating a local image with a repository on a registry is username/repository:tag. The tag is optional, but recommended, since it is the mechanism that registries use to give Docker images a version. We will give the repository and tag meaningful names for the context, such asget-started:part2. This puts the image in the get-started repository and tags it as part2.

Now, we put it all together to tag the image. Run docker tag image with our username, repository, and tag names so that the image uploads to our desired destination. The syntax of the command is:

```
docker tag image username/repository:tag
```

For example:

```
docker tag friendlyhello pooja1995/get-started:part2
```

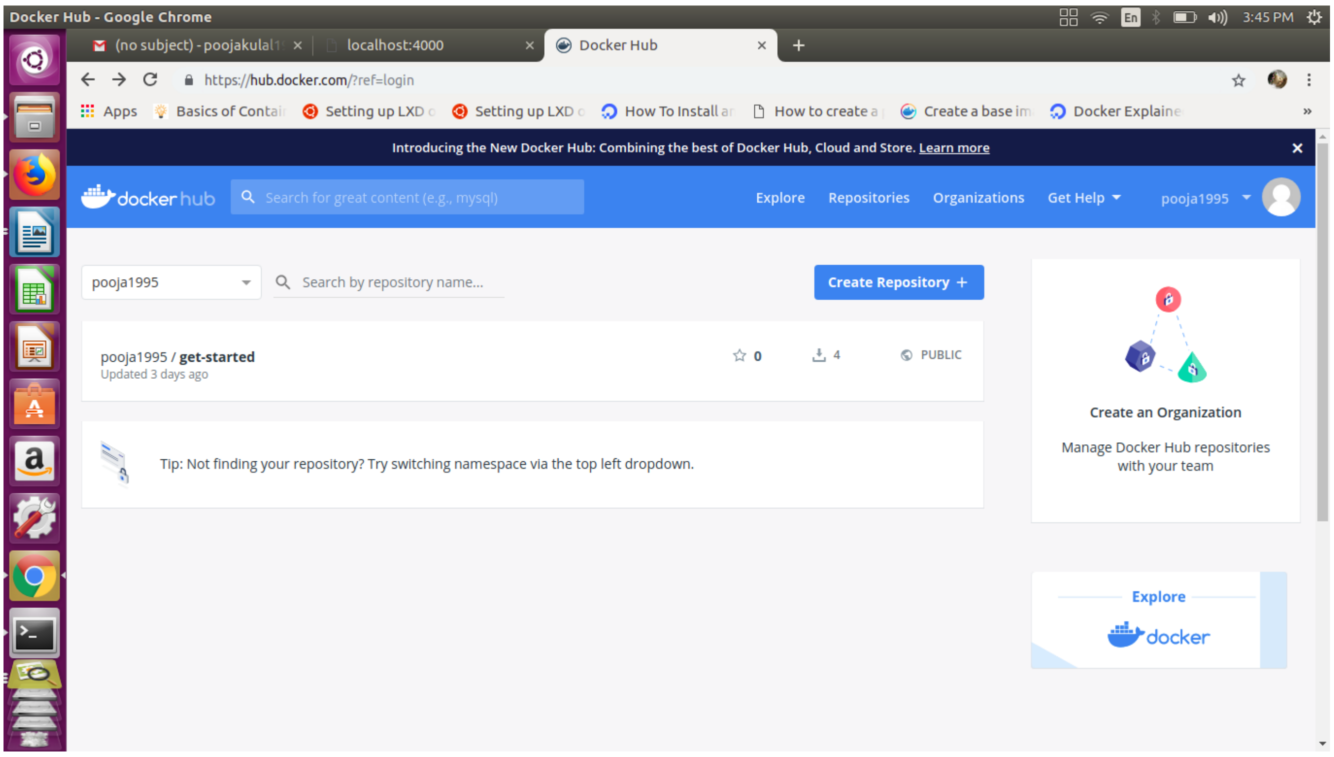
Here pooja1995 will the id used to login to the docker hub and get-strarted:part2 will be the repository name we are tagging to.

3.3.3. Publish the image

Upload your tagged image to the repository:

```
docker push username/repository: tag
```

Once complete, the results of this upload are publicly available. If we log in to Docker Hub, we see the new image there, with its pull command as shown in below image.



Introducing the New Docker Hub: Combining the best of Docker Hub, Cloud and Store. Learn more

pooja1995 / **get-started** 0 4 PUBLIC
Updated 3 days ago

Tip: Not finding your repository? Try switching namespace via the top left dropdown.

Manage Docker Hub repositories with your team

Figure 3.3.3.1: docker hub repository

CHAPTER 4

CONCLUSION

A major benefit of containers is their portability. A container wraps up an application with everything it needs to run, like configuration files and dependencies. Since containers do not require a separate operating system, they use up less resources. While a VM often measures several gigabytes in size, a container usually measures only a few dozen megabytes, making it possible to run many more containers than VMs on a single server. Since containers have a higher utilisation level with regard to the underlying hardware, we require less hardware, resulting in a reduction of bare metal costs as well as datacentre costs. Although containers run on the same server and use the same resources, they do not interact with each other. If one application crashes, other containers with the same application will keep running flawlessly and won’t experience any technical problems. This isolation also decreases security risks: If one application should be hacked or breached by malware, any resulting negative effects won’t spread to the other running containers. As mentioned before, containers are lightweight and start in less than a second since they do not require an operating system boot. Creating, replicating or destroying containers is also just a matter of seconds, thus greatly speeding up the development process, the time to market and the operational speed. Releasing new software or versions has never been so easy and quick. But the increased speed also offers great opportunities for improving customer experience, since it enables organisations and developers to act quickly, for example when it comes to fixing bugs or adding new features.

Docker is by far the most popular containerisation platform, but to successfully adopt containers we will also need to implement a container orchestration system. Kubernetes, based on over 10 years of experience in running containerised workloads at Google, is the clear market leader in container orchestration. In the next work of the project kubernetes will be used and implemented

REFERENCES

[1] PH. Kamp, R. Watson. Jails: Confining the omnipotent root. In Proceedings of the 2nd International SANE Conference, volume 43, page 116, 2000.

[2] A. Gholami, E. Laure. Security and privacy of sensitive data in cloud computing: a survey of recent developments. ArXiv preprint arXiv:1601.01498, 2016.

[3] E. Reshetova, J. Karhunen, T. Nyman, N Asokan. Security of oslevel virtualization technologies. In Nordic Conference on Secure IT Systems, pages 77–93. Springer, 2014.

[4] DevOps.com, ClusterHQ. State-of-container-usagejune-2016.pdf.https://clusterhq.com/assets/pdfs/state-of-container-usage-june-2016.pdf, 2016. (Accessed on 11/25/2016).

[5] Linux-Vserver. Linux-vserver. http://www.linux-vserver.org, June 2013. (Accessed on 11/16/2016).

[6] D. Price, A. Tucker. Solaris zones: Operating system support for consolidating commercial workloads. In LISA, volume 4, pages 241–254, 2004.

[7] J.Victor. SolarisTM containerstechnologyarchitectureguide. Sun Microsystems, pages 91–115, 2006.

[8] ClusterHQ. The current state of container usage. https:// clusterhq.com/assets/pdfs/state-of-container-usage-june-2015. Pdf, 2015. (Accessed on 11/30/2016).

[9] OpenVZ. Openvz virtuozzo containers wiki. https://openvz. Org/Main Page, November 2016. (Accessed on 11/16/2016).

[10] PARALLELS. An introduction to operating system virtualization and parallels cloud server. https: //virtuozzo.com/wp-content/uploads/2016/03/Virtuozzo Intro OS Virtualization WP Ltr 2013 March2016.pdf, 2016. (Accessed on 11/16/2016).

[11] Virtuozzo. Virtuozzo – containers, vms, storage virtualization. https://virtuozzo.com/, 2016. (Accessed on 11/16/2016).

[12] Z. Kozhirbayev, R.O. Sinnott. A performance comparison of container-based technologies for the cloud. Future Generation Computer Systems, 68:175–182, 2016.

[13] Canonical Ltd. Linux containers – lxc – documentation. https: //linuxcontainers.org/lxc/documentation/,2016. (Accessedon 11/24/2016).

[14] S. Graber. Lxc 1.0 release. https://lists.linuxcontainers.org/ pipermail/lxc-devel/2014-February/008165.html, 2014. (Accessed on 11/25/2016).

[15] Canonical Ltd. Linux containers – lxd – introduction. https: //linuxcontainers.org/lxd/introduction/, 2016. (Accessed on 11/24/2016

[16] Docker. Docker engine frequently asked questions (faq) docker. https://docs.docker.com/engine/faq/, 2016. (Accessed on 11/24/2016).