

Problem: How to build a custom web-app image and link to mongodb container using docker-compose.

Solution:

Before we will start solving this problem let's understand some important points.

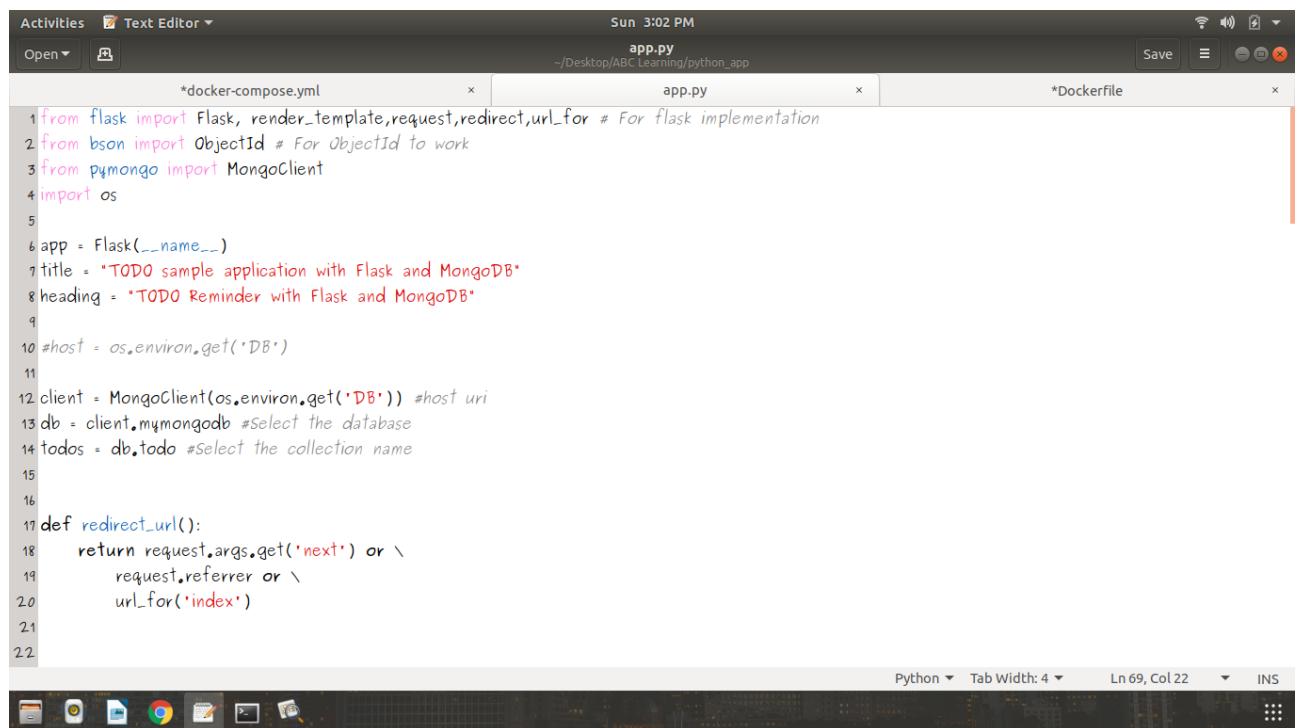
What is docker-compose.yml file ?

A docker-compose.yml is a yaml file which defines how Docker container will behave when you will execute this file.

In this problem we will run a python web application(flask) which is linked with mongodb nosql database.

Let's start:

Step1: First create a python web application and name that file as 'app.py'



The screenshot shows a desktop environment with a terminal window open. The terminal has three tabs: 'docker-compose.yml', 'app.py', and 'Dockerfile'. The 'app.py' tab is active and contains the following Python code:

```
1 from flask import Flask, render_template, request, redirect, url_for # For flask implementation
2 from bson import ObjectId # For ObjectId to work
3 from pymongo import MongoClient
4 import os
5
6 app = Flask(__name__)
7 title = "TODO sample application with Flask and MongoDB"
8 heading = "TODO Reminder with Flask and MongoDB"
9
10 #host = os.environ.get('DB')
11
12 client = MongoClient(os.environ.get('DB')) #host uri
13 db = client.mymongodb #Select the database
14 todos = db.todo #Select the collection name
15
16
17 def redirect_url():
18     return request.args.get('next') or \
19         request.referrer or \
20         url_for('index')
21
22
```

The code is a simple Flask application that interacts with a MongoDB database named 'mymongodb' and a collection named 'todo'. It includes a helper function 'redirect_url' to handle redirection logic.

Step2: Create a Dockerfile which will create a custom image that will execute your application.

```
1 # Use an official Python runtime as a parent image
2 FROM python:2.7-slim
3 # Set the working directory to /app
4 WORKDIR /app
5 # Copy the current directory contents into the container at /app
6 COPY . /app
7 # Install any needed packages specified in requirements.txt
8 RUN pip install --trusted-host pypi.python.org -r requirements.txt
9 # Make port 80 available to the world outside this container
10 EXPOSE 5000
11 # Define environment variable
12 ENV NAME World
13 # Run app.py when the container launches
14 CMD ["python", "app.py"]
```

Step3: Create a file named ‘docker-compose.yml’ where we will write a script which will tell docker to do what is mentioned in ‘docker-compose.yml’.

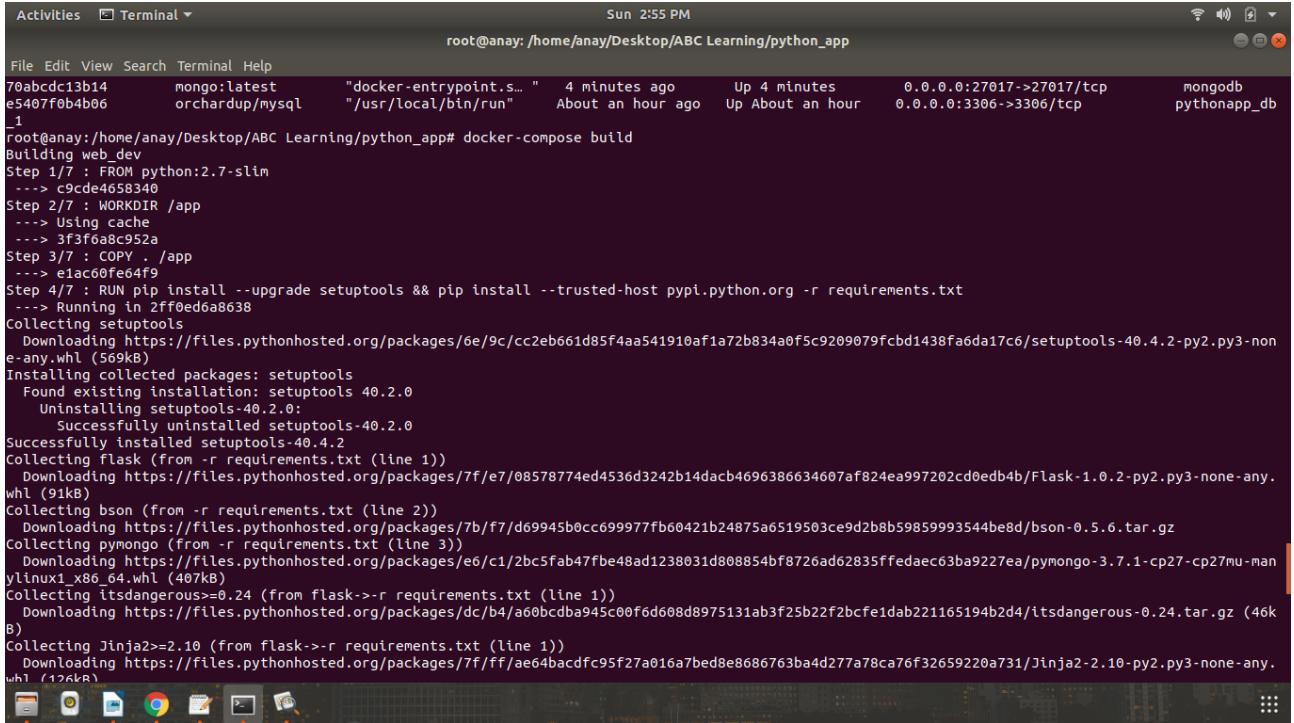
```
version: '3.5'
services:
  web_dev:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./app
    environment:
      - ENV=development
      - PORT=5000
      - DB=mongodb://mongodb:27017
  mongodb:
    image: mongo:latest
    container_name: mongodb
    environment:
      - MONGO_DATA_DIR=/usr/data/db
      - MONGO_LOG_DIR=/dev/null
    volumes:
      - ./data/db:/usr/data/db
    ports:
      - "27017:27017"
```

Step4: To run this docker-compose.yml file we need to have docker-compose service in our system . If you do not have this, you can install by this command ‘apt-get install docker-compose’.

To run this docker-compose.yml file, we need to execute a command ‘docker-compose up -d’ or to build your application we can run a command ‘docker-compose build’.

First we will build this docker-compose file.

```
$ docker-compose build
```



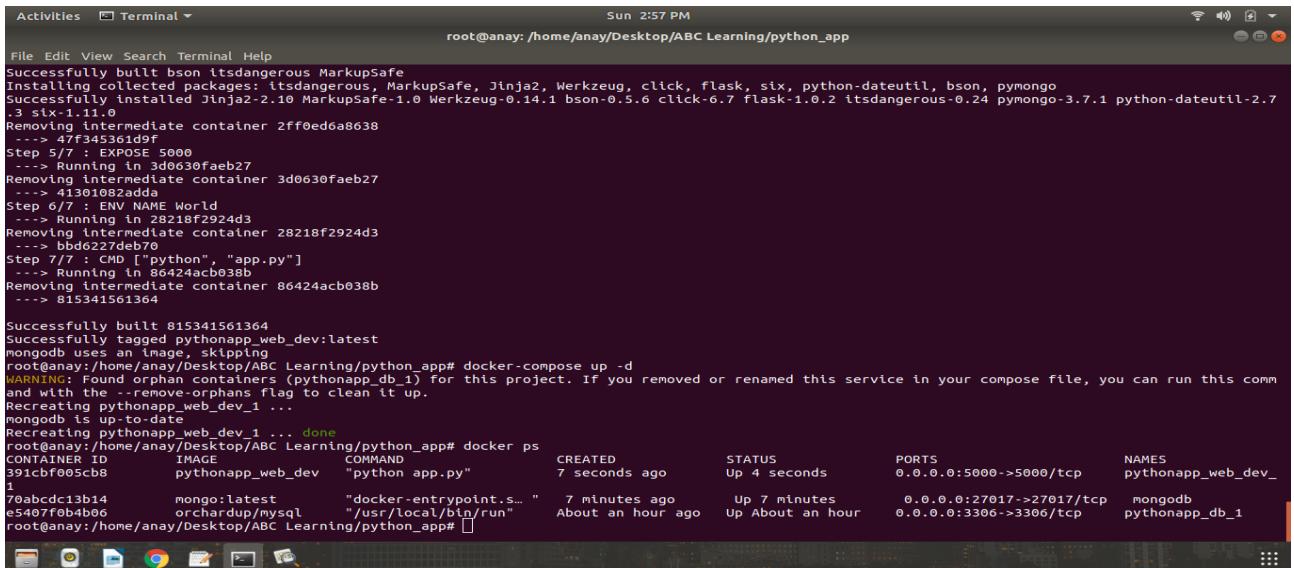
```
Activities Terminal Sun 2:55 PM
root@anay:/home/anay/Desktop/ABC Learning/python_app

File Edit View Search Terminal Help
70abcdc13b14      mongo:latest      "docker-entrypoint.s..."    4 minutes ago   Up 4 minutes   0.0.0.0:27017->27017/tcp      mongodb
e5407f0b4b06      orchardup/mysql   "/usr/local/bin/run"   About an hour ago Up About an hour  0.0.0.0:3306->3306/tcp      pythonapp_db
_1
root@anay:/home/anay/Desktop/ABC Learning/python_app# docker-compose build
Building web dev
Step 1/7 : FROM python:2.7-slim
--> c9cde4658340
Step 2/7 : WORKDIR /app
--> Using cache
--> 3f3f6a8c952a
Step 3/7 : COPY . /app
--> e1ac60fe64f9
Step 4/7 : RUN pip install --upgrade setuptools && pip install --trusted-host pypi.python.org -r requirements.txt
--> Running in 2ff0ed6a8638
Collecting setuptools
  Downloading https://files.pythonhosted.org/packages/6e/9c/cc2eb661d85f4aa541910af1a72b834a0f5c9209079fcbd1438fa6da17c6/setuptools-40.4.2-py2.py3-none-any.whl (569kB)
Installing collected packages: setuptools
  Found existing installation: setuptools 40.2.0
    Uninstalling setuptools-40.2.0:
      Successfully uninstalled setuptools-40.2.0
Successfully installed setuptools-40.4.2
Collecting flask (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7f/e7/08578774ed4536d3242b14dacb4696386634607af824ea997202cd0edb4b/Flask-1.0.2-py2.py3-none-any.whl (91kB)
Collecting bson (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/7b/f7/d69945b0cc699977fb60421b24875a6519503ce9d2b8b59859993544be8d/bson-0.5.6.tar.gz
Collecting pymongo (from -r requirements.txt (line 3))
  Downloading https://files.pythonhosted.org/packages/e6/c1/2bc5fab47fbe48ad1238031d808854bf8726ad62835ffedaec63ba9227ea/pymongo-3.7.1-cp27-cp27mu-manylinux1_x86_64.whl (407kB)
Collecting itsdangerous<=0.24 (from flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/dc/b4/a60bcd8a945c00f6d608d8975131ab3f25b22f2bcfe1dab221165194b2d4/itsdangerous-0.24.tar.gz (46kB)
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-none-any.whl (126kB)

```

After building , we can run this as well through a command ‘docker-compose up -d’.

```
$ docker-compose up -d
```



```
Activities Terminal Sun 2:57 PM
root@anay:/home/anay/Desktop/ABC Learning/python_app

File Edit View Search Terminal Help
Successfully built bson itsdangerous MarkupSafe
Installing collected packages: itsdangerous, MarkupSafe, Jinja2, Werkzeug, click, flask, six, python-dateutil, bson, pymongo
Successfully installed Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 bson-0.5.6 click-6.7 flask-1.0.2 itsdangerous-0.24 pymongo-3.7.1 python-dateutil-2.7.3 six-1.11.0
Removing intermediate container 2ff0ed6a8638
--> 47f345361d9f
Step 5/7 : EXPOSE 5000
--> Running in 3d0630faeb27
Removing intermediate container 3d0630faeb27
--> 4301082adda
Step 6/7 : ENV NAME World
--> Running in 28218f2924d3
Removing intermediate container 28218f2924d3
--> bb6d227deb70
Step 7/7 : CMD ["python", "app.py"]
--> Running in 86424acb038b
Removing intermediate container 86424acb038b
--> 815341561364

Successfully built 815341561364
Successfully tagged pythonapp_web_dev:latest
mongodb uses an image, skipping
root@anay:/home/anay/Desktop/ABC Learning/python_app# docker-compose up -d
WARNING: Found orphan containers (pythonapp_db_1) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphan flag to clean it up.
Recreating pythonapp_web_dev_1 ...
mongodb is up-to-date
Recreating pythonapp_web_dev_1 ... done
root@anay:/home/anay/Desktop/ABC Learning/python_app# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS                         NAMES
391cbf00scb8        pythonapp_web_dev   "python app.py"   7 seconds ago     Up 4 seconds          0.0.0.0:5000->5000/tcp   pythonapp_web_dev_
_1
70abcdc13b14        mongo:latest        "docker-entrypoint.s..." 7 minutes ago     Up 7 minutes         0.0.0.0:27017->27017/tcp   mongodb
e5407f0b4b06        orchardup/mysql   "/usr/local/bin/run" About an hour ago Up About an hour  0.0.0.0:3306->3306/tcp   pythonapp_db_1
root@anay:/home/anay/Desktop/ABC Learning/python_app#
```

Now, we can see that two containers are running named pythonapp_web_dev_1 and mongodb. Now open your browser and enter the url '<http://localhost:5000>'.

Problem: Docker example to show the difference between RUN , CMD and ENTRYPOINT.

Docker build images automatically by reading the instruction that we have written in our Dockerfile. And one can say that Dockerfile is a text document that contains all the commands or instruction a user could call on the command line to create an Image.

This scenario will describe you about some of those commands which are RUN, CMD, Entry Point.

1. RUN:

Run has two forms:

---> RUN<command>

The RUN command is the central executing directive for Dockerfiles. It takes a command as its argument and runs it to form the image. Unlike CMD, it actually is used to build the image (forming another layer on top of the previous one which is committed).

Example:

Lets take a base image of ubuntu:latest and we will install jdk in it. So for this task our Dockerfile would be like this as given below.

```
FROM ubuntu:latest  
  
RUN apt-get update -y && apt-get upgrade -y && apt-get install default-jdk -y
```

After building this Dockerfile we will get to know that JDK is installed in this image.

```
root@anay:/home/anay/Desktop/ABC Learning/RUN# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
jdk_app	latest	1958d4e6b915	12 minutes ago	632MB

We will run this image and run a command in java -version.

```
root@b047807c2138:/# java -version
openjdk version "10.0.2" 2018-07-17
OpenJDK Runtime Environment (build 10.0.2+13-Ubuntu-1ubuntu0.18.04.2)
OpenJDK 64-Bit Server VM (build 10.0.2+13-Ubuntu-1ubuntu0.18.04.2, mixed mode)
```

2. CMD

The command CMD, similarly to RUN, can be used for executing a specific command. However, unlike RUN it is not executed during build, but when a container is instantiated using the image being built. Therefore, it should be considered as an initial, default command that gets executed (i.e. run) with the creation of containers based on the image.

an example for CMD would be running an application upon creation of a container which is already installed using RUN (e.g. RUN apt-get install ...) inside the image.

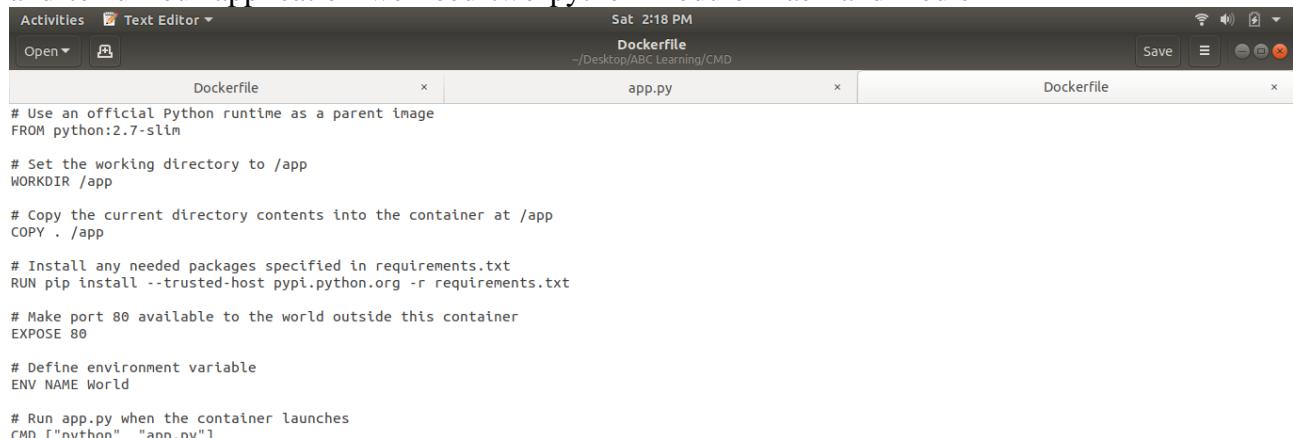
Usage 1: CMD application "argument", "argument", ..

CMD "echo" "Hello docker!"

Example: We will deploy a simple python application in docker conatiner.

We have a dockerfile for that. We will be using python2.7-slim

and to run our application we need two python module Flask and Redis



```
# 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

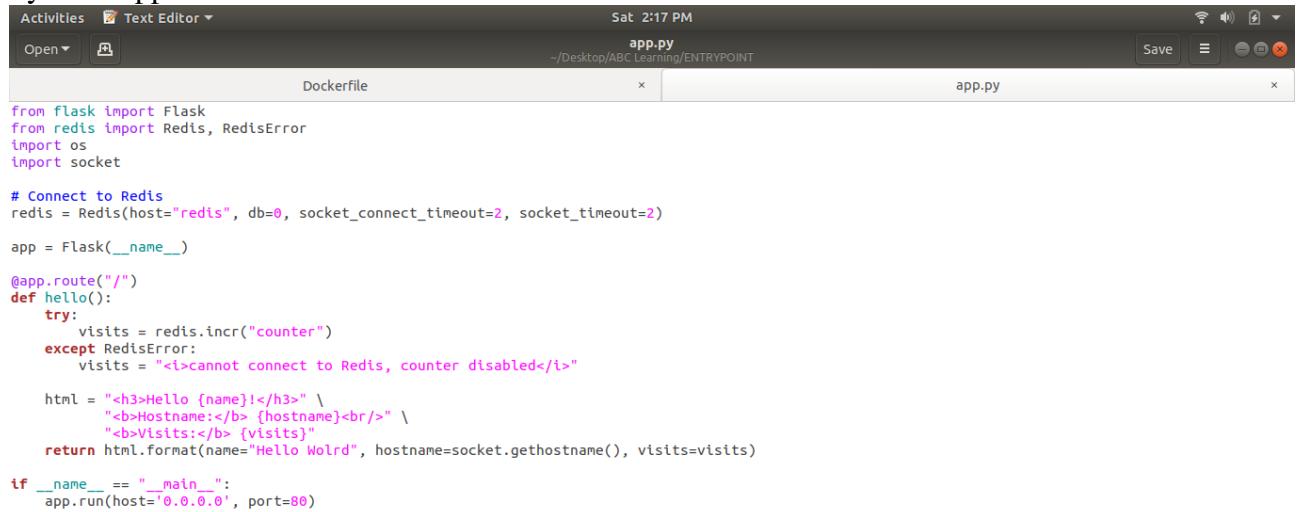
# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```



Python Application:



```
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)

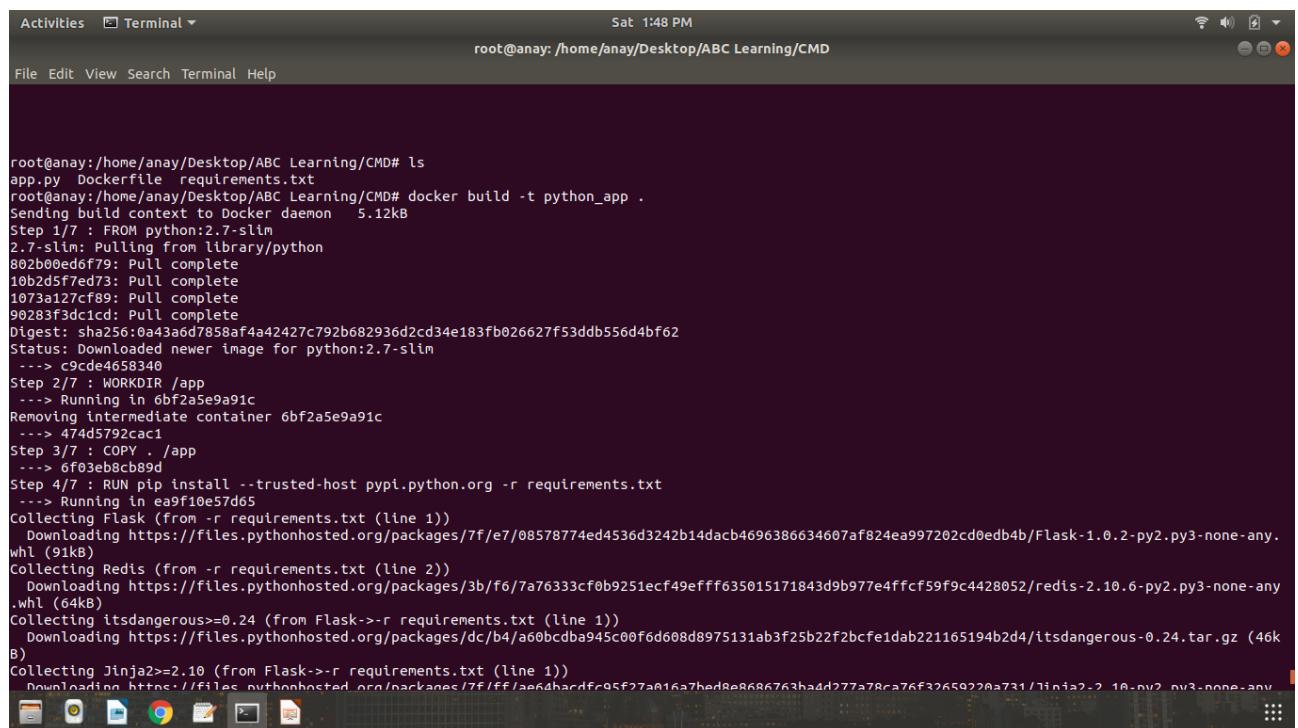
app = Flask(__name__)

@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="Hello World", hostname=socket.gethostname(), visits=visits)

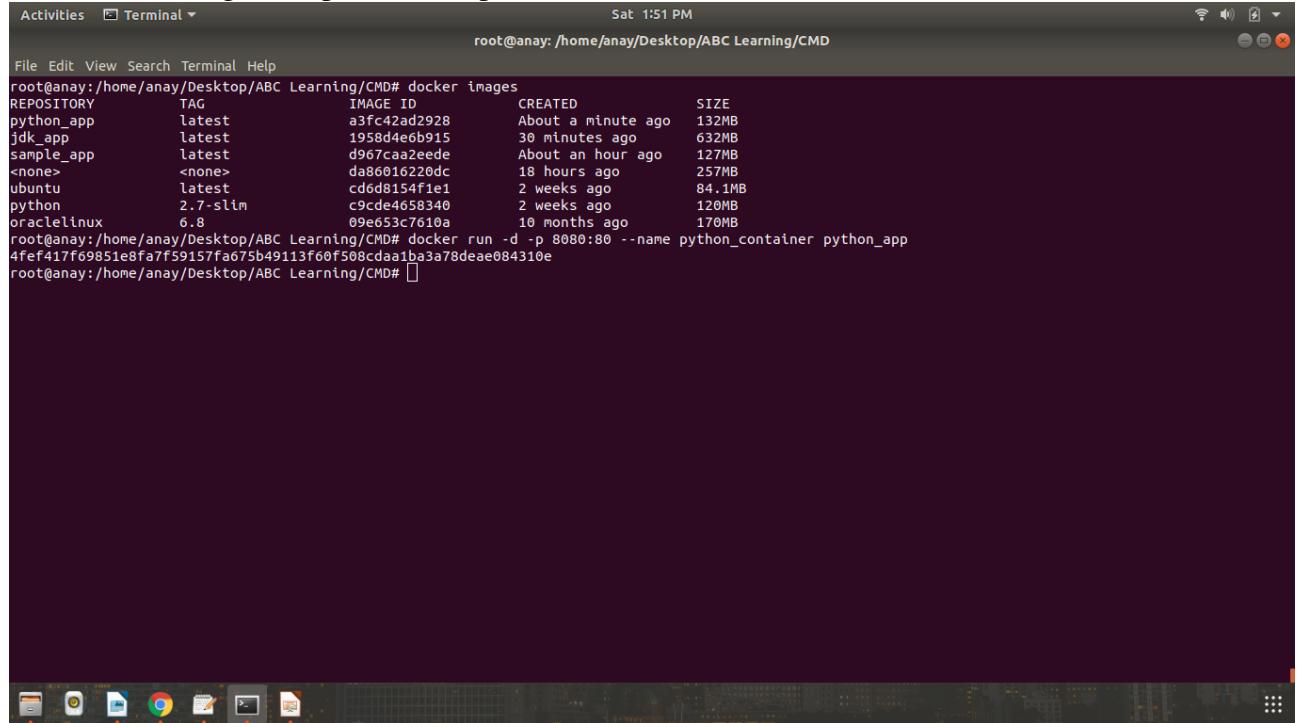
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80)
```

Building Dockerfile:



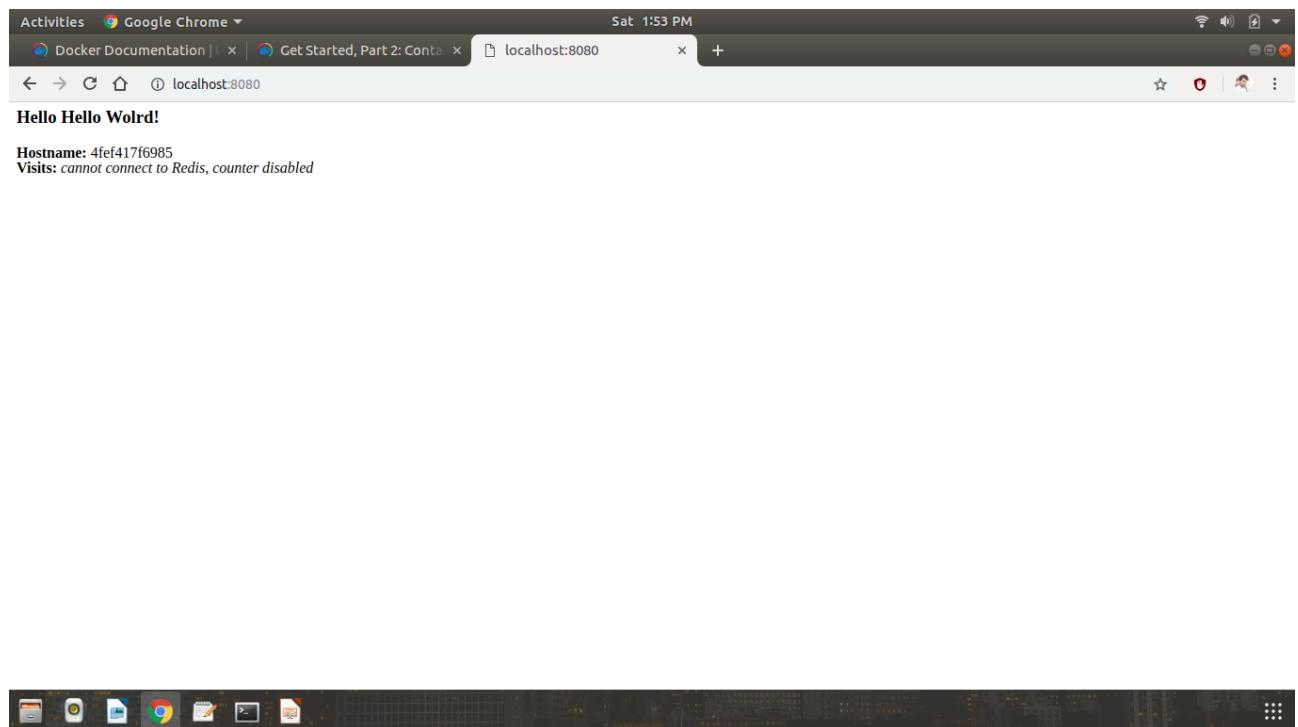
```
root@anay:/home/anay/Desktop/ABC Learning/CMD# ls
app.py Dockerfile requirements.txt
root@anay:/home/anay/Desktop/ABC Learning/CMD# docker build -t python_app .
Sending build context to Docker daemon 5.12kB
Step 1/7 : FROM python:2.7-slim
2.7-slim: Pulling from library/python
802b00ed0f79: Pull complete
10b2d5f7ed73: Pull complete
1073a127cf89: Pull complete
90283f3dc1cd: Pull complete
Digest: sha256:0a43a6d7858af4a42427c792b682936d2cd34e183fb026627f53ddb556d4bf62
Status: Downloaded newer image for python:2.7-slim
Status: Downloaded newer image for python:2.7-slim
--> c9cde4658340
Step 2/7 : WORKDIR /app
--> Running in 6bf2a5e9a91c
Removing intermediate container 6bf2a5e9a91c
--> 474d5792cac1
Step 3/7 : COPY . /app
--> 6f03eb8cb89d
Step 4/7 : RUN pip install --trusted-host pypi.python.org -r requirements.txt
--> Running in ea9f10e57d65
Collecting Flask (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7f/e7/08578774ed4536d3242b14dacb4696386634607af824ea997202cd0edb4b/Flask-1.0.2-py2.py3-none-any.whl (91kB)
Collecting Redis (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/3b/f6/7a76333cf0b9251ecf49efff635015171843d9b977e4ffcf59f9c4428052/redis-2.10.6-py2.py3-none-any.whl (64kB)
Collecting itsdangerous>=0.24 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/dc/b4/a60bcd945c00f6d608d8975131ab3f25b22f2bcfe1dab221165194b2d4/itsdangerous-0.24.tar.gz (46kB)
Collecting Jinja2>=2.10 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7f/ff/ae64hacdfc95f27a016a7bed8e8686763ba4d277a78ca76f32659220a731/Jinja2-2.10-py2.py3-none-any.whl (114kB)
```

Run docker image and publish the port no:



```
Activities Terminal Sat 1:51 PM
root@anay:/home/anay/Desktop/ABC Learning/CMD# docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
python_app      latest       a3fc42ad2928   About a minute ago  132MB
jdk_app         latest       1958d4e6b915   30 minutes ago  632MB
sample_app      latest       d967caa2zeede  About an hour ago  127MB
<none>          <none>      da86016220dc   18 hours ago   257MB
ubuntu          latest       cd6d8154f1e1   2 weeks ago    84.1MB
python          2.7-slim    c9cde4658340   2 weeks ago    120MB
oraclelinux     6.8         09e653c7610a   10 months ago   170MB
root@anay:/home/anay/Desktop/ABC Learning/CMD# docker run -d -p 8080:80 --name python_container python_app
4fef417f69851e8fa7f59157fa675b49113f60f508cd01ba3a78deae084310e
root@anay:/home/anay/Desktop/ABC Learning/CMD#
```

Now open to test if app is running or not open the browser and open the url <http://localhost:8080>.



3.ENTRYPOINT: This argument sets the concrete default application that is used every time a container is created using the image. For example, if you have installed a specific application inside an image and you will use this image to only run that application, you can state it with ENTRYPOINT and whenever a container is created from that image, your application will be the target.

```
# Usage: ENTRYPOINT application "argument", "argument", ..
```

```
# Remember: arguments are optional. They can be provided by CMD or during the creation of a container.
```

```
# Usage example with CMD:
```

```
# Arguments set with CMD can be overridden during *run*
```

```
CMD "Hello docker!"
```

```
ENTRYPOINT echo
```

NOTE: Entrypoint is use to set the command and parameters that will be exectued very first time when a container start running.

The syntax of Entrypoint is :

```
ENTRYPOINT ["executable","parameter1"]
```

Dockerfile:



```
Activities Text Editor ▾ Sat 2:19 PM
Open ▾ Dockerfile
*Dockerfile
~/Desktop/ABC Learning/ENTRYPOINT
Save ⌂ ⌓ ⌚

# 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

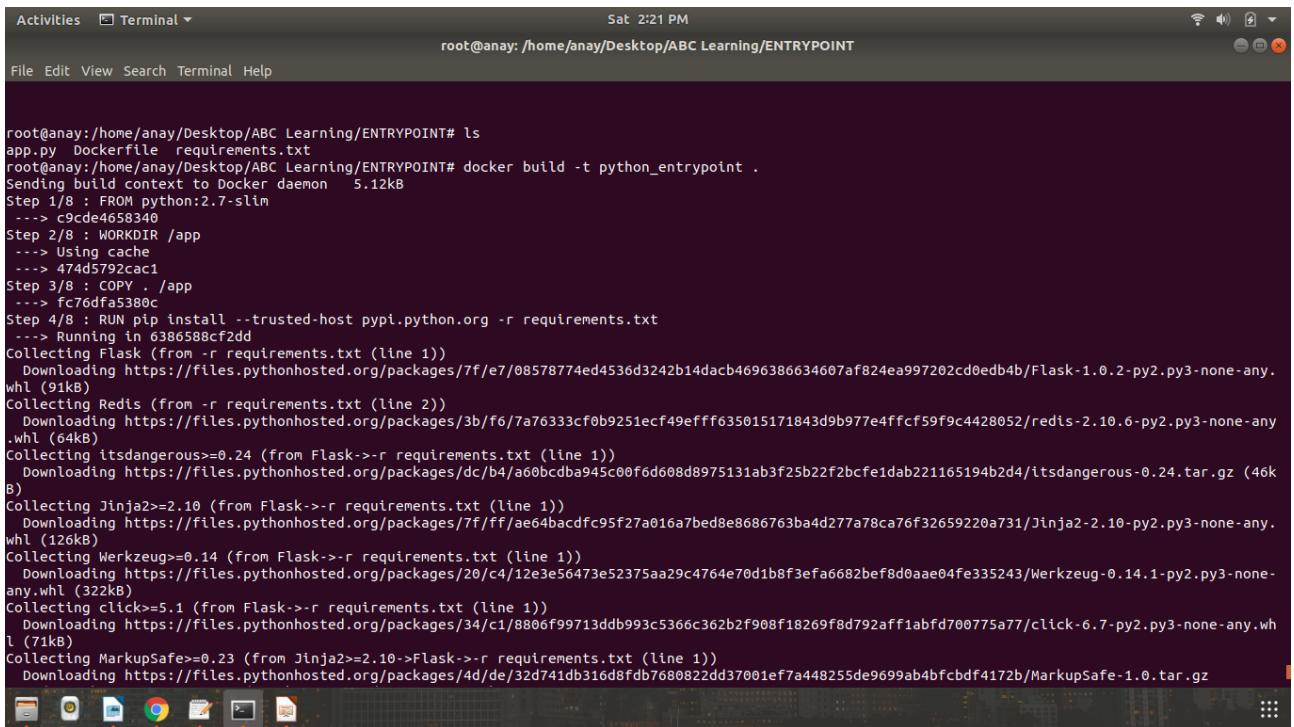
# Define environment variable
ENV NAME World

# Run app.py when the container launches
ENTRYPOINT ["python"]
CMD ["app.py"]

#CMD ["python", "app.py"]
```



Building the docker file:

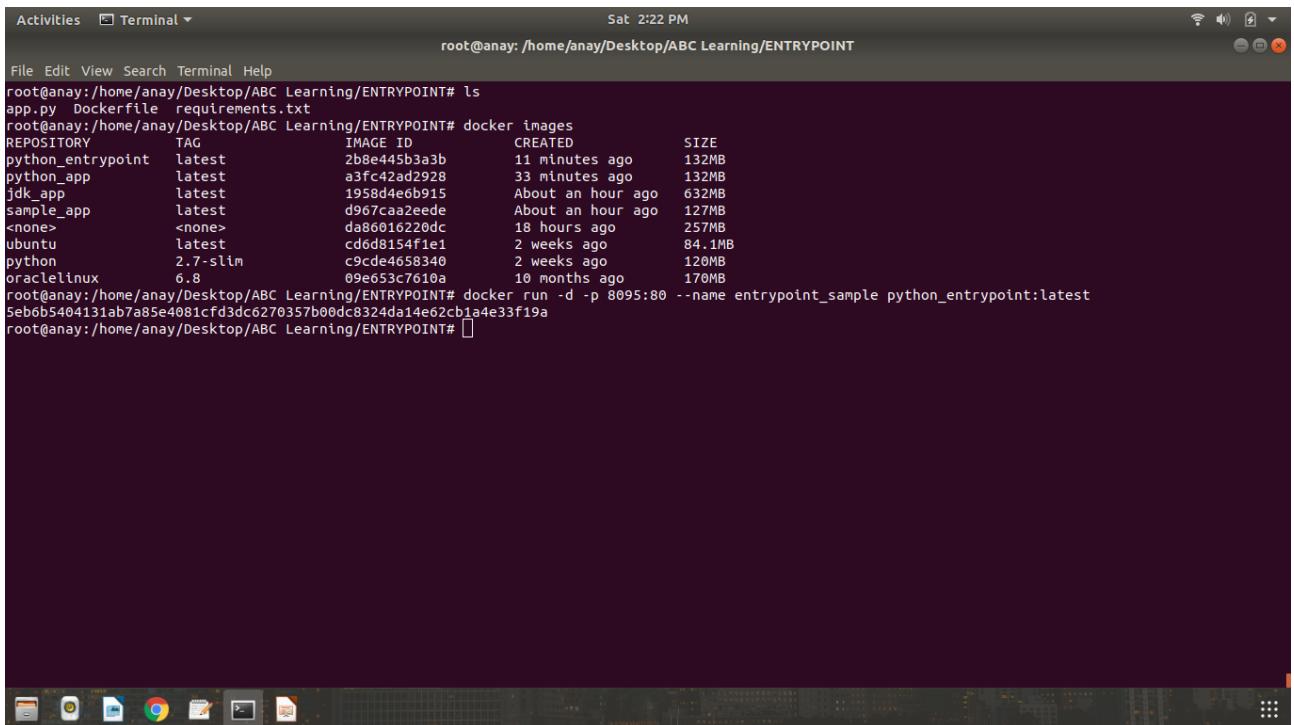


```
Sat 2:21 PM
root@anay:/home/anay/Desktop/ABC Learning/ENTRYPPOINT

File Edit View Search Terminal Help

root@anay:/home/anay/Desktop/ABC Learning/ENTRYPPOINT# ls
app.py Dockerfile requirements.txt
root@anay:/home/anay/Desktop/ABC Learning/ENTRYPPOINT# docker build -t python_entrypoint .
Sending build context to Docker daemon 5.12kB
Step 1/8 : FROM python:2.7-slim
--> c9cde4658340
Step 2/8 : WORKDIR /app
--> Using cache
--> 474d5792cac1
Step 3/8 : COPY . /app
--> fc76dfa5380c
Step 4/8 : RUN pip install --trusted-host pypi.python.org -r requirements.txt
--> Running in 6386588cf2dd
Collecting Flask (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/7f/e7/08578774ed4536d3242b14dacb4696386634607af824ea997202cd0edb4b/Flask-1.0.2-py2.py3-none-any.whl (91kB)
Collecting Redis (from -r requirements.txt (line 2))
  Downloading https://files.pythonhosted.org/packages/3b/f6/7a76333cf0b9251ecf49efff635015171843d9b977e4ffcf59f9c4428052/redis-2.10.6-py2.py3-none-any.whl (64kB)
Collecting itsdangerous>=0.24 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/dc/b4/a60bcdcb945c00f6d608d8975131ab3f25b22f2bcfe1dab221165194b2d4/itsdangerous-0.24.tar.gz (46kB)
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-none-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.py3-none-any.whl (322kB)
Collecting click>=5.1 (from Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/34/c1/8806f99713ddb993c5366c362b2f908f18269f8d792aff1abfd700775a77(click-6.7-py2.py3-none-any.whl (71kB)
Collecting MarkupSafe>=0.23 (from Jinja2>=2.10->Flask->-r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/4d/de/32d741db316d8fdb7680822dd37001ef7a448255de9699ab4bfcbdf4172b/MarkupSafe-1.0.tar.gz
```

Running docker image:

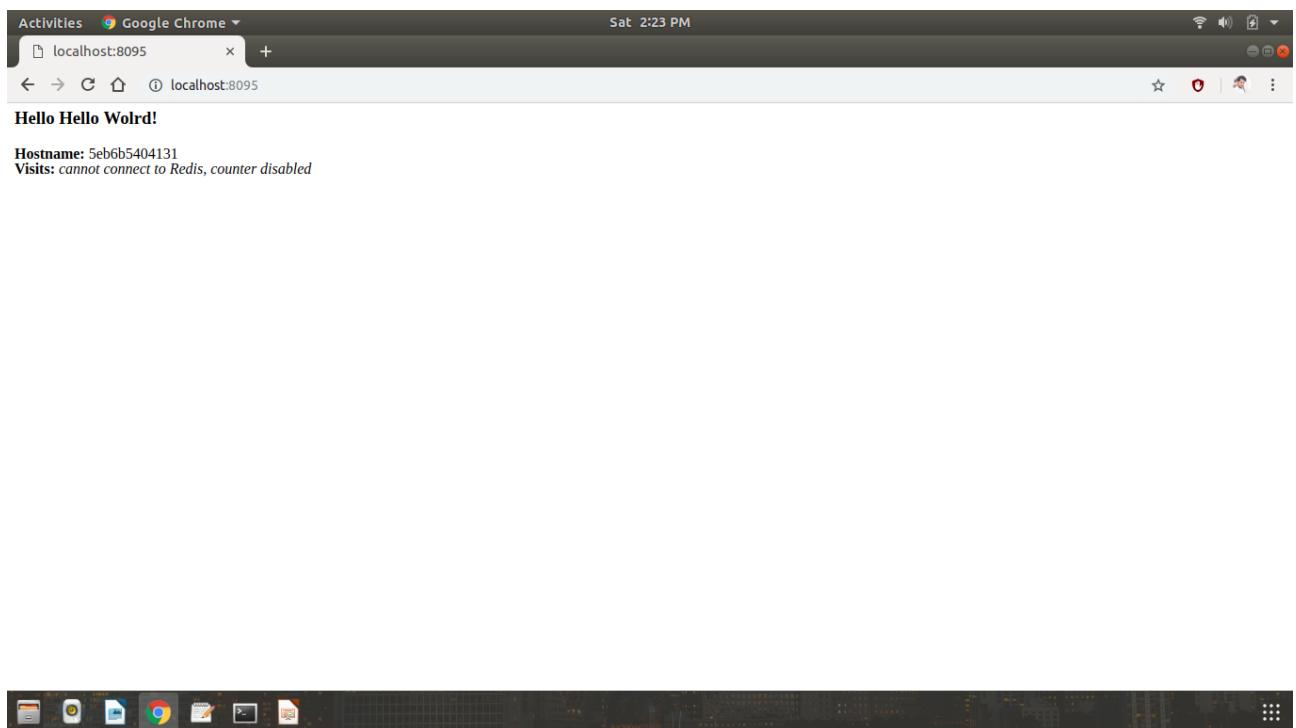


```
Sat 2:22 PM
root@anay:/home/anay/Desktop/ABC Learning/ENTRYPPOINT

File Edit View Search Terminal Help

root@anay:/home/anay/Desktop/ABC Learning/ENTRYPPOINT# ls
app.py Dockerfile requirements.txt
root@anay:/home/anay/Desktop/ABC Learning/ENTRYPPOINT# docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
python_entrypoint   latest   2b8e445b3a3b    11 minutes ago  132MB
python_app          latest   a3fc42ad2928    33 minutes ago  132MB
jdk_app             latest   1958d4eeb915    About an hour ago  632MB
sample_app          latest   d967caa2eede  About an hour ago  127MB
<none>              <none>  da86016220dc  18 hours ago   257MB
ubuntu              latest   cd6d8154f1e1    2 weeks ago    84.1MB
python               2.7-slim  c9cde4658340    2 weeks ago    120MB
oraclelinux          6.8     09e653c7610a   10 months ago   170MB
root@anay:/home/anay/Desktop/ABC Learning/ENTRYPPOINT# docker run -d -p 8095:80 --name entrypoint_sample python_entrypoint:latest
5eb6b5404131ab7a85e4081cf3dc6270357b00dc8324da14e62cb1a4e33f19a
root@anay:/home/anay/Desktop/ABC Learning/ENTRYPPOINT# 
```

Testing the application:



Docker Volume:

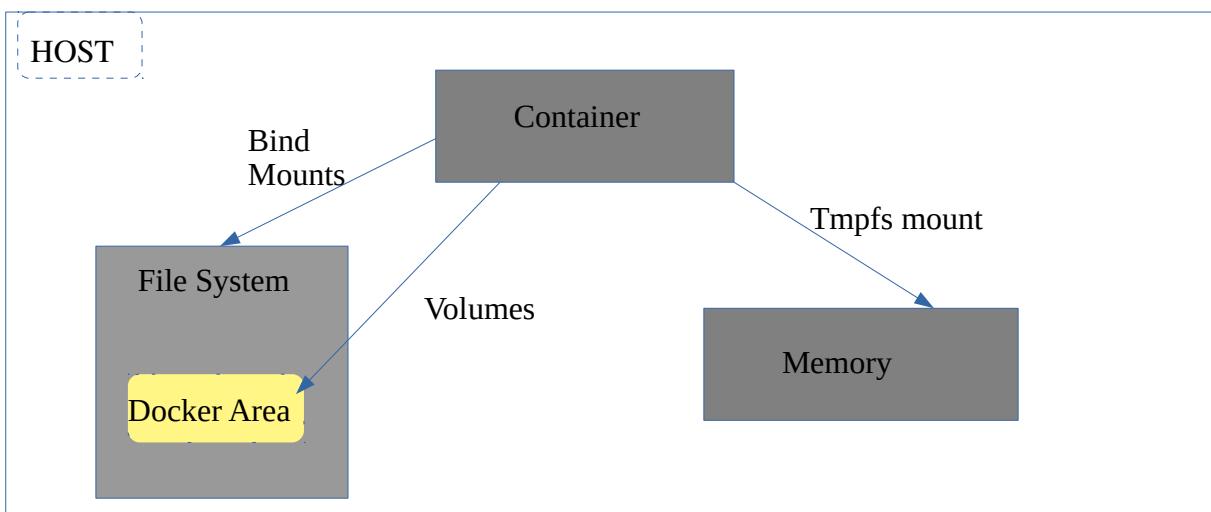
Let's imagine that you run a mongodb/mysql container. Now you are storing your data in mongodb/mysql db. Now if you need to take a backup of this or you want your data to be persists then docker volume would be the best option for you.

Docker volumes are the preferred method for persisting data which is generated and used by Docker containers. These volumes are completely managed by Docker itself.

There are some advantages of Docker volumes over bind mounts:

1. We can easily take back up of container or migrate than bind mounts.
2. We can manage Docker volumes by using Docker rest api i.e. Docker CLI etc.
3. We can safely share volumes among multiple container.
4. Volumes doesn't increase the size of container.

NOTE: The content of volumes exist outside the lifecycle of a container.



Originally, for volumes we use `-v` or `--volume` flag for standalone containers and `--mount` flag for swarm services.

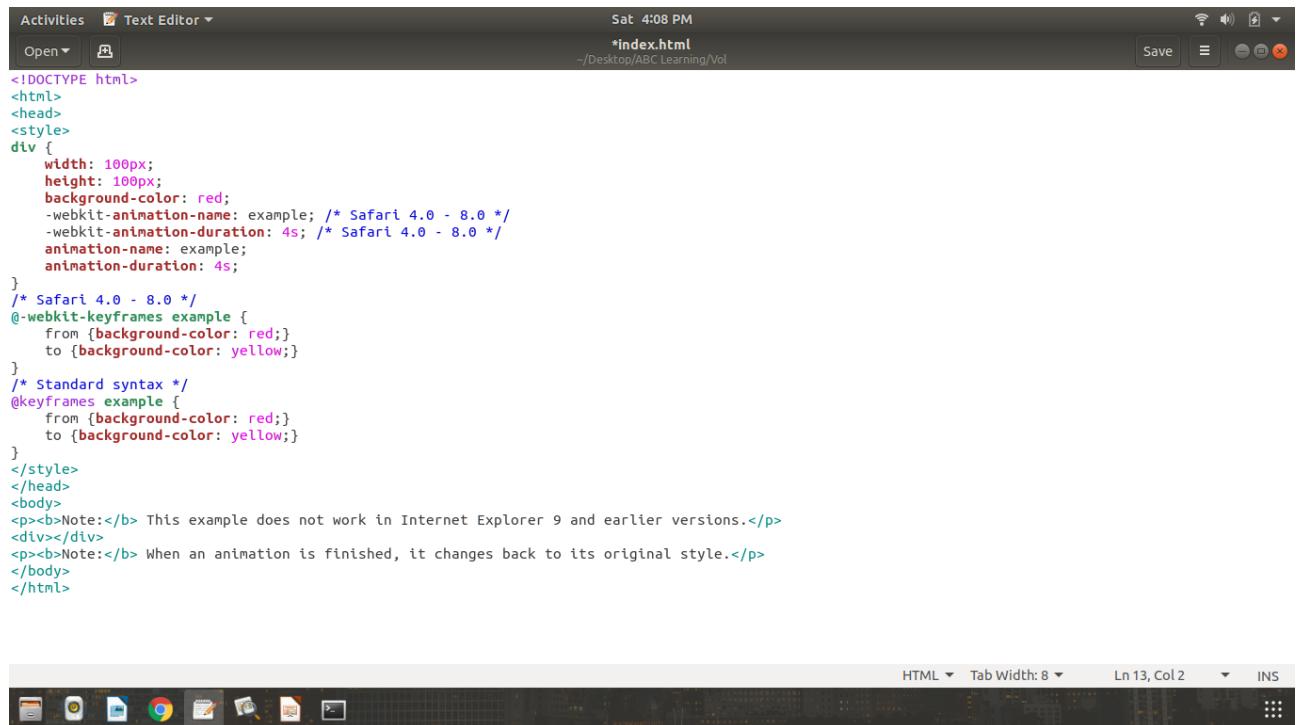
The difference between `-v` syntax and `--mount` syntax is that `-v` syntax combines all the options together in one field as `--mount` syntax separates them.

Syntax of `--mount` is :

```
- - - mount 'type=volume,src=<VOLUME-NAME>,dst=<CONTAINER-PATH>,volume-driver=local,volume-opt=type=nfs,volume-opt=device=<nfs-server>:<nfs-path>,"volume-opt=o=addr=<nfs-address>,vers=4,soft,timeo=180,bg,tcp,rw'
```

Now, let's take an example, we will create a sample web application and will host at nginx server.

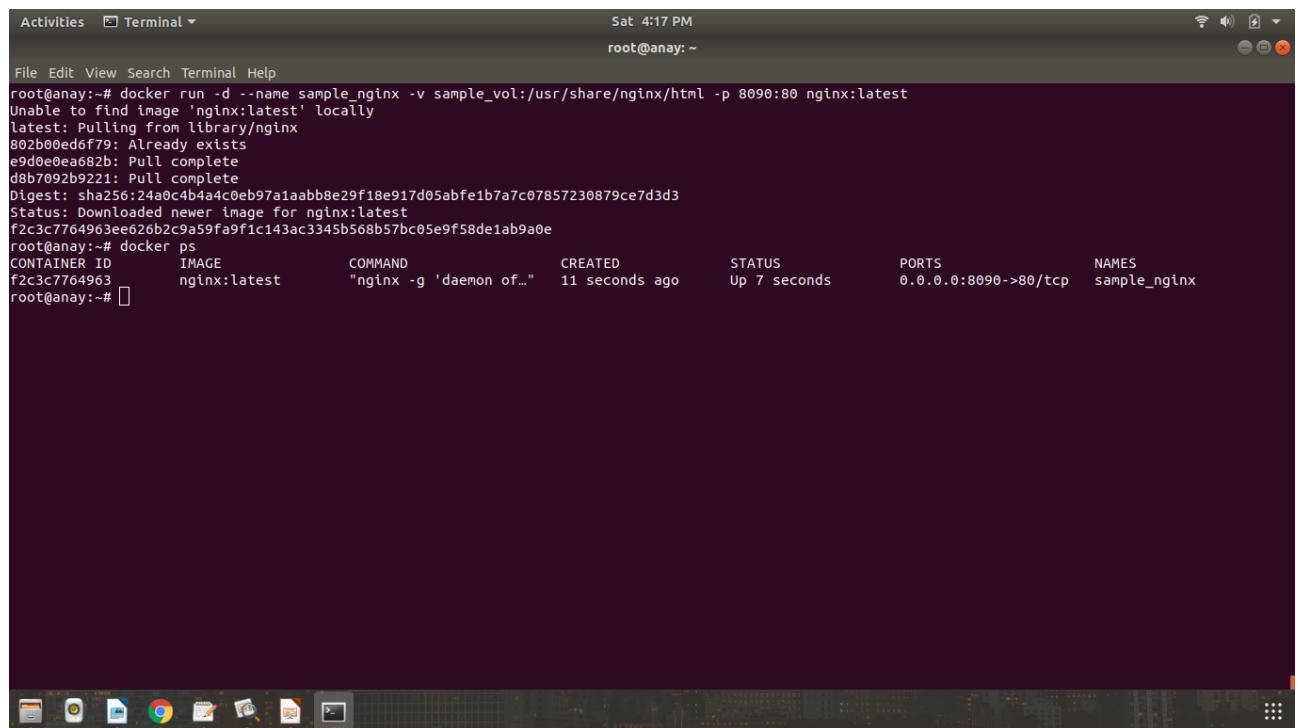
Step1: craete a html page.



```
Activities Text Editor ▾ Sat 4:08 PM
Open ▾ Save
*index.html
~/Desktop/ABC Learning/Vol
<!DOCTYPE html>
<html>
<head>
<style>
div {
    width: 100px;
    height: 100px;
    background-color: red;
    -webkit-animation-name: example; /* Safari 4.0 - 8.0 */
    -webkit-animation-duration: 4s; /* Safari 4.0 - 8.0 */
    animation-name: example;
    animation-duration: 4s;
}
/* Safari 4.0 - 8.0 */
@webkit-keyframes example {
    from {background-color: red;}
    to {background-color: yellow;}
}
/* Standard syntax */
@keyframes example {
    from {background-color: red;}
    to {background-color: yellow;}
}
</style>
</head>
<body>
<p><b>Note:</b> This example does not work in Internet Explorer 9 and earlier versions.</p>
<div></div>
<p><b>Note:</b> When an animation is finished, it changes back to its original style.</p>
</body>
</html>
```

Step2: Run an image nginx:latest with volumes.

If we start with a volumes that does not exist, docker will create the volume for us. Now we will mount the volume ‘sample_vol’ into ‘usr/share/nginx/html/’ in the container.

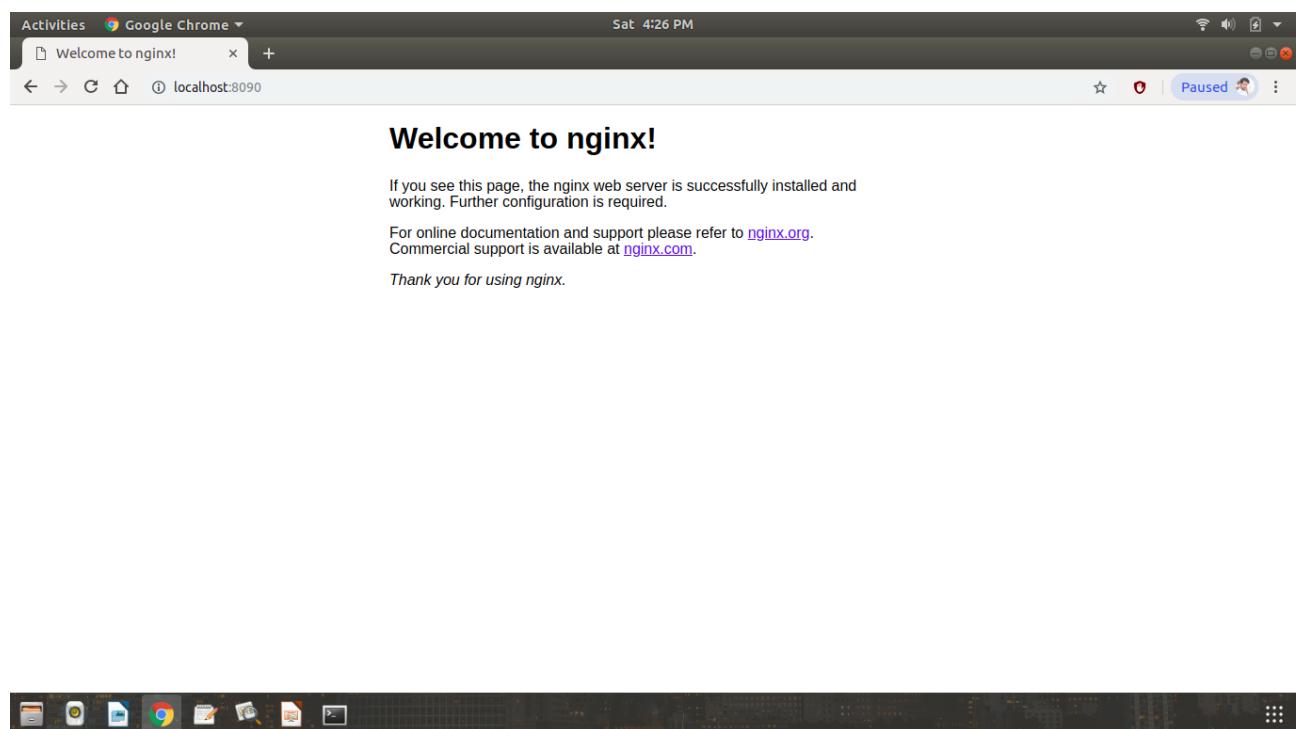


```
Activities Terminal ▾ Sat 4:17 PM
root@anay: ~
File Edit View Search Terminal Help
root@anay:~# docker run -d --name sample_nginx -v sample_vol:/usr/share/nginx/html -p 8090:80 nginx:latest
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
802b00eddf79: Already exists
e9d0e0ea682b: Pull complete
d0b7092b9221: Pull complete
Digest: sha256:24a0c4b4a4c0eb97a1aabb8e29f18e917d05abfe1b7a7c07857230879ce7d3d3
Status: Downloaded newer image for nginx:latest
f2c3c7764963ee626b2c9a59fa9f1c143ac3345b568b57bc05e9f58de1ab9a0e
root@anay:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                 NAMES
f2c3c7764963        nginx:latest       "nginx -g 'daemon of..."   11 seconds ago    Up 7 seconds          0.0.0.0:8090->80/tcp   sample_nginx
root@anay:~#
```

Step3: Now we will use docker inspect command to verify that the value was created and mounted correctly at time when we execute the previous command. We have to concentrate the Mounts section:

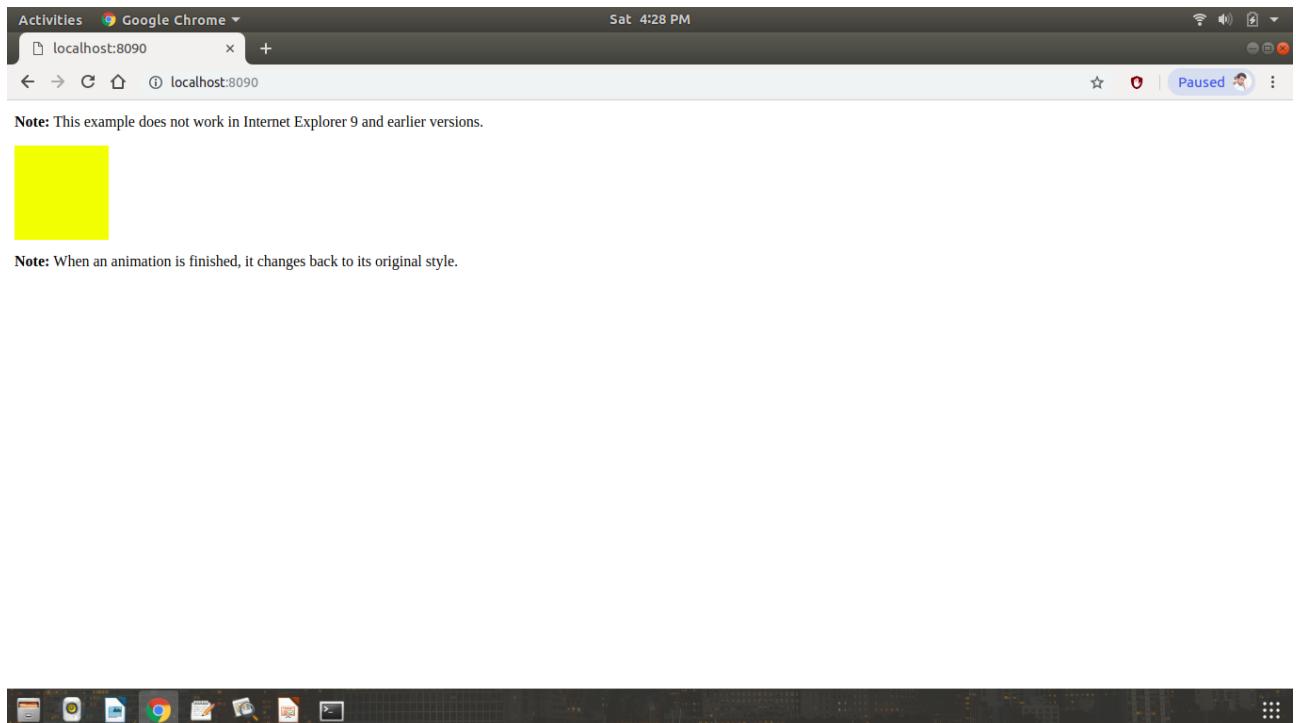
```
Activities Terminal Sat 4:21 PM
root@anay: ~
File Edit View Search Terminal Help
"GraphDriver": {
  "Data": [
    "LowerDir": "/var/lib/docker/overlay2/18f205c6eec5bcbfab14fb87e616c81a11a45e5e9a6c9a48ee0aabc3f1f942c3-init/diff:/var/lib/docker/overl
ay2/a0dfc22b602ec437b95c82b4800166863791e14a7a8d090a903eb05f5c5f03e3/diff:/var/lib/docker/overlay2/c1070ec2a4529d5ce10b6acf03f625c7598f1ae1361493cfe04
394e9a5dfc7b3/diff:/var/lib/docker/overlay2/e6743608ca783c7d6f98c8490c83ec071cceff335015e08df2f9caa7f6639f3/diff",
    "MergedDir": "/var/lib/docker/overlay2/18f205c6eec5bcbfab14fb87e616c81a11a45e5e9a6c9a48ee0aabc3f1f942c3/merged",
    "UpperDir": "/var/lib/docker/overlay2/18f205c6eec5bcbfab14fb87e616c81a11a45e5e9a6c9a48ee0aabc3f1f942c3",
    "WorkDir": "/var/lib/docker/overlay2/18f205c6eec5bcbfab14fb87e616c81a11a45e5e9a6c9a48ee0aabc3f1f942c3/work"
  ],
  "Name": "overlay2"
},
"Mounts": [
  {
    "Type": "volume",
    "Name": "sample_vol",
    "Source": "/var/lib/docker/volumes/sample_vol/_data",
    "Destination": "/usr/share/nginx/html",
    "Driver": "local",
    "Mode": "z",
    "RW": true,
    "Propagation": ""
  }
],
"Config": {
  "Hostname": "f2c3c7764963",
  "Domainname": "",
  "User": "",
  "AttachStdin": false,
  "AttachStdout": false,
  "AttachStderr": false,
  "ExposedPorts": {
    "80/tcp": {}
  },
  "Tty": false,
  "OpenStdin": false,
  "StdinOnce": false
}
}

```



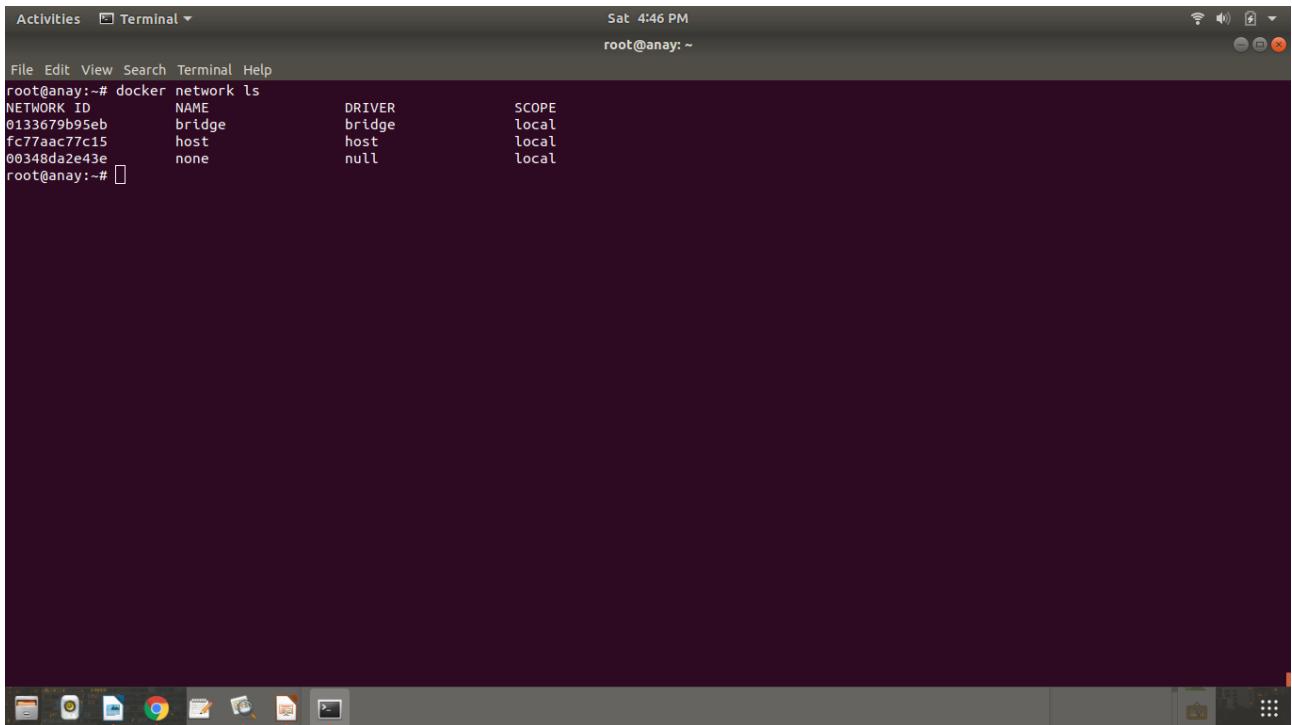
Step4: Next, we will navigate to ‘cd /var/lib/docker/volumes/sample_vol/_data/’ and will paste our html page here.

Step5: Open your browser and we will see our new html page.



Docker Networking

When we install Docker, it creates three network automatically, which can be listed using docker network command ‘docker network ls’



```
Activities Terminal Sat 4:46 PM root@anay: ~
File Edit View Search Terminal Help
root@anay:~# docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
0133679b95eb   bridge    bridge      local
fc77aac77c15   host      host       local
00348da2e43e   none     null       local
root@anay:~#
```

Docker network types:

Docker Network	
Bridge	The bridge network represents the docker0 network present in all Docker installations. Docker daemon connects containers to this network by default.
Host	Host network adds a container on the host's network stack. We will find that the network configuration inside the container is identical to the host.
None	The None network adds a container to a container -specific network stack. This container lacks a network interface.

When to use Bridge Network:

if you have a set of containers and each providing micro-services to the other and should not be exposed to the external world.

User – defined Bridge Network:

In terms of networking, a bridge network is a Link layer device which forwards traffic b/w network segments.

We can create our own bridge network.

- 1) User-defined bridges provides better isolation and interoperability between containerized applications.
- 2) User-defined bridges provides automatic DNS resolution between containers.
- 3) Containers can be attached and detached from user-defined network on the fly

NOTE: linked containers on the default bridge network share env variables.

We can manage a user-defined bridge, to create user defined network run the following command.

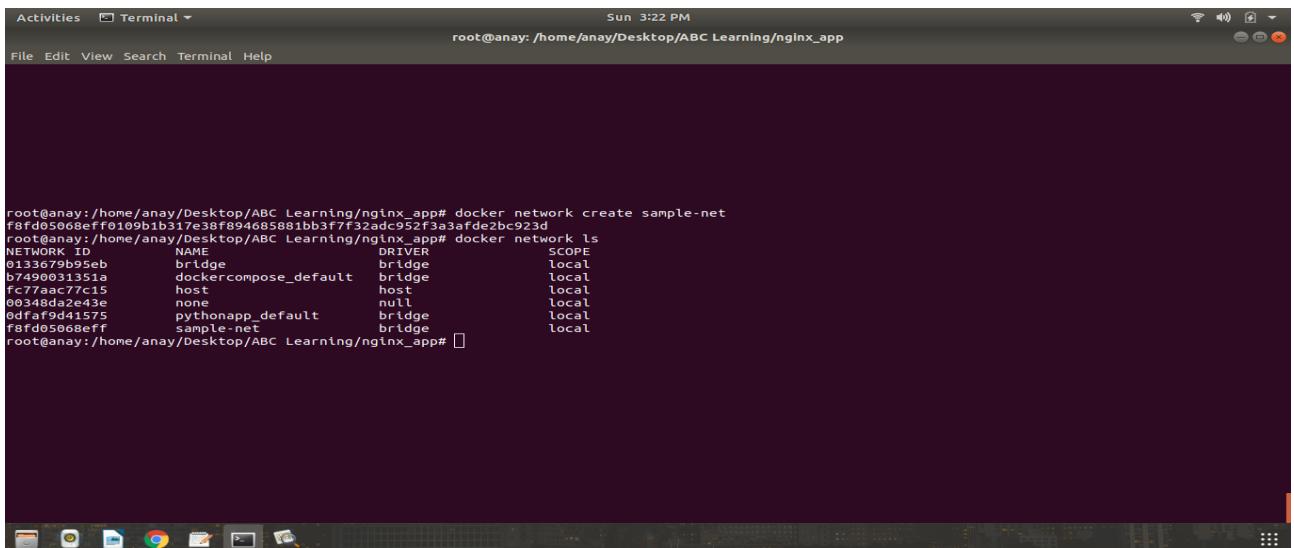
```
$ docker network create sample-network
```

And if you wanted to remove this bridge network run the following command. But before we remove it first disconnect all the container connected to this network.

```
$ docker network rm sample-network
```

Example: We will create a static web app and deploy in nginx serve and will connect with a user-defined bridge network ‘sample_net’.

Step1: create a network ‘sample-net’



A screenshot of a Linux terminal window titled "Terminal". The window shows a root shell session. The user has run the command "docker network create sample-net" followed by "docker network ls". The output lists several Docker networks:

NAME	DRIVER	SCOPE
01336fb95eb	bridge	local
b7490031351a	dockercompose_default	bridge
fc77a7c7c15	host	local
00348da2e43e	none	null
0dfaef9d41575	pythonapp_default	bridge
f8fd05068eff	sample-net	bridge

Step2: Create a Dockerfile to create a custom image;

```
FROM nginx:alpine
COPY . /usr/share/nginx/html
EXPOSE 80
```

Next, build this Dockerfile.

```
$ docker build -t nginx_app .
```

Step3: Now we will start this image 'nginx_app' in a container and connect with the user defined network 'sample-net'.

```

Activities Terminal Help
root@anay:/home/anay/Desktop/ABC Learning/nginx_app# ls
Dockerfile index.html
root@anay:/home/anay/Desktop/ABC Learning/nginx_app# docker build -t ngnx_app .
Sending build context to Docker daemon 3.584kB
Step 1/3 : FROM nginx:alpine
alpine: Pulling from library/nginx
c67f3896b22c: Pull complete
89df19b7dc5c: Pull complete
c34d3c110d67: Pull complete
9acbb8076c62: Pull complete
Digest: sha256:829a63ad2b1389e393e5defcf5df25860347d09643c335d1dc3d91d25326d3067
Status: Downloaded newer image for nginx:alpine
--> 99a032453556
Step 2/3 : COPY . /usr/share/nginx/html
--> 36f8bdcc478b8
Step 3/3 : EXPOSE 80
--> Running in c5a253c239a6
Removing intermediate container c5a253c239a6
--> 87be4bdd7e6a
Successfully built 87be4bdd7e6a
Successfully tagged ngnx_app:latest
root@anay:/home/anay/Desktop/ABC Learning/nginx_app# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
ngnx_app latest 87be4bdd7e6a 9 seconds ago 17.4MB
pythonapp_web_dev latest 815341561364 About an hour ago 138MB
<none> <none> 1f2f37a5fee9 About an hour ago 138MB
<none> <none> dd6ba32fb3d0 About an hour ago 138MB
<none> <none> 830cc117b21e About an hour ago 138MB
pythonapp_web_app latest 919633c521de 2 hours ago 147MB
<none> <none> 8066c05c29cf 2 hours ago 147MB
mongo latest c1d6c06b5775 2 days ago 381MB
nginx alpine 994032453556 11 days ago 17.4MB
python 2.7-slim c9cde4658340 2 weeks ago 120MB
orchardup/mysql latest 90606cb24f0e 4 years ago 292MB
root@anay:/home/anay/Desktop/ABC Learning/nginx_app# 

```

root@anay:/home/anay/Desktop/ABC Learning/nginx_app# docker run -d -p 8090:80 --name sampl_nginx_container --network sample-net ngnx_app

2b04c2665c28d2ccdde14071fc3672a1b1f78c5d6af22acb39c88010e600c6d4

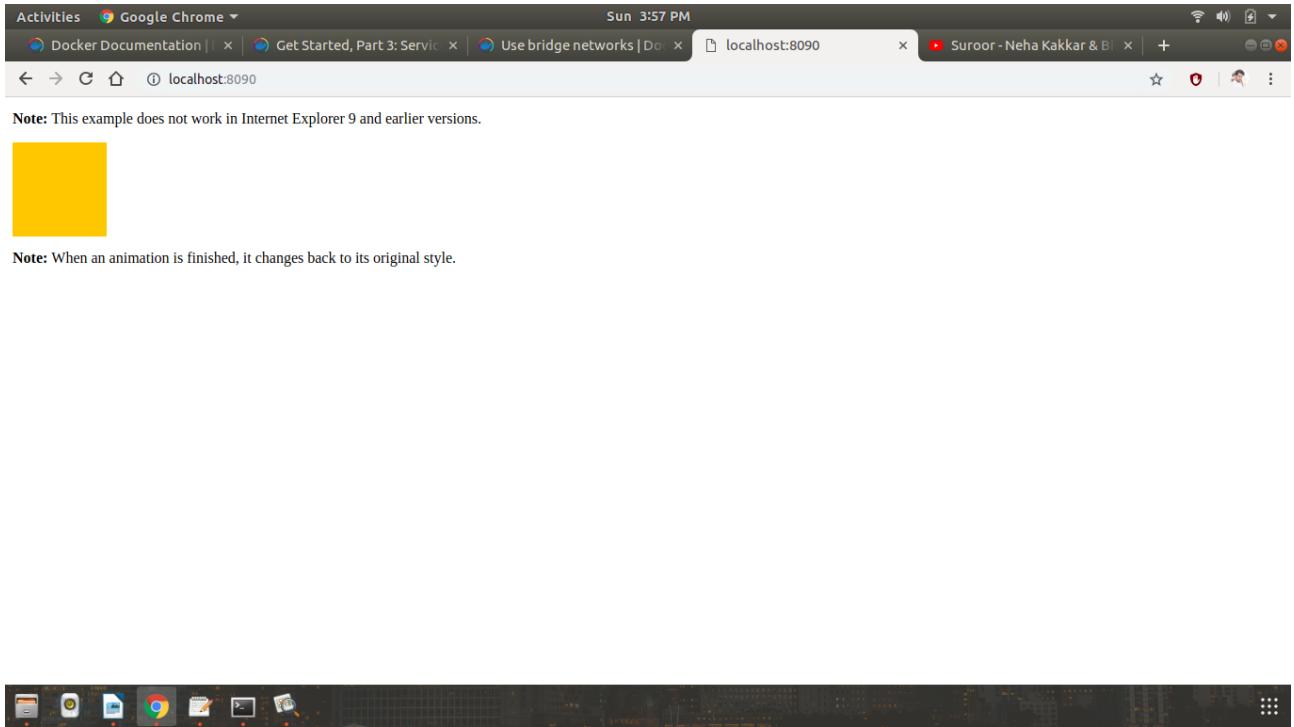
Next , open your browser and enter your url <http://localhost:8090>

you can also verify:

```

Activities Terminal Help
root@anay:/home/anay/Desktop/ABC Learning/nginx_app# 
}, "SandboxKey": "/var/run/docker/netns/ffa47d46da56", "SecondaryIPAddresses": null, "SecondaryIPv6Addresses": null, "EndpointID": "", "Gateway": "", "GlobalIPv6Address": "", "GlobalIPv6PrefixLen": 0, "IPAddress": "", "IPPrefixLen": 0, "IPv6Gateway": "", "MacAddress": "", "Networks": { "sample-net": { "IPAMConfig": null, "Links": null, "Aliases": [ "2b04c2665c28" ], "NetworkID": "f8fd0506eff0109b1b317e38f894685881bb3f7f32adc952f3a3afde2bc923d", "EndpointID": "fe518081ce46e27ffe09ce53a67ed069c7ff4d9becaa43b936d4420eb2feb8b2", "Gateway": "172.20.0.1", "IPAddress": "172.20.0.2", "IPPrefixLen": 16, "IPv6Gateway": "", "GlobalIPv6Address": "", "GlobalIPv6PrefixLen": 0, "MacAddress": "02:42:ac:14:00:02", "DriverOpts": null } } }
]
root@anay:/home/anay/Desktop/ABC Learning/nginx_app# 

```



index.html page:

```
<!DOCTYPE html>

<html>
<head>
<style>
div {
    width: 100px;
    height: 100px;
    background-color: red;
    -webkit-animation-name: example; /* Safari 4.0 - 8.0 */
    -webkit-animation-duration: 4s; /* Safari 4.0 - 8.0 */
    animation-name: example;
    animation-duration: 4s;
}
/* Safari 4.0 - 8.0 */

@-webkit-keyframes example {
```

```
from {background-color: red;}  
to {background-color: yellow;}  
}  
  
/* Standard syntax */  
  
@keyframes example {  
from {background-color: red;}  
to {background-color: yellow;}  
}  
  
</style>  
</head>  
<body>  
<p><b>Note:</b> This example does not work in Internet Explorer 9 and earlier versions.</p>  
<div></div>  
<p><b>Note:</b> When an animation is finished, it changes back to its original style.</p>  
</body>  
</html>
```

Problem:Docker swarm implementation and load balancer example.

Before we start, lets understand some concepts.

Swarm cluster:

A swarm is a group of machines that are running Docker and joined into a cluster. After we execute swarm command , we can continue to run the docker commands but the only difference will be that after that the container will start running on a cluster by a swarm manager.

The machines in a swarm can be physical or virtual. And after joining a swarm, they are referred to as nodes.

Note: Enabling swarm mode instantly makes the current machine a swarm manager.

How to enable swarm?

Basically a swarm is made up of multiple nodes, which can be anything physical or virtual machines. To enable swarm we have to execute the following command as given below:

```
$ docker swarm init
```

We can join another machines in swarm as workers by runing the command as given below:

```
$ docker swarm join
```

Docker Machine:

The Docker machine can create and tear down whole fleets of Docker enabled hosts in a second.

This Docker machine ships with a numbers of drivers which integrates Docker machine with a different virtual machines technology or cloud based virtual computing providers.

Step1: We will create three docker hosts by using a driver virtualbox.

We can create a host using the command given below:

```
$ docker-machine create --driver virtualbox host1
```

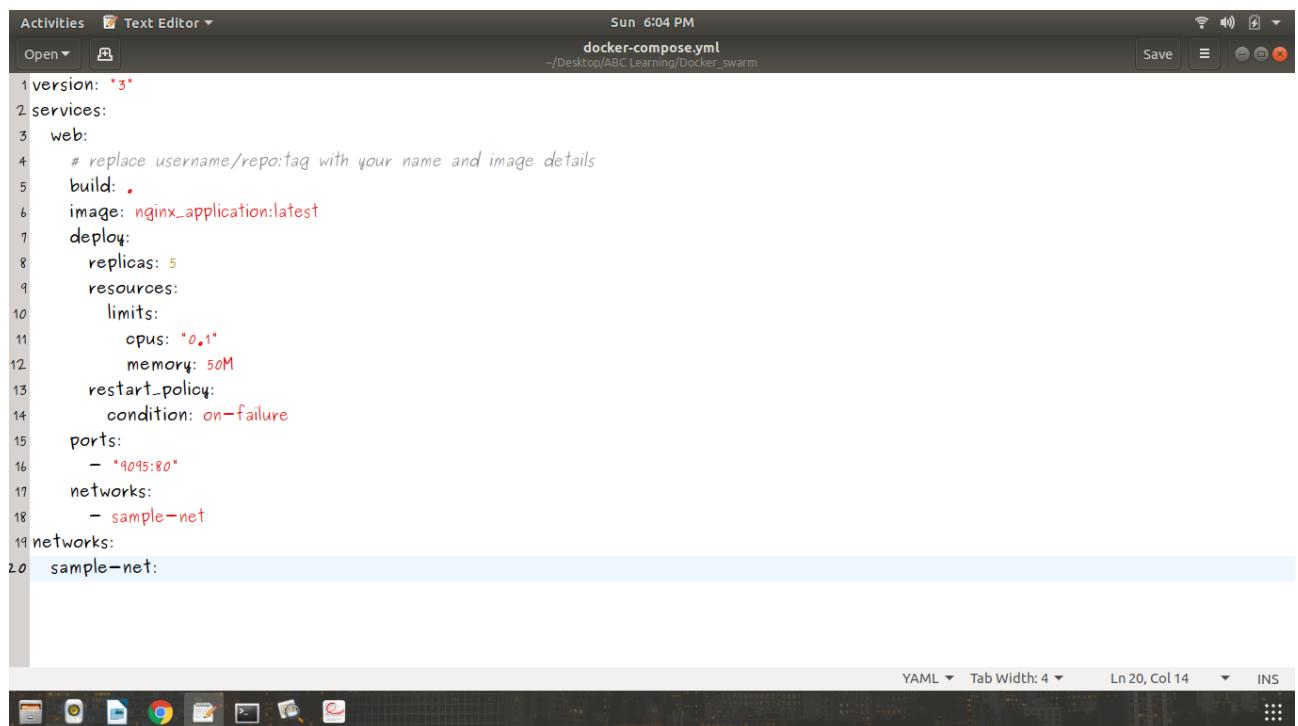
To install Docker-machine run this command:

```
root@anay:~# base=https://github.com/docker/machine/releases/download/v0.14.0 &&
> curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&
> sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```

Now Let's take an example:

We will deploy a simple static web app which is running on nginx.

Step1: create a docker-compose.yml file and Dockerfile , index.html is same as we used in pervious examples.



A screenshot of a Linux desktop environment showing a terminal window. The terminal title is "Text Editor". The window contains a YAML configuration file named "docker-compose.yml" located at "/Desktop/ABC Learning/Docker_swarm". The file defines a single service named "web" with the following specifications:

```
version: '3'
services:
  web:
    # replace username/repo:tag with your name and image details
    build: .
    image: nginx_application:latest
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "9095:80"
    networks:
      - sample-net
networks:
  sample-net:
```

The "sample-net" network definition is highlighted with a light blue background. The terminal interface includes standard Linux desktop icons in the dock at the bottom.

Now, let's create three physical virtual machine using virtualbox driver.

```
Activities Terminal Sun 6:10 PM
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# docker-machine create --driver virtualbox host1
Running pre-create checks...
(host1) Image cache directory does not exist, creating it at /root/.docker/machine/cache...
(host1) No default Boot2Docker ISO found locally, downloading the latest release...
(host1) Latest release for github.com/boot2docker/boot2docker is v18.06.1-ce
(host1) Downloading /root/.docker/machine/cache/boot2docker.iso from https://github.com/boot2docker/boot2docker/releases/download/v18.06.1-ce/boot2docker.iso...
(host1) 0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Creating machine...
(host1) Copying /root/.docker/machine/cache/boot2docker.iso to /root/.docker/machine/machines/host1/boot2docker.iso...
(host1) Creating VirtualBox VM...
(host1) Creating SSH key...
(host1) Starting the VM...
(host1) Check network to re-create if needed...
(host1) Found a new host-only adapter: "vboxnet0"
(host1) Waiting for an IP...
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: docker-machine env host1
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm#
```

In the same way create two more machines.

And you can verify that they have created or not using a command given below:

```
$ docker-machine ls
```

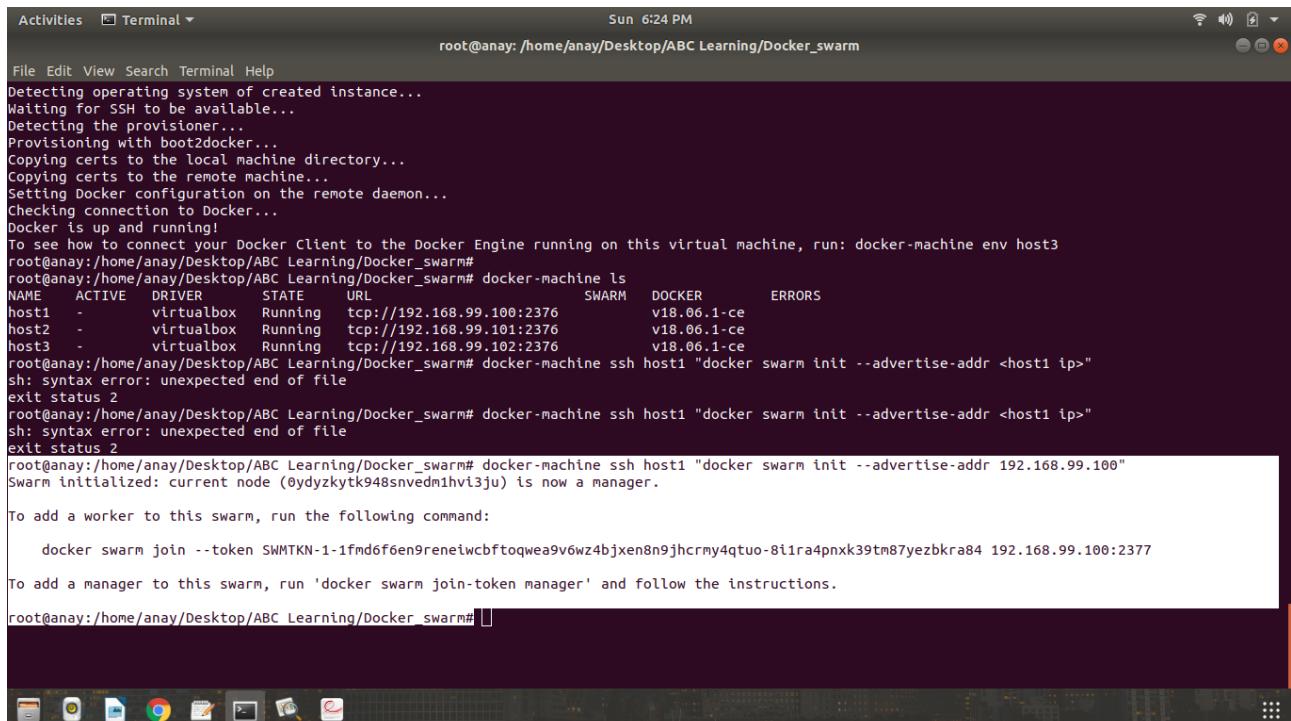
```
Activities Terminal Sun 6:15 PM
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# docker-machine ls
File Edit View Search Terminal Help
NAME ACTIVE DRIVER STATE URL SWARM DOCKER ERRORS
host1 - virtualbox Running tcp://192.168.99.100:2376 v18.06.1-ce
host2 - virtualbox Running tcp://192.168.99.101:2376 v18.06.1-ce
host3 - virtualbox Running tcp://192.168.99.102:2376 v18.06.1-ce
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm#
```

Step2: Now initialize the swarm and add nodes.

The first machine acts as the manager, which will execute management commands and authenticates workers to join the swarm.

Here we will send a command to a machine using a command given below:

```
$ docker-machine ssh <machinename> "<commands>"
```



```
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# docker-machine ssh host1 "docker swarm init --advertise-addr <host1 ip>"  
sh: syntax error: unexpected end of file  
exit status 2  
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# docker-machine ssh host1 "docker swarm init --advertise-addr <host1 ip>"  
sh: syntax error: unexpected end of file  
exit status 2  
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# docker-machine ssh host1 "docker swarm init --advertise-addr 192.168.99.100"  
Swarm initialized: current node (0ydyzkytk948snvedm1hvl3ju) is now a manager.  
To add a worker to this swarm, run the following command:  
  docker swarm join --token SWMTKN-1-1fmd6f6en9reneiwcftoqwea9v6wz4bjxen8n9jhcrmy4q tuo-8i1ra4pxk39tm87yezbkra84 192.168.99.100:2377  
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.  
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm#
```

In this step, we have started swarm and add host1 as manager.

Next, we will add host2 in swarm,

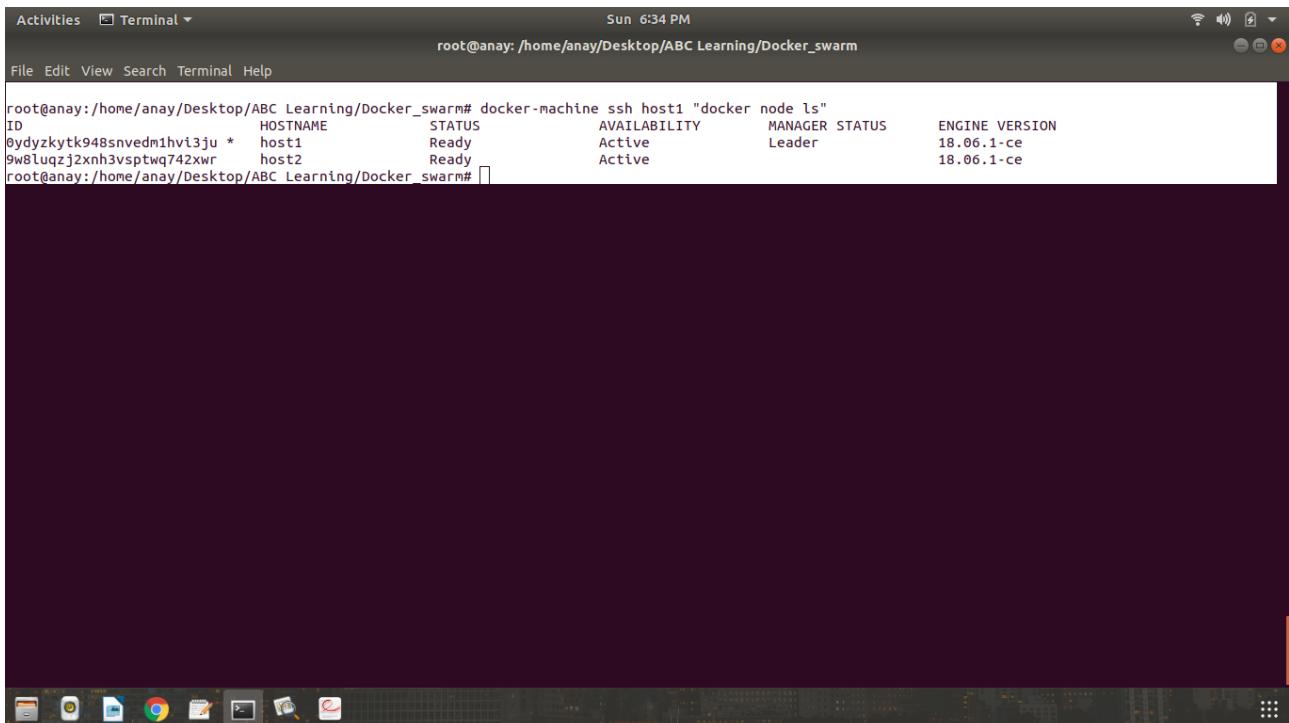
```
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# docker-machine ssh host2  
"docker      swarm      join      --token      SWMTKN-1-  
1fmd6f6en9reneiwcftoqwea9v6wz4bjxen8n9jhcrmy4q tuo-8i1ra4pxk39tm87yezbkra84  
192.168.99.100:2377"
```

This node joined a swarm as a worker.

```
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm#
```

Step3: Now we will verify that host2 acts as a node for host1 or not. To do this we need to run a command as given below:

```
$docker-machine ssh <machine_name> "docker node ls"
```



A screenshot of a Linux desktop environment showing a terminal window. The terminal title is "root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm". The command "docker node ls" has been run, displaying the following table:

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
0ydyzkytk948snvedm1hvi3ju *	host1	Ready	Active	Leader	18.06.1-ce
9w8Luqzj2xnh3vsptwg742xwr	host2	Ready	Active		18.06.1-ce

Step4: we will deploy our application.

Now hard part is over and we will deploy our application in swarm.

So far , we have been using docker commands in docker-machine ssh to talk to the virtual machines. There is one more option to run docker-machine env <machine_name> to get and run a command that configures our current shell to talk to Docker daemon on the VM.

```
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# docker-machine env host1

export DOCKER_TLS_VERIFY="1"

export DOCKER_HOST="tcp://192.168.99.100:2376"

export DOCKER_CERT_PATH="/root/.docker/machine/machines/host1"

export DOCKER_MACHINE_NAME="host1"

# Run this command to configure your shell:

# eval $(docker-machine env host1)
```

So as it is given that we need to run eval \$(docker-machine env host1) to configure our shell to talk to host1.

```
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# eval $(docker-machine env host1)

root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# docker-machine ls

NAME ACTIVE DRIVER STATE URL SWARM
DOCKER ERRORS

host1 * virtualbox Running tcp://192.168.99.100:2376 v18.06.1-ce
host2 - virtualbox Running tcp://192.168.99.101:2376 v18.06.1-ce
host3 - virtualbox Running tcp://192.168.99.102:2376 v18.06.1-ce
```

As we can see * to left of host1 which indicates that host1 is active.

Netx, we will deploy ur application.

```
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# docker stack deploy -c docker-compose.yml my_nginx_demo

Creating network my_nginx_demo_sample-net
Creating service my_nginx_demo_web

root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm#
```

Now, to check what are the services running as given below:

Activities Terminal Help

Sun 6:56 PM root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm

```
Creating service my_nginx_demo_web
root@anay:/home/anay/Desktop/ABC Learning/Docker_swarm# docker stack ps my_nginx_demo
ID          NAME      IMAGE           NODE      DESIRED STATE   CURRENT STATE     ERROR
y0qv3ucow7mt  my_nginx_demo_web.1  nginx_app:latest  host1    Running        Assigned 6 seconds ago
b54pkwibcl7h  \_ my_nginx_demo_web.1  nginx_app:latest  host1    Shutdown       Rejected 9 seconds ago  "No such i
image: nginx_app:latest"
fuynya36cmi2  \_ my_nginx_demo_web.1  nginx_app:latest  host2    Shutdown       Rejected 36 seconds ago  "No such i
image: nginx_app:latest"
sizjoqzomtx  \_ my_nginx_demo_web.1  nginx_app:latest  host2    Shutdown       Rejected 57 seconds ago  "No such i
image: nginx_app:latest"
toid0eoey3   \_ my_nginx_demo_web.1  nginx_app:latest  host1    Shutdown       Rejected about a minute ago  "No such i
image: nginx_app:latest"
rg4p6hsrl7tx  my_nginx_demo_web.2  nginx_app:latest  host1    Running        Preparing 15 seconds ago
ilxajxh0poppt \_ my_nginx_demo_web.2  nginx_app:latest  host2    Shutdown       Rejected 37 seconds ago  "No such i
image: nginx_app:latest"
rvmk5htger9m  \_ my_nginx_demo_web.2  nginx_app:latest  host1    Shutdown       Rejected about a minute ago  "No such i
image: nginx_app:latest"
pmf9i3u697vt  \_ my_nginx_demo_web.2  nginx_app:latest  host1    Shutdown       Rejected about a minute ago  "No such i
image: nginx_app:latest"
ygowte8odqhi  \_ my_nginx_demo_web.2  nginx_app:latest  host1    Shutdown       Rejected 2 minutes ago  "No such i
image: nginx_app:latest"
onxvozyv6scn  my_nginx_demo_web.3  nginx_app:latest  host1    Running        Preparing 17 seconds ago
mbk2cvbjju6w4  \_ my_nginx_demo_web.3  nginx_app:latest  host2    Shutdown       Rejected 23 seconds ago  "No such i
image: nginx_app:latest"
q0e5mnz5ecy7  \_ my_nginx_demo_web.3  nginx_app:latest  host1    Shutdown       Rejected 38 seconds ago  "No such i
image: nginx_app:latest"
pm4zj2qraiāz  \_ my_nginx_demo_web.3  nginx_app:latest  host2    Shutdown       Rejected 55 seconds ago  "No such i
image: nginx_app:latest"
35bpncildlo8  \_ my_nginx_demo_web.3  nginx_app:latest  host1    Shutdown       Rejected about a minute ago  "No such i
56f3svkj3i2   my_nginx_demo_web.4  nginx_app:latest  host1    Running        Assigned 17 seconds ago
```


Now, you can see this application. Open your browser and enter the IP of host1 and we can see the same page as we have seen in previous example.

Problem: Execute a script or application using docker exec command

The docker exec command provides the needed help to users, who are trying to deploy their application which is running in background.

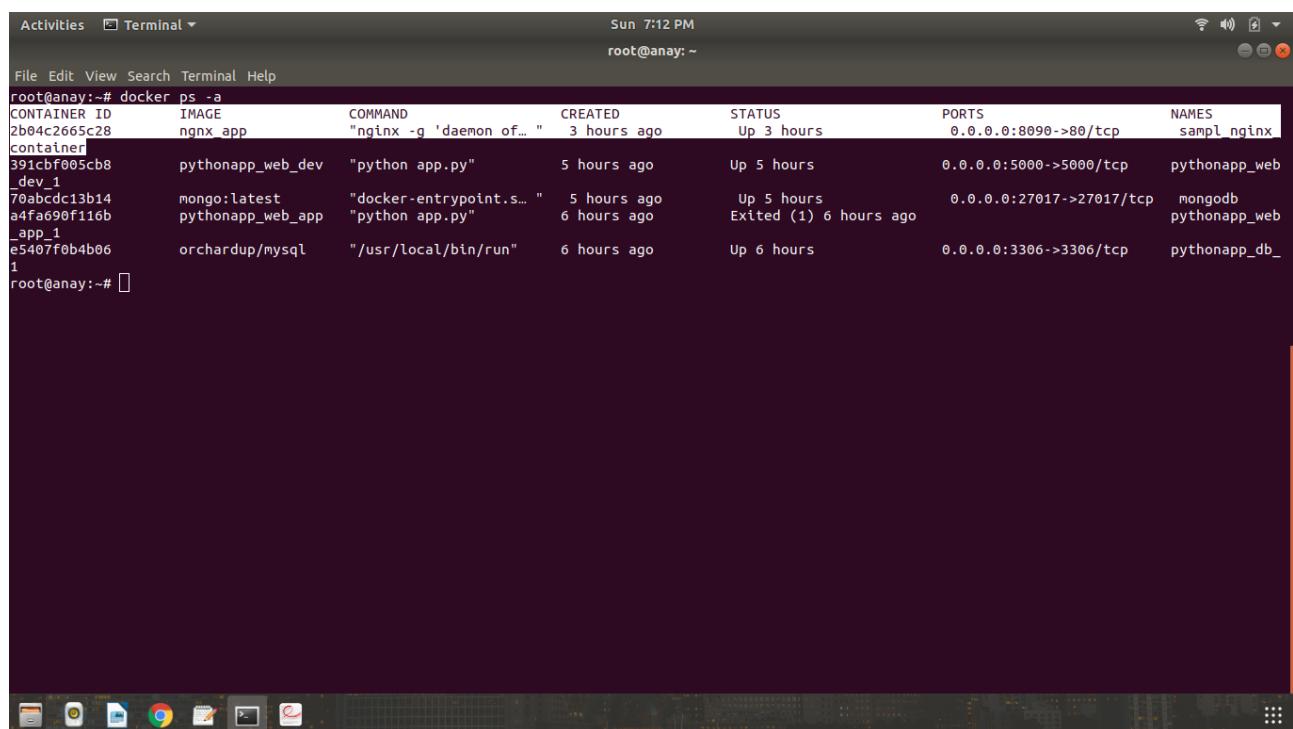
Because of this command, it is not necessary to log in to runn the SSH daemon in the container.

NOTE: docker exec command can only access the running containers. This command spawns a new process in the target container using Docker API and CLI. To check that we can run pe - aef command inside the target container.

Example:

We have some container like nginx_app which we have created in VOLUME section.

Type docker ps -a.



```
root@anay:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                 NAMES
2b04c2665c28        nginx              "nginx -g 'daemon off;"   3 hours ago        Up 3 hours         0.0.0.0:8090->80/tcp   sampl_nginx
391cbf005cb8        pythonapp_web_dev  "python app.py"          5 hours ago       Up 5 hours         0.0.0.0:5000->5000/tcp  pythonapp_web
70abcccd13b14        mongo:latest      "docker-entrypoint.s..."  5 hours ago       Up 5 hours         0.0.0.0:27017->27017/tcp  mongodb
a4fa696ff116b        pythonapp_web_app  "python app.py"          6 hours ago       Exited (1) 6 hours ago   pythonapp_web
e5407f0b4b06        orchardup/mysql    "/usr/local/bin/run"     6 hours ago       Up 6 hours         0.0.0.0:3306->3306/tcp  pythonapp_db_1
root@anay:~#
```

Now, we will run docker exec command and will enter inside the container.

A screenshot of a Linux desktop environment showing a terminal window. The terminal title is "Terminal" and the status bar shows "Sun 7:14 PM" and "root@anay: ~". The terminal content is as follows:

```
File Edit View Search Terminal Help
root@anay:~# docker exec -it sampl_nginx_container \sh
/ # ls
bin  dev  etc  home  lib  media  mnt  proc  root  run  sbin  srv  sys  tmp  usr  var
/ # 
```

As you can see we entered inside the container.