# Docker: Enterprise Container Platform

Abdul Rehman Tariq

**Table of Contents**

[Docker: Enterprise Container Platform 1](#_Toc529528367)

[1. Introduction 4](#_Toc529528368)

[1.1 Docker Engine 4](#_Toc529528369)

[2. How to install Dockers on your machine 5](#_Toc529528370)

[3. Run Hello world 6](#_Toc529528371)

[4. Docker Objects 8](#_Toc529528372)

[4.1 Images 8](#_Toc529528373)

[4.2 Container 8](#_Toc529528374)

[5. Define a container includes these files below 9](#_Toc529528375)

[5.1 Dockerfile 9](#_Toc529528376)

[5.2 Requirements.txt 12](#_Toc529528377)

[5.3 App.py 12](#_Toc529528378)

[6. How to build the app 14](#_Toc529528379)

[7. How to run the app 14](#_Toc529528380)

[8. How to share your image 15](#_Toc529528381)

[9. How to tag and publish the image 15](#_Toc529528382)

[10. How to pull and run the image from remote repository 16](#_Toc529528383)

[11. Services in docker 16](#_Toc529528384)

[12. Swarm 16](#_Toc529528385)

[13. Stack 17](#_Toc529528386)

[14. Docker hub 17](#_Toc529528387)

**List of Figures**

Figure 1: Docker Information……………………………………………………………………………………………………3

Figure 2: Message Hello World…………………………………………………………………………………………………4

Figure 3: Listed Images…………………………………………………………………………………………………………....4

Figure 4: Listed Containers……………………………………………………………………………………………………….5

Figure 5: Existing Directories…………………………………………………………………………………………………….6

Figure 6: Directory path after Changed…………………………………………………………………………………….6

Figure 7: New file command…………………………………………………………………………………………………....7

Figure 8: New file editor…………………………………………………………………………………………………………..7

Figure 9: File with code…………………………………………………………………………………………………………….8

Figure 10: Save and exit command…………………………………………………………………………………………..8

Figure 11: Encryption key…………………………………………………………………………………………………………9

Figure 12: New File……………………………………………………………………………………………………….............9

Figure 13: New file……………………………………………………………………………………………………................9

Figure 14: All files in existing directory……………………………………………………………………………………10

Figure 15: Build the app………………………………………………………………………………………………………….11

Figure 16: Run the app……………………………………………………………………………………………………........11

Figure 17: Web page………………………………………………………………………………………………………………12

Figure 18: Login……………………………………………………………………………………………………………………..12

Figure 19: Tag name………………………………………………………………………………………………………….……12

Figure 20: Docker hub repositories page………………………………………………………………………….…….13

# Introduction

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.With Docker, we can manage your infrastructure in the same ways you manage your applications.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don’t need the extra load of a hypervisor, but run directly within the host machine’s kernel.

# Docker Engine

Docker Engine is a client-server application with these major components:

* A server which is a type of long-running program called a daemon process (the dockerd command).
* A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
* A command line interface (CLI) client (the docker command).



# How to install Dockers on your machine

**Step 1: Install Docker**

First install a latest stable version of Docker from the Docker community and also can download from this URL https://store.docker.com/editions/community/docker-ce-desktop-windows.

**Step 2: Install Git**

Next install GitBash for command line. I recommended GitBash instead of default windows cmd.

**Step 3: Test your Docker version**

You can check your version of Docker in command line by use this command.

docker --version (This command is used for Docker version checking)

The result produced by this command is below.

Docker version 17.12.0-ce, build c97c6d6

To view more details of Docker installation you can use below command.

docker info

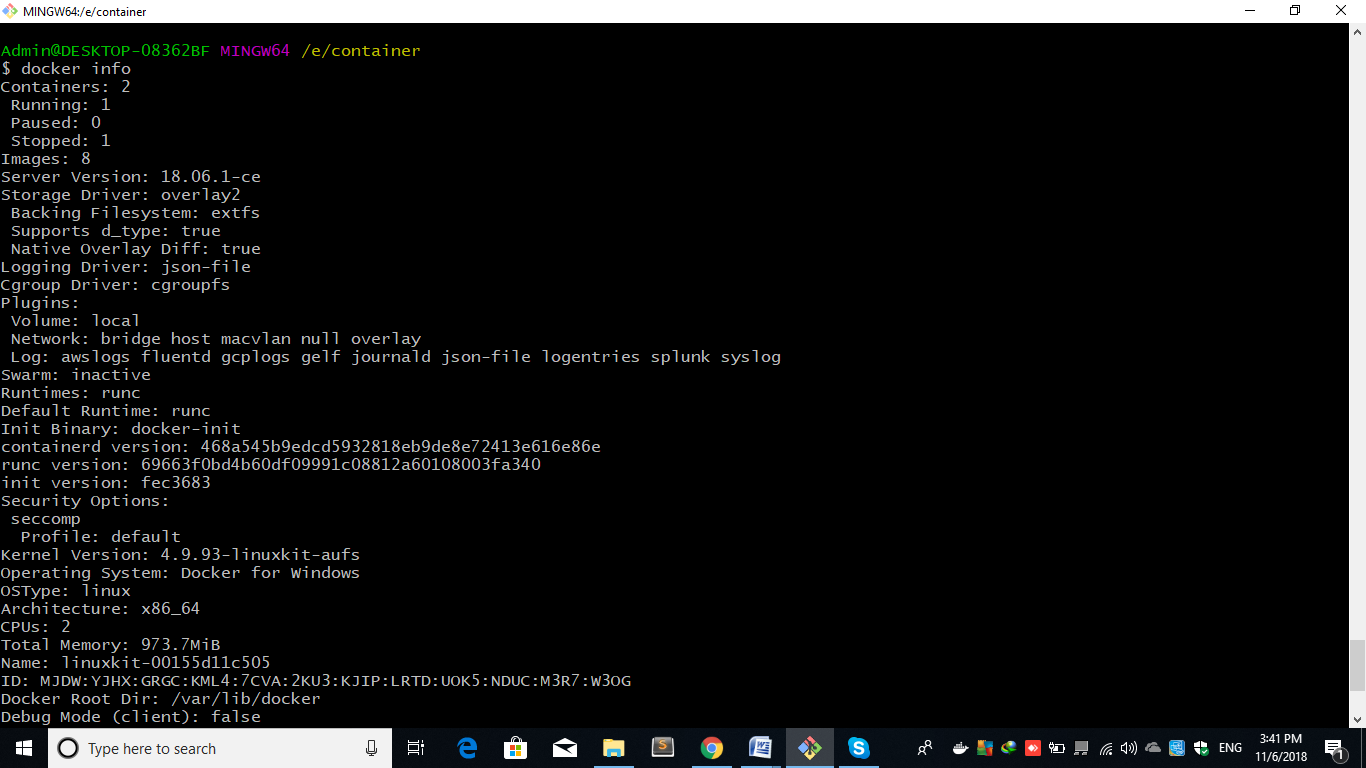
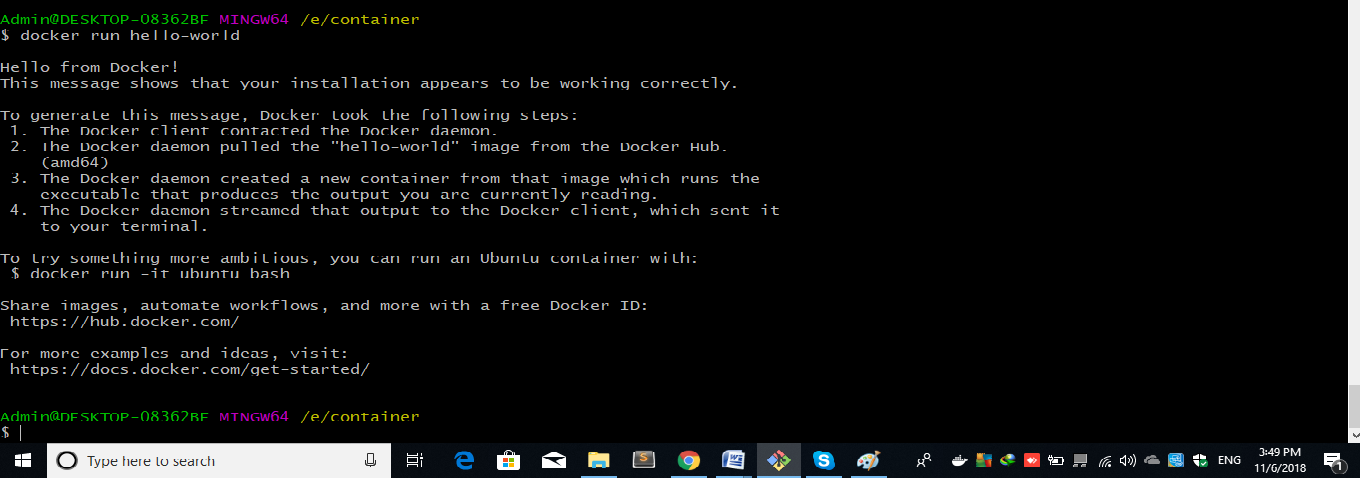
**Figure 1: Docker Information**

Figure 1 shows the result of command 🡪 docker info

# Run Hello world

To run your first hello world by image using this command below:

docker run hello-world (This Command is use for run the image)



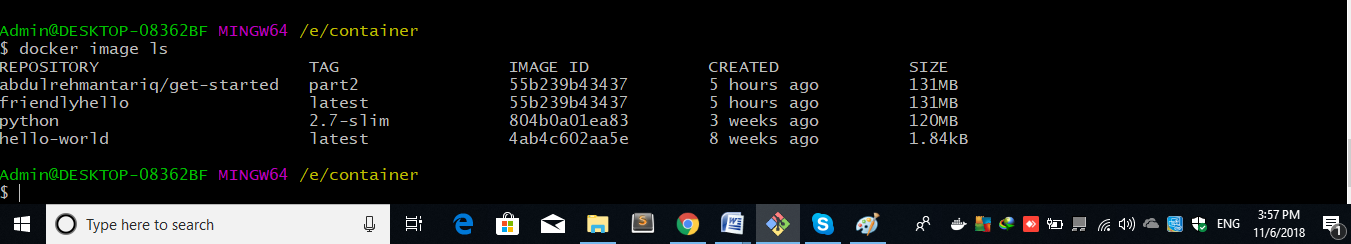
**Figure 2: Message Hello World**

Screen 2 shows the result of command 🡪 docker run hello-world

* 1. **Check your all listed images**

You can check your all list of images by using this command below:

docker image ls

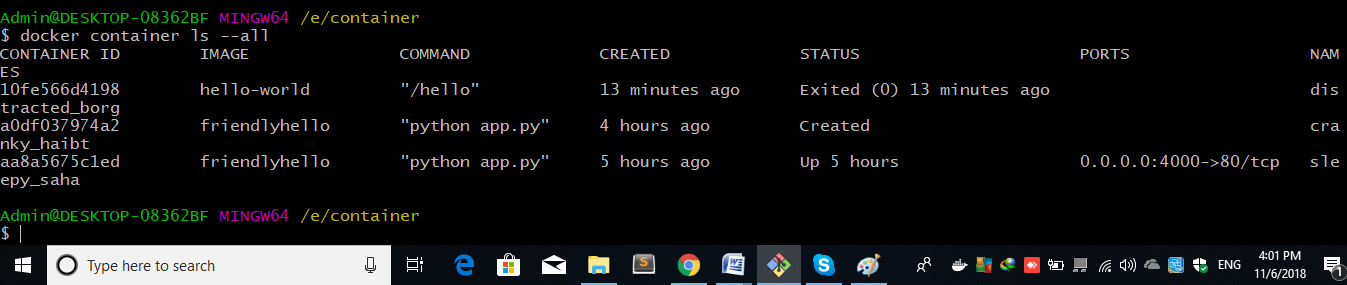
**Figure 3: Listed Images**

Screen 3 shows the result of command 🡪 docker image ls

* 1. **Check your all listed container**

You can also check your all list of container by using this command below:

docker container ls–all



**Figure 4: Listed Containers**

Screen 4 shows the result of command 🡪 docker container ls –all

**Below the commands using for Docker initial steps:**

# Display Docker version and info

docker--version

docker info

# Run Docker image

docker run hello-world

# List Docker images

docker image ls

# List Docker containers (running, all)

docker container ls

docker container ls--all

# Docker Objects

# Images

An image is a read-only template with instructions for creating a Docker container. An Image contains all the dependencies, libraries, and the application itself. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run. You might create your own images or you might only use those created by others and published in a registry.

# Container

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state. A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

# Define a container includes these files below

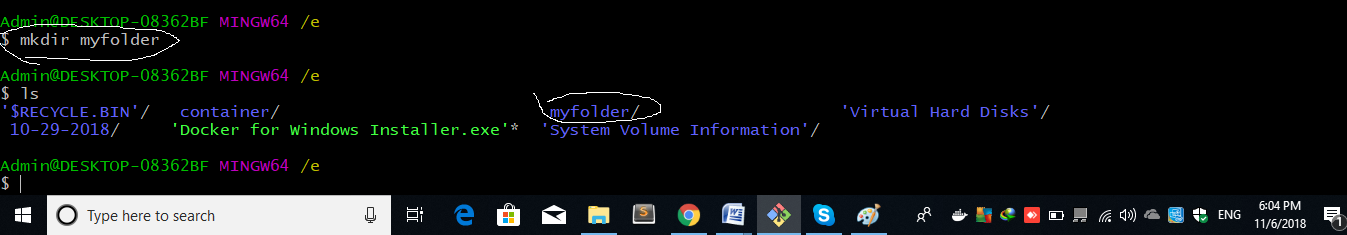
# Dockerfile

First create an empty folder in your disk by using this command below

mkdir Your Folder Name (This command is used for create new folder)

You can see and confirmed your existing folder by using this command below

ls (This command is used for seen the folders inside directory)



**Figure 5: Existing Directories**

Second move your current directory into your new make directory by using this command below:

cd your directory name (This command is used changed the directory)



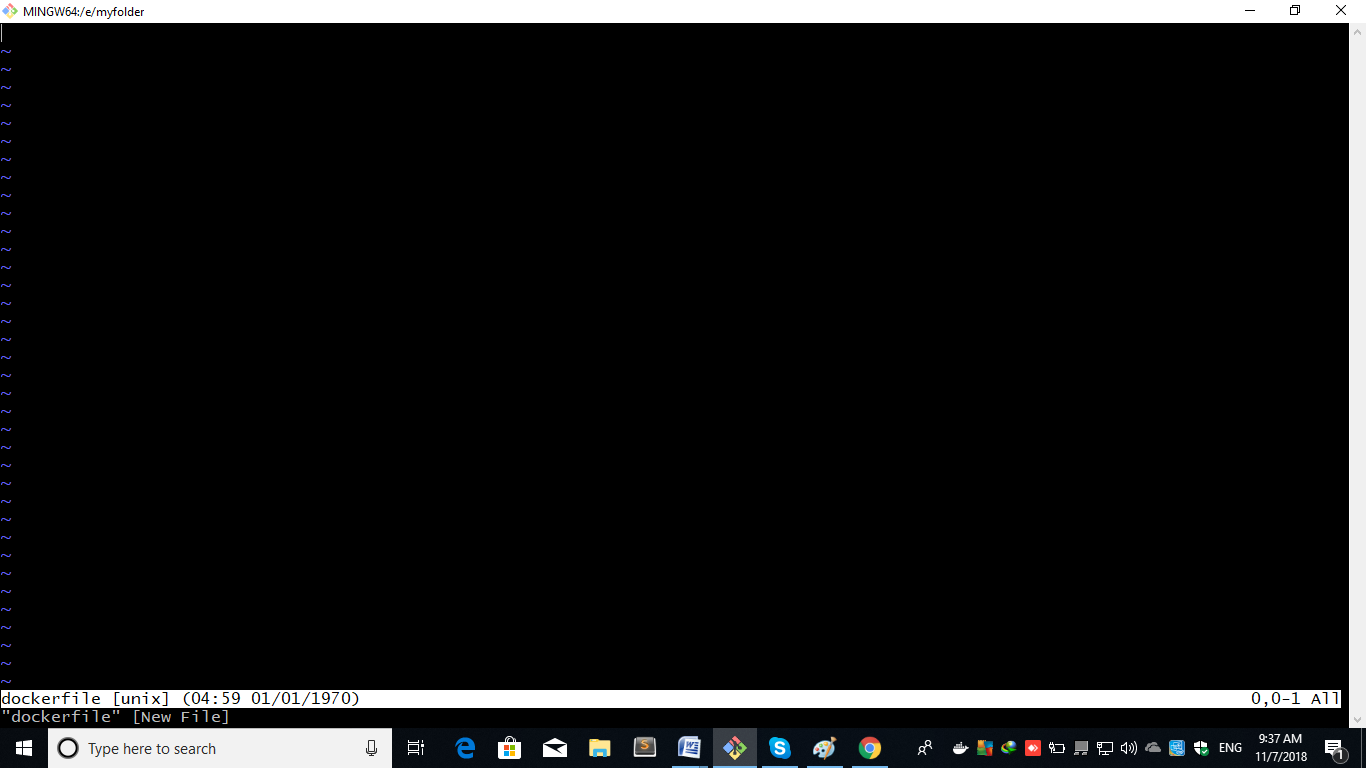
**Figure 6: Directory path after Changed**

Third make a new file in your new make directory as per instruction below and add all the references and libraries of python then save the file with the name dockerfile.

vim file name and press enter(This command is used for create and edit the file)

**Figure 7: New file command**

**After press enter it will show you this screen (figure 8)**

****

**Figure 8: New file editor**

In this screen copy paste the code below

# 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

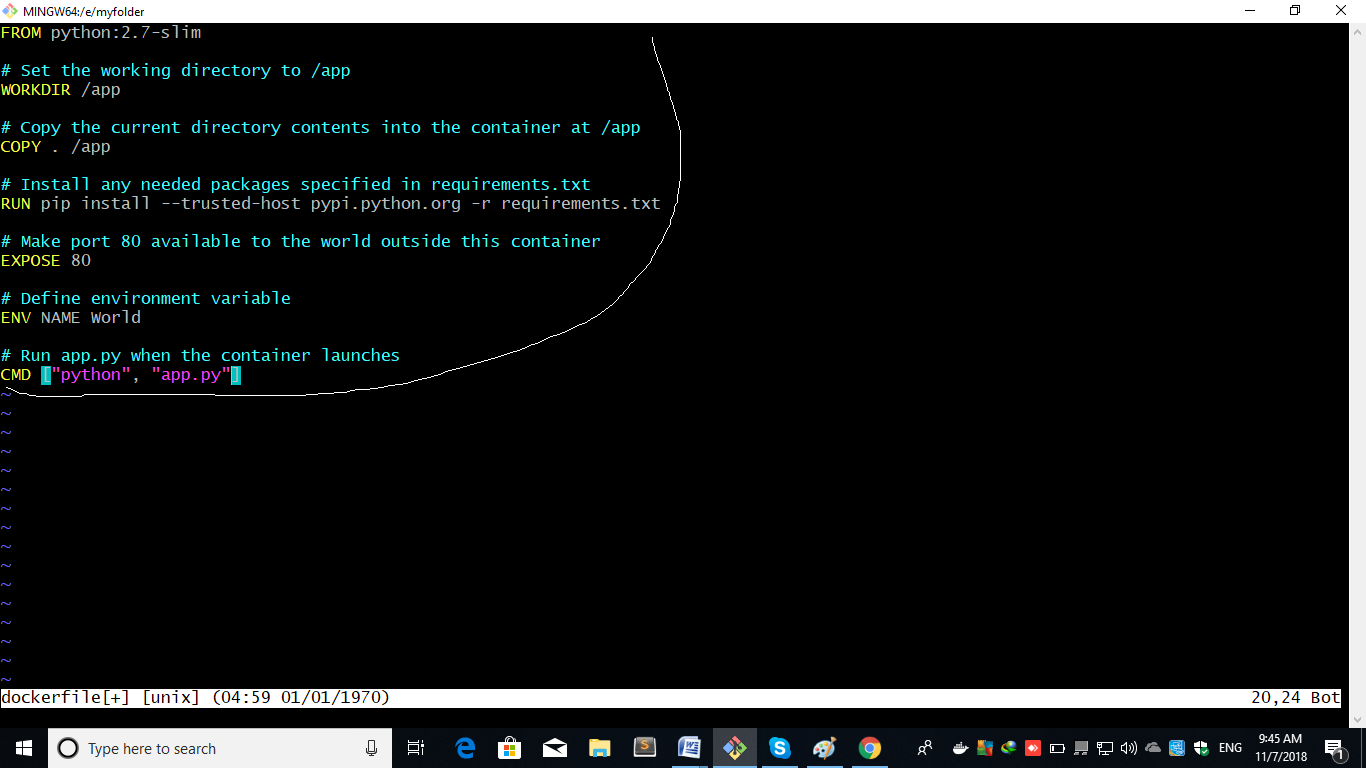
EXPOSE 80

# Define environment variable

ENV NAME World

# Run app.py when the container launches

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



**Figure 9: File with code**

After the paste press esc and then enter **:x** you can see in Figure 10



**Figure 10: Save and exit command**

After press if it will ask you for encryption key then enter any 5 digits or more numeric key like below

****

**Figure 11: Encryption key**

**If you are till on file edit screen then press again esc and enter :x again.**

# Requirements.txt

Now we create requirement file in the existing directory along docker file.

Again repeat the vim command now this time with the name requirements.txt like below



**Figure 12: New File**

After this press enter it will again move to you edit screen like Figure 8

Now copy and paste the code below

Flask

Redis

**After paste press esc and enter :x it will move back to you previous screen.**

# App.py

Now we create app.py file same like previous processes

Again enter vim command with name app.py and hit enter.

****

**Figure 13: New file**

After enter it will move to you edit screen like **figure 8**

Now copy paste the code below

fromflaskimportFlask

fromredisimportRedis,RedisError

importos

importsocket

# Connect to Redis

redis=Redis(host="redis",db=0,socket\_connect\_timeout=2,socket\_timeout=2)

app=Flask(\_\_name\_\_)

@app.route("/")

defhello():

try:

visits=redis.incr("counter")

exceptRedisError:

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

html="<h3>Hello {name}!</h3>" \

"<b>Hostname:</b> {hostname}<br/>" \

"<b>Visits:</b> {visits}"

returnhtml.format(name=os.getenv("NAME","world"),hostname=socket.gethostname(),visits=visits)

if\_\_name\_\_=="\_\_main\_\_":

app.run(host='0.0.0.0',port=80)

After paste press esc and enter :x like **Figure 10**

**After create successfully these files you can verify that these files are created or not in my directory by using this command below**

$ ls

****

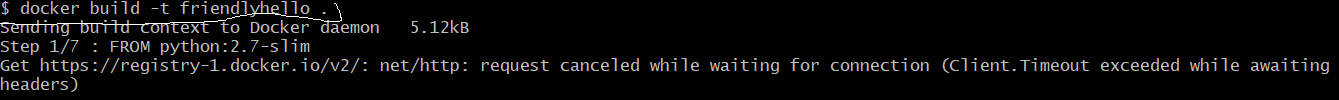
**Figure 14: All files in existing directory**

It will show you all the three files which we are created.

# How to build the app

Now you are able to build the app by using this command below it will creates a docker image and you can also verify this docker image

docker build -t friendlyhello .



**Figure 15: Build the app**

It will take some time be patient

You can verify your image by using this command

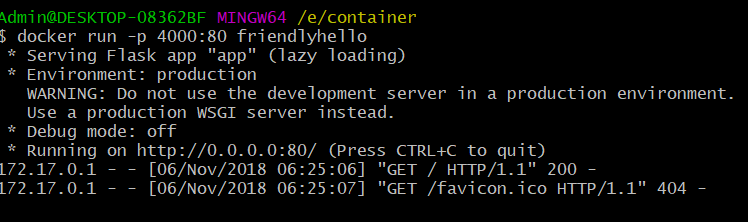
$ docker image ls

# How to run the app

Now we run the app by using this command below

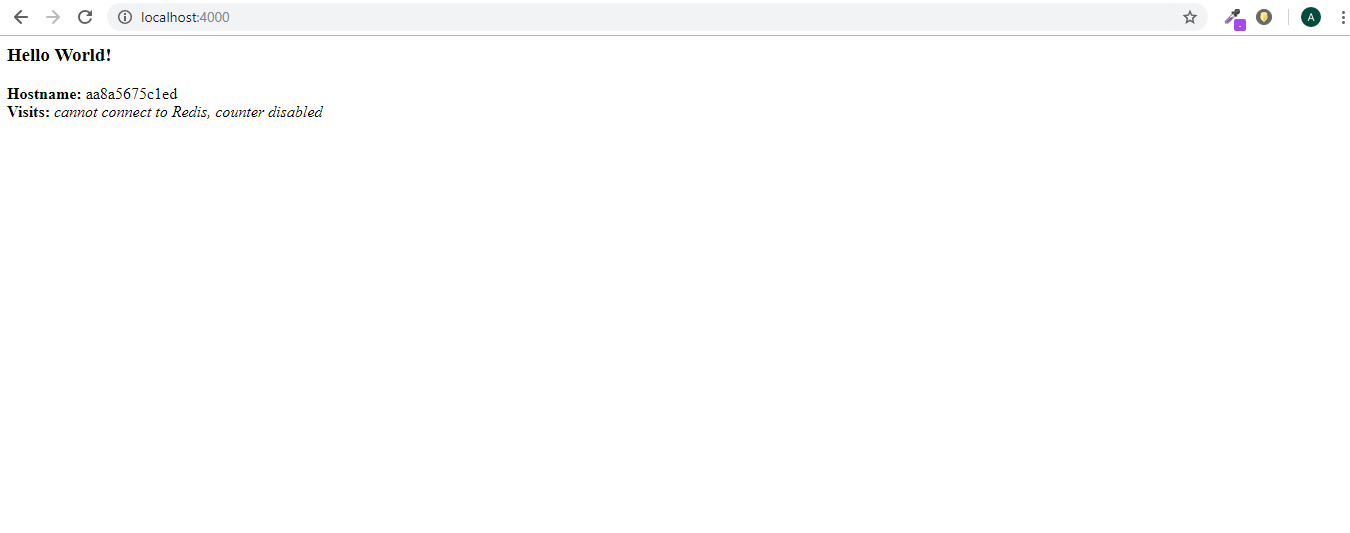
docker run -p 4000:80 friendlyhello

After successfully run it will show you



**Figure 16: Run the app**

Now you can see your app in browser by enter this URL <http://localhost:4000/>

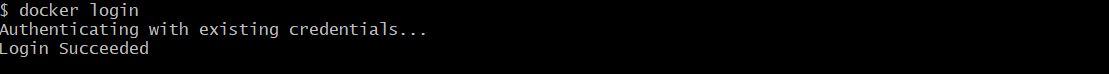


**Figure 17: Web page**

# How to share your image

You can share your image in docker hub by signing in docker account. If you have login credentials then by enter this command you will sign in.

$ docker login



**Figure 18: Login**

# How to tag and publish the image

Now if you want to publish your image docker recommended tag a image by this

docker tag image username/repositoryname:tagname

docker hello.png

**Figure 19: Tag name**

Now you are able to push your image by using this command

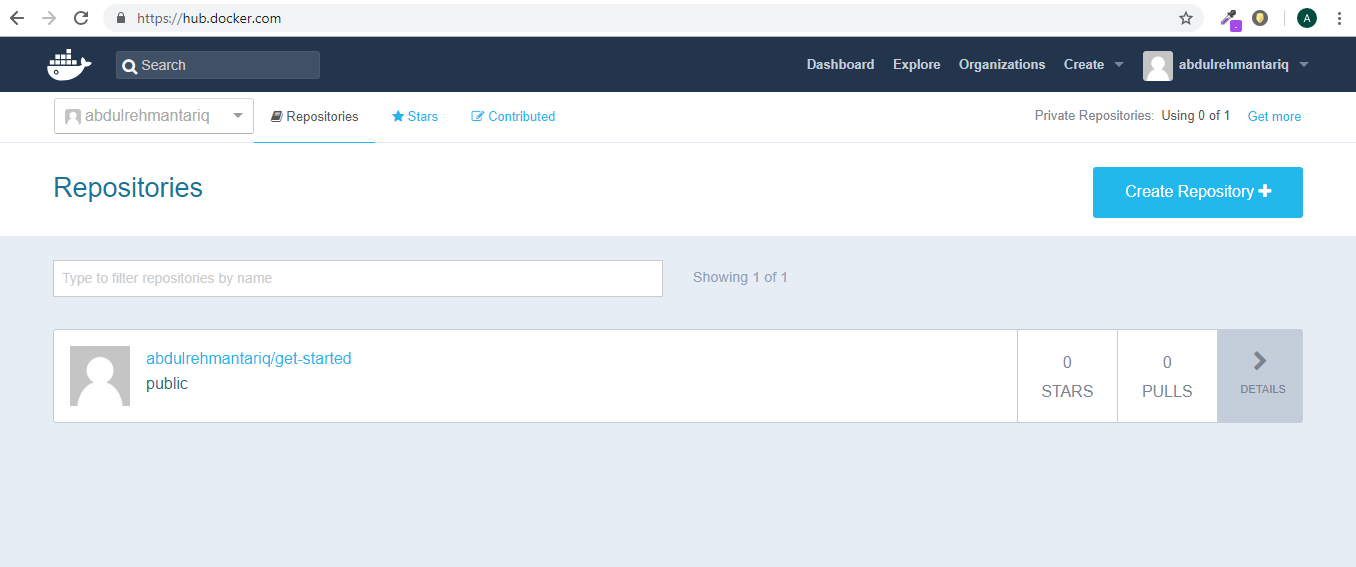
docker push username/repositoryname:tagname

# How to pull and run the image from remote repository

Now you can run your image with any machine by this command

docker run -p 4000:80 username/repositoryname:tagname

And you also verify your publish image in docker hub by signing in



**Figure 20: Docker hub repositories page**

# Services in docker

Services allow you to scale containers across multiple Docker daemons, which all work together as a swarm with multiple managers and workers. Each member of a swarm is a Docker daemon, and the daemons all communicate using the Docker API. A service allows you to define the desired state, such as the number of replicas of the service that must be available at any given time. By default, the service is load-balanced across all worker nodes. To the consumer, the Docker service appears to be a single application. Docker Engine supports swarm mode in Docker 1.12 and higher.

# Swarm

A swarm is a group of machines that are running Docker and joined into a cluster. After that has happened, you continue to run the Docker commands you’re used to, but now they are executed on a cluster by a **swarm manager**. The machines in a swarm can be physical or virtual. After joining a swarm, they are referred to as **nodes**.

# Stack

A stack is a group of interrelated services that share dependencies, and can be orchestrated and scaled together. A single stack is capable of defining and coordinating the functionality of an entire application (though very complex applications may want to use multiple stacks).

# Docker hub

Docker Hub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.